(1)

DEFENSE COMMUNICATIONS AGENCY

DDN PROTOCOL HANDBOOK

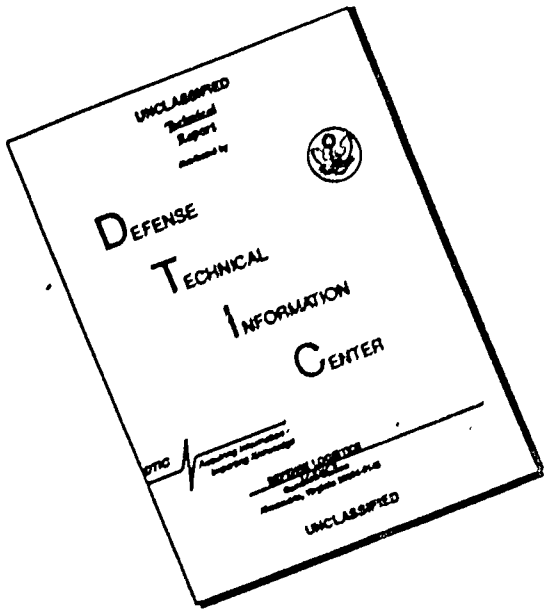
Volume Two

DARPA INTERNET PROTOCOLS

DECEMBER 1985

DDN

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Distribution Statement A Approved for public release Distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NIC 50004 NIC 50005 NIC 50006	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION SRI International DDN Network Information Ctr	6b. OFFICE SYMBOL (If applicable)
7a. NAME OF MONITORING ORGANIZATION Defense Data Network Program Management Office	
6c. ADDRESS (City, State, and ZIP Code) Menlo Park, CA 94025	7b. ADDRESS (City, State, and ZIP Code) McLean, VA 22102
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
DDN Protocol Handbook (3 vols.) (Unclassified)

12. PERSONAL AUTHOR(S)
Feinler, E.; Jacobsen, O.; Stahl, M.; Ward, C., eds.

13a. TYPE OF REPORT	13b. TIME COVERED FROM ___ TO ___	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
		851200	ca. 3000

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Defense Data Network; DDN; DDN protocols; Network protocols; TCP/IP; Transmission Control Protocol/ Internet Protocol; Network subscriber access

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The DDN (Defense Data Network) Protocol Handbook is a three volume work which gathers together many documents of interest to those wishing to implement the Department of Defense suite of protocols on various computers to be attached to the DDN. The official Military Standard communication protocols in use on the DDN are included, as are several ARPANET (Advanced Research Projects Agency Network) research protocols which are currently in use, and some protocols currently undergoing review. Tutorial information and auxiliary documents are also included. In addition to its use as a source guide for protocol implementation pruposes, the handbook can be used by vendors wishing to make their products compatible with DoD needs, by researchers wishing to improve the protocols, and by implementors of local area networks (LANs) wishing their networks to interact with the DDN.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL J. Redfield	22b. TELEPHONE (Include Area Code) (415) 859-6187
22c. OFFICE SYMBOL NJ 292	

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. SECURITY CLASSIFICATION OF THIS PAGE
All other editions are obsolete

E. Redfield
SRI International
Network Information Center
[illegible] Ravenswood Ave.
Menlo Park, CA 94025

DEFENSE COMMUNICATIONS AGENCY

DDN PROTOCOL HANDBOOK

Volume Two

DARPA INTERNET PROTOCOLS

DECEMBER 1985

Editors:

Elizabeth J. Feinler
Ole J. Jacobsen
Mary K. Stahl
Carol A. Ward

Additional copies of this document may be obtained from the SRI International Information Center, SRI International, 333 Ravenswood Avenue, Room EJ291, Menlo Park, CA 94025-3493 or from the Defense Technical Information Center (DTIC), Cameron Station, Alexandria, VA 22314.

ACKNOWLEDGEMENTS

The DDN Protocol Handbook was compiled by the DDN Network Information Center (NIC) for the Defense Data Network Program Management Office (DDN PMO) of the Defense Communications Agency (DCA) under contract number DCA-200-83-0025.

The editors are indebted to the authors of the many RFCs included in the body of this document. Special thanks goes to the following people for their invaluable support and contributions toward the production of the Handbook: Jonathan B. Postel from the University of Southern California Information Sciences Institute; Michael L. Corrigan, John Claitor, and John R. Walker from the DDN PMO; Edward Brady, Philip S. Selvaggi, Edward A. Cain from the Defense Communications Engineering Center; Chris J. Perry and Michael A. Padlipsky from the Mitre Corporation; and Diane Fountaine from the Office of the Assistant Secretary of Defense for Command, Control, Communications and Intelligence (C^3I).

TABLE OF CONTENTS - VOLUME TWO

SECTION 1. INTRODUCTION TO VOLUME TWO

Volume Two, of the three-volume DDN Protocol Handbook, contains protocol information pertaining to the DARPA Internet community. It includes specifications for all current official DARPA Internet protocols plus auxiliary information needed to implement the protocols. The review process for acceptance of a new protocol for use by the DARPA Internet research community is described, as is the administrative structure of the DARPA Internet Research program.

Some of the protocols in this volume have now been adopted as DoD Military Standards (MIL STDs). The MIL STD versions can be found in Volume One. Note that the specification style is different for the two versions of protocols, with more emphasis being put on descriptive detail in the case of the DARPA Internet documents. This makes the DARPA documents helpful for researchers who are interested in the development of the protocol, or who are planning to write protocol implementation programs.

Information included in this volume of the Protocol Handbook was supplied to the DDN Network Information Center (NIC) by the Deputy Chairman of the Internet Advisory Board (IAB), on behalf of the Defense Advanced Research Projects Agency (DARPA). The post of Deputy Chairman is currently held by Dr. Jonathan B. Postel, University of Southern California, Information Sciences Institute (POSTEL@USC-ISIF.ARPA).

Please note that many of the protocols and RFCs that make up the various sections of this Handbook have previously been printed as separate documents. Consequently, some of them have their own separate page numbering. So that the reader can easily distinguish between the two sets of paging, the page numbering for the Handbook as a whole is centered below the footer line, whereas any page numbering specific to an individual document is printed above the footer line.

SECTION 2. BACKGROUND

2.1 A Brief History of the ARPANET

The ARPANET was the first packet-switched store-and-forward host-to-host digital computer network. It originated as a purely experimental network in late 1969 under a research and development program sponsored by the U.S. Department of Defense. The network was designed to provide efficient communication between heterogeneous computers so that hardware, software, and data resources could be conveniently shared by a wide community of users. Today the ARPANET provides support for a large number of government projects with an operational network of several hundred nodes and host computers. The three main services offered by the network are MAIL, FILE TRANSFER, and TELNET (the ability to remotely log in to one computer from another). A number of other services are offered by special purpose programs which allow the implementation of "distributed computer systems".

The ARPANET has evolved from a single, packet-switched network, using Interface Message Processors (IMPs) and leased telephone circuits as the network "backbone", to an "internet", a collection of many different kinds of networks tied together by means of "gateways". The DARPA Internet today provides access to several hundred Local Area Networks (LANs) as well as other public and private data networks in many parts of the world. Interoperation of different types of networks is now a major part of the research activity in the DARPA Internet Community. This fact is reflected in the DARPA Internet protocols.

In 1983, the existing ARPANET was administratively divided into two unclassified networks, ARPANET and MILNET, to meet the growing need for an unclassified operational military network as well as the need for a research and development network. The physical split into separate networks was completed in September 1984. Each network now has its own backbone, and is interconnected through controlled gateways to the other. The ARPANET serves primarily as an experimental research and development network, while the MILNET functions as an operational military network for non-classified traffic. Communication and resource sharing between them continue, but are subject to administrative restrictions.

2.2 Management of the ARPANET

The DDN, including ARPANET, is operated for the DoD by the Defense Communications Agency. For an overview of the management structure for ARPANET, see Figure 2-1.

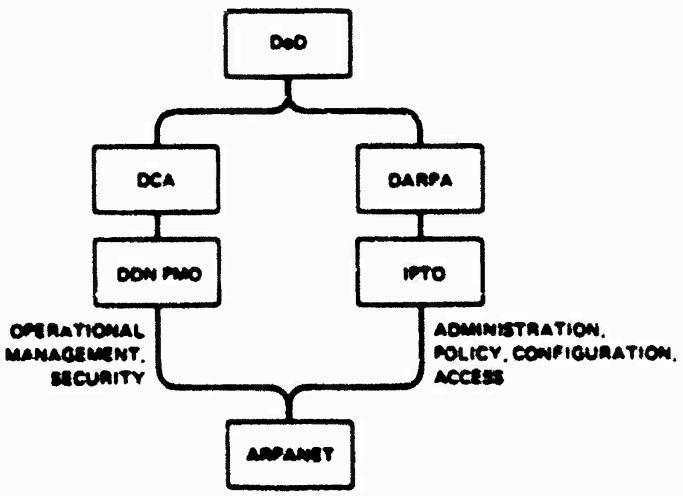


Figure 2-1: Management of the ARPANET

Individuals who have a requirement to attach equipment to the ARPANET should also consult the *ARPANET Information Brochure* which is available from the NIC or from DTIC.

2.2.1 DARPA/IPTO

DARPA's Information Processing Techniques Office (IPTO) is dedicated to developing advanced information processing and computer communications technologies for critical military and national security applications. The building of the ARPANET and development of its protocols was an IPTO program, which has evolved into what is now known as the Internet Research Program.

Through IPTO, DARPA sets policy for, and manages use of, the ARPANET. This is done within broad guidelines established for all DDN networks by the DDN PMO. It also funds the ARPANET, and funds research carried out on the ARPANET. It is important to emphasize that the DDN PMO operates and manages the ARPANET, including the node software and hardware, while DARPA pays the backbone operating costs, sets policy for the ARPANET, and approves access for DARPA-sponsored subscribers.

IEN 48

THE CATENET MODEL FOR INTERNETWORKING

Vint Cerf
DARPA/IPTO

July 1978

The Catenet Model for Internetworking

Introduction

The term "catenet" was introduced by L. Pouzin in 1974 in his early paper on packet network interconnection [1]. The U.S. DARPA research project on this subject has adopted the term to mean roughly "the collection of packet networks which are connected together." This is, however, not a sufficiently explicit definition to determine, for instance, whether a new network is in conformance with the rules for network interconnection which make the catenet function as confederation of co-operating networks. This paper attempts to define the objectives and limitations of the ARPA-internetworking project and to make explicit the catenet model on which the internetworking strategy is based.

Objectives

The basic objective of this project is to establish a model and a set of rules which will allow data networks of widely varying internal operation to be interconnected, permitting users to access remote resources and to permit intercomputer communication across the connected networks.

One motivation for this objective is to permit the internal technology of a data network to be optimized for local operation but also permit these locally optimized nets to be readily interconnected into an organized catenet. The term "local" is used in a loose sense, here, since it means "peculiar to the particular network" rather than "a network of limited geographic extent." A satellite-based network such as the ARPA packet satellite network therefore has "local" characteristics (e.g., broadcast operation) even though it spans many thousands of square miles geographically speaking.

A second motivation is to allow new networking technology to be introduced into the existing catenet while remaining functionally compatible with existing systems. This allows for the phased introduction of new and obsolescence of old networks without requiring a global simultaneous change.

Assumptions

One of the first questions which must be settled in a project of this sort is "what types of data networks should be included in the catenet model?" The answer to this question is rooted in the basic functionality of each candidate network. Each network is assumed to support the attachment of a collection of programmable computers. Our essential assumption is that any participating data network can carry a datagram containing no less than 1000

bits of data not including a local network header containing local control information. Furthermore, it is assumed that the participating network allows switched access so that any source computer can quickly enter datagrams for successive and different destination computers with little or no delay (i.e., on the order of tens of milliseconds or less switching time).

Under these assumptions, we can readily include networks which offer "datagram" interfaces to subscribing host computers. That is, the switching is done by the network based on a destination address contained in each datagram passing across the host to network interface.

The assumptions do not rule our virtual circuit interface networks, nor do they rule out very fast digital circuit switching networks. In these cases, the important functionality is still that a datagram can be carried over a real or virtual circuit from source to destination computer, and that the switching delay is below a few tens of milliseconds.

An important administrative assumption is that the format of an internet datagram can be commonly agreed, along with a common internet addressing plan. The basic assumption regarding datagram transport within any particular network is that the datagram will be carried, embedded in one or more packets, or frames, across the network. If fragmentation and reassembly of datagrams occurs within a network it is invisible for purposes of the catenet model. Provision is also made in the datagram format for the fragmentation of datagrams into smaller, but identically structured datagrams which can be carried independently across any particular network. No a priori position is taken regarding the choice between internal (invisible) fragmentation and reassembly or external (visible) fragmentation. This is left to each network to decide. We will return to the topic of datagram format and addressing later.

It is very important to note that it is explicitly assumed that datagrams are not necessarily kept in the same sequence on exiting a network as when they entered. Furthermore, it is assumed that datagrams may be lost or even duplicated within the network. It is left up to higher level protocols in the catenet model to recover from any problems these assumptions may introduce. These assumptions do not rule out data networks which happen to keep datagrams in sequence.

It is also assumed that networks are interconnected to each other by means of a logical "gateway." As the definition of the gateway concept unfolds, we will see that certain types of network interconnections are "invisible" with respect to the catenet model. All gateways which are visible to the catenet model have the characteristic that they can interpret the address

fields of internet datagrams so as to route them to other gateways or to destinations within the networks directly attached to (or associated with) the gateway. To send a datagram to a destination, a gateway may have to map an internet address into a local network address and embed the datagram in one or more local network packets before injecting it into the local network for transport.

The set of catenet gateways are assumed to exchange with each other at least a certain minimum amount of information to enable routing decisions to be made, to isolate failures and identify errors, and to exercise internet flow and congestion control. Furthermore, it is assumed that each catenet gateway can report a certain minimum amount of status information to an internetwork monitoring center for the purpose of identifying and isolating catenet failures, collecting minimal performance statistics and so on.

A subset of catenet gateways may provide access control enforcement services. It is assumed that a common access control enforcement mechanism is present in any catenet gateway which provides this service. This does not rule out local access control imposed by a particular network. But to provide globally consistent access control, commonality of mechanism is essential.

Access control is defined, at the catenet gateway, to mean "permitting traffic to enter or leave a particular network." The criteria by which entrance and exit permission are decided are the responsibility of network "access controllers" which establish access control policy. It is assumed that catenet gateways simply enforce the policy of the access controllers.

The Catenet Model

It is now possible to offer a basic catenet model of operation. Figure 1 illustrates the main components of the model. Hosts are computers which are attached to data networks. The host/network interfaces are assumed to be unique to each network. Thus, no assumptions about common network interfaces are made. A host may be connected to more than one network and it may have more than one connection to the same network, for reliability.

Gateways are shown as if they were composed of two or more "halves." Each half-gateway has two interfaces:

1. A interface to a local network.

2. An interface to another gateway-half.

One example is given of a gateway with three "halves" connecting networks A, B, and C. For modelling purposes, it is appropriate to treat this case as three pairs of gateway halves, each pair bilaterally joining a pair of networks.

The model does not rule out the implementation of monolithic gateways joining two or more nets, but all gateway functions and interactions are defined as if the gateways consisted of halves, each of which is associated with a specific network.

A very important aspect of this model is that no a priori distinction is made between a host/network interface and a gateway/network interface. Such distinctions are not ruled out, but they are not relevant to the basic catenet model.

As a consequence, the difference between a host which is connected to two networks and a monolithic gateway between networks is entirely a matter of whether table entries in other gateways identify the host as a gateway, and whether the standard gateway functionality exists in the host. If no other gateway or host recognizes the dual net host as a gateway or if the host cannot pass datagrams transparently from one net to the next, then it is not considered a catenet gateway.

The model does not rule out the possibility of implementing a gateway-half entirely as part of a network switching node (e.g., as software in an ARPANET IMP). The important aspect of gateway-halves is the procedure and protocol by which the half-gateways exchange datagrams and control information.

The physical interface between directly connected gateway halves is of no special importance. For monolithic gateways, it is typically shared memory or an interprocess communication mechanism of some kind; for distinct gateway halves, it might be HDLC, VDH, any other line control procedure, or inter-computer buss mechanism.

Hidden Gateways

No explicit network hierarchy is assumed in this model. Every network is known to all catenet gateways and each catenet gateway knows how to route internet datagrams so they will eventually reach a gateway connected to the destination network.

The absence of an explicit hierarchical structure means that some network substructures may be hidden from the view of the catenet gateways. If a network is composed of a hierarchy of internal networks connected together with gateways, these "hidden gateways" will not be visible to the catenet gateways unless the internal networks are assigned global network addresses and their interconnecting gateways co-operate in the global routing and network flow control procedures.

Figure 2 illustrates a simple network hierarchy. For purposes of, identification, the three catenet gateways have been labelled G(AX), G(BX) and G(CX) to indicate that these gateways join networks A and X, B and X and C and X, respectively. Only G(AX), G(BX) , and G(CX) are considered catenet gateways. Thus they each are aware of networks A, B, C and X and they each exchange routing and flow-control information in a uniform way between directly connected halves.

Network X is composed of three internal networks labelled u, v and w. To distinguish them from the catenet gateways, the "hidden gateways" of net X are labelled HG(nm) where "nm" indicate which nets the hidden gateways join. For example, HG(vw) joins nets v and w. The notation for HG is symmetric, i.e., HG(vw)=HG(wv).

Gateways G(AX), G(BX), G(CX) exchange connectivity and other flow control information among themselves, via network X. To do this, each gateway half must know an address, local to network X, which will allow network X to route datagrams from G(AX) to G(BX), for example.

From the figure, it is plain that G(BX) is really a host on network B and network v. But network v is not one of the globally recognized networks. Furthermore, traffic from G(AX) to G(BX) may travel from net u to net v or via nets u and w to net v. To maintain the fiction of a uniform network X, the gateway halves of G(AX), G(BX) and G(CX) attached to net X must be aware of the appropriate address strings to use to cause traffic to be routed to each catenet gateway on net X. In the next section, we outline a basic internet addressing philosophy which permits such configurations to work.

Local Gateways

Another element of the catenet model is a "local gateway" associated with each host. The local gateway is capable of reassembling fragmented internet datagrams, if necessary, and is responsible for encapsulation of internet datagrams in local network packets. The local gateway also selects internet gateways through which to route internet traffic, and responds to

routing and flow control advice from the local network and attached catenet gateways.

For example, a local gateway might encapsulate and send an internet datagram to a particular gateway on its way to a distant network. The catenet gateway might forward the packet to another gateway and send an advisory message to the local gateway recommending a change in its catenet gateway routing table. Local gateways do not participate in the general routing algorithm executed among the catenet gateways.

Internet Addressing

The basic internet datagram format is shown in Figure 3. By assumption, every network in the catenet which is recognized by the catenet gateways has a unique network number. Every host in each network is identified by a 24 bit address which is prefixed by the network number. The same host may have several addresses depending on how many nets it is connected to or how many network access lines connect it to a particular network.

For the present, it is assumed that internet addresses have the form: Net.Host. "Net" is an 8 bit network number. "Host" is a 24 bit string identifying a host on the "Net," which can be understood by catenet and possibly hidden gateways.

The catenet gateways maintain tables which allow internet addresses to be mapped into local net addresses. Local gateways do likewise, at least to the extent of mapping an "out-of-network" address into the local net address of a catenet gateway.

In general, catenet gateways maintain a table entry for each "Net" which indicates to which gateway(s) datagrams destined for that net should be sent. For each "Net" to which the gateway is attached, the gateway maintains tables, if necessary, to permit mapping from internet host addresses to local net host addresses. The typical case is that a gateway half is connected to only one network and therefore only needs to maintain local address information for a single network.

It is assumed that each network has its own locally specific addressing conventions. To simplify the translation from internet address to local address, it is advantageous, if possible, to simply concatenate a network identifier with the local "host" addresses to create an internet address. This strategy makes it potentially trivial to translate from internet to local net addresses.

More elaborate translations are possible. For example, in the case of a network with a "hidden" infrastructure, the "host" portion of the internet address could include additional structure which is understood only by catenet or hidden gateways attached to that net.

In order to limit the overhead of address fields in the header, it was decided to restrict the maximum length of the host portion of the internet address to 24 bits. The possibility of true, variable-length addressing was seriously considered. At one point, it appeared that addresses might be as long as 120 bits each for source and destination. The overhead in the higher level protocols for maintaining tables capable of dealing with the maximum possible address sizes was considered excessive.

For all the networks presently expected to be a part of the experiment, 24 bit host addresses are sufficient, even in cases where a transformation other than the trivial concatenation of local host address with network address is needed to form the 32 bit internet host address.

One of the major arguments in favor of variable length "addressing" is to support what is called "source-routing." The structure of the information in the "address" really identifies a route (e.g., through a particular sequence of networks and gateways). Such a capability could support ad hoc network interconnections in which a host on two nets could serve as a private gateway. Though it would not participate in catenet routing or flow control procedures, any host which knows of this private gateway could send "source-routed" internet datagrams to that host.

To support experiments with source routing, the internet datagram includes a special option which allows a source to specify a route. The option format is illustrated in Figure 4. The option code identifies the option and the length determines its extent. The pointer field indicates which intermediate destination address should be reached next in the source-selected route.

Source routing can be used to allow ad hoc network interconnections to occur before a new net has been assigned a global network identifier.

In general, catenet gateways can only interpret internet addresses of the form Net.Host. Private gateways could interpret other, local addresses for desired destinations. If a source knew the local addresses of each intermediate private gateway, it could construct a source-route which is the concatenation of the local addresses of each intermediate host.

Local and internet addresses could be inter-mixed in a single source route as long as catenet gateways only had to interpret full internet addresses when the source-routed datagram appeared for servicing. Private gateways could interpret local and internet addresses, as desired.

Since the source or destination of a source-routed datagram may not have an internet address, it may be necessary to provide a return route for replies. This might be done by modifying the content of the original route to contain "back Pointer" to intermediate destinations. Note that the local address of a private gateway in one network is usually different from its local address in the adjacent network.

Typically, a source would create a route which contains first the internet address of the host or gateway nearest to the desired destination. The next address in the route would be the local address of the destination. Figure 5 illustrates this notion. Host A.a wants to communicate with host Z. But Z is not attached to a formally recognized network.

To achieve its goal, host A.a can emit source-routed packets with the route: "B.y, Z." B.y identifies the host (private gateway) between net B and the new network as the first intermediate stop. The private gateway uses the "Z" information to deliver the datagram to the destination. When the datagram arrives, its route should contain "y,A.a" if the private gateway knows how to interpret A.a or "y, W, A.a" if the private gateway only knows about addresses local to network B.

Other Issues

The catenet model should provide for error messages originating within a network to be carried usefully back to the source. A global encoding of error messages or status messages is needed.

It is assumed that the gateway halves of a given network have a common status reporting, flow and congestion control mechanism. However, the halves on different nets may operate differently. There should be a defined interface between gateway halves which permits internet flow, congestion and error control to be exercised.

A gateway monitoring center [3] is postulated which can collect, correlate and display current gateway status. Such a center should not be required for the internet protocols to function, but could be used to manage an internet environment.

Accounting, accountability and access control procedures should be defined for the global catenet.

References

1. Pouzin, L., "A Proposal for Interconnecting Packet Switching Networks," Proceedings of EUROCOMP, Bronel University, May 1974, pp. 1023-36.

2. Postel, J. "Internetwork Datagram Protocol Specification," Version 4, Internetwork Experiment Note No. 41, Section 2.3.2.1, Internet Experiment Notebook, June 1978.

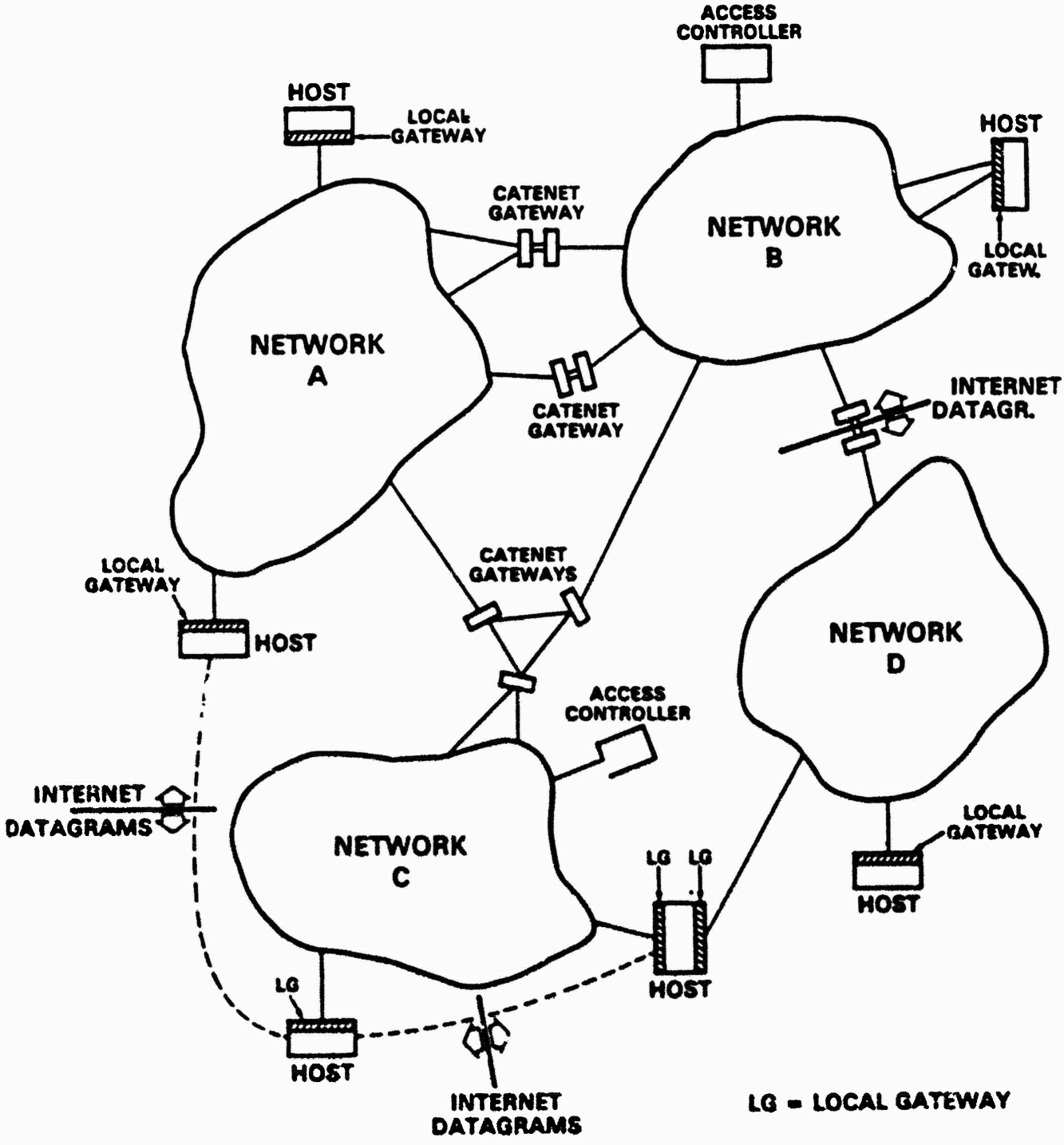
3. Davidson, John, "CATENET MONITORING AND CONTROL: A model for the Gateway Component," IEN #32, Section 2.3.3.12, Internet Notebook, April 1978.

NOTE: The figures are not included in the online version. They may be obtained from:

Jon Postel
USC - Information Sciences Institute
Suite 1100
4676 Admiraly Way
Marina del Rey, California 90291

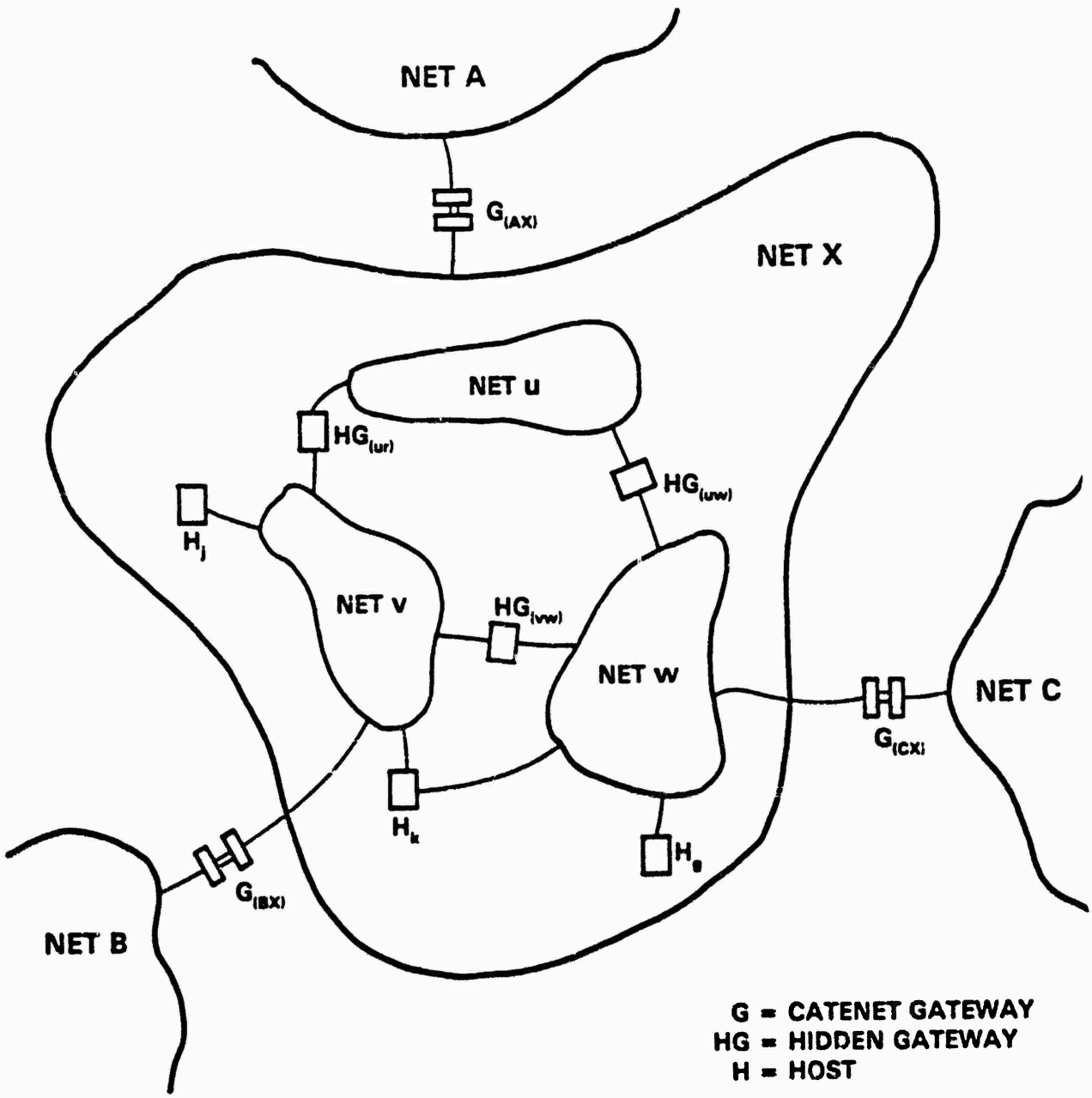
Phone: (213) 822-1511

ARPANET Mailbox: POSTEL@ISIF



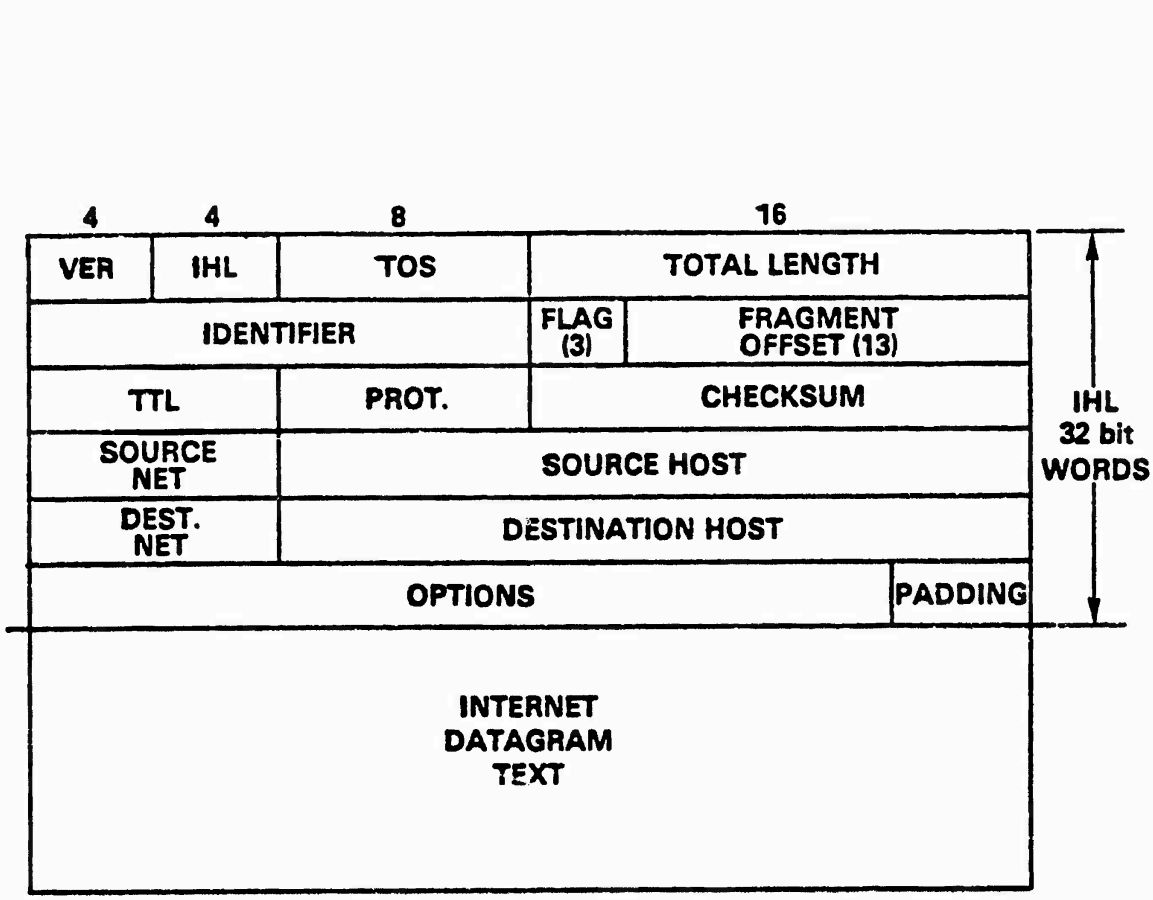
BASIC CATENET MODEL

Figure 1



HIDDEN GATEWAYS IN HIERARCHICAL NETWORKS

Figure 2



VER = VERSION TYPE

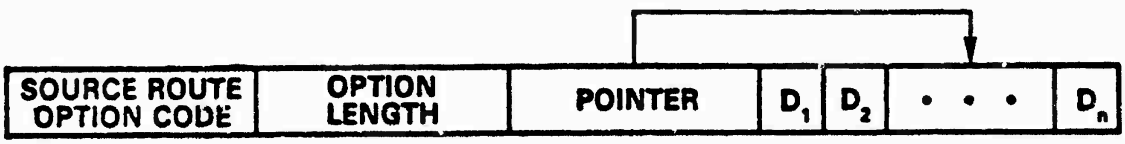
IHL = INTERNET HEADER LENGTH IN 32 bit WORDS

TOS = TYPE OF INTERNET SERVICE DESIRED
e.g. "LOW DELAY", "LOW COST", "HIGH BANDWIDTH", "HIGH RELIABILITY", "DON'T DISCARD".

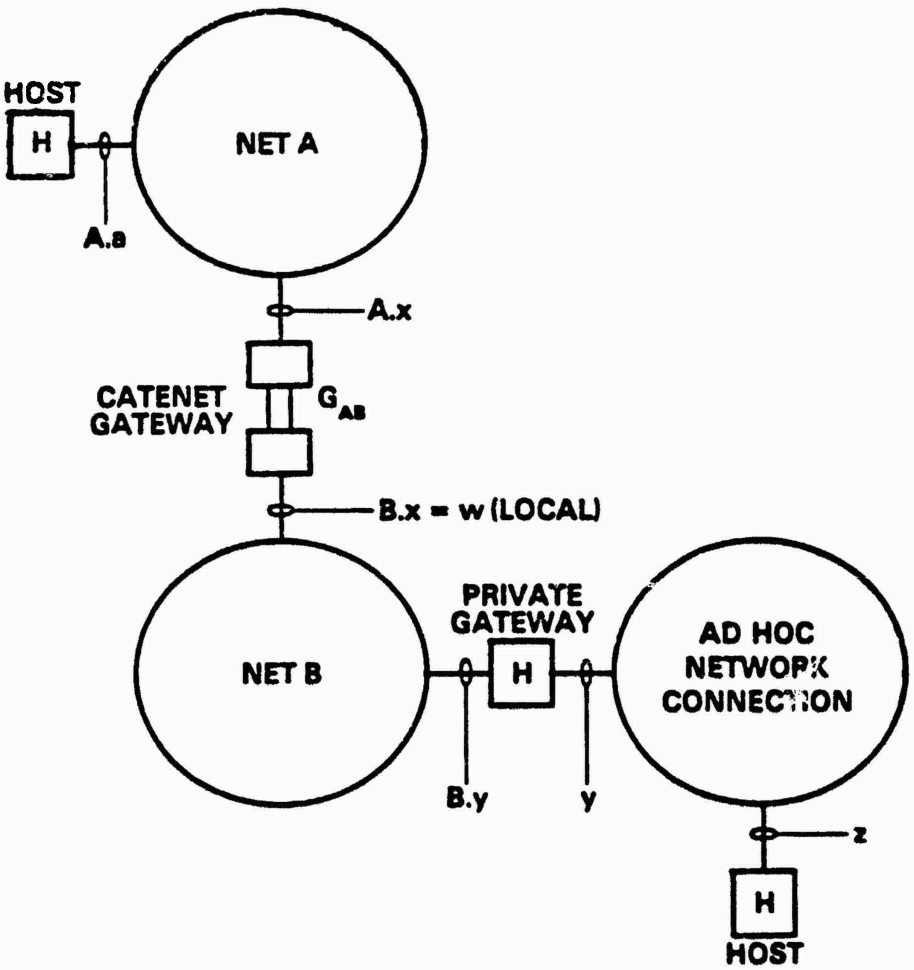
FLAG = CONTROL INDICATIONS SUCH AS
"OPTIONS PRESENT","MORE FRAGMENTS".

PROT = PROTOCOL IDENTIFIER (e.g. TCP, REAL-TIME)

INTERNET DATAGRAM FORMAT
FIGURE 3



SOURCE ROUTE OPTION FORMAT
FIGURE 4



PRIVATE GATEWAYS AND SOURCE ROUTING
FIGURE 5

The DARPA Internet Protocol Suite

Dr. Barry M. Leiner
Mr. Robert Cole
Dr. Jon Postel
Dr. David Mills

Dr. Leiner is with the Defense Advanced Research Projects Agency, Arlington, VA 22209. Mr. Cole is with University College, London. Dr. Postel is with University of Southern California, Information Sciences Institute, Los Angeles, CA. Dr. Mills is with M/A-COM Linkabit, McLean, VA.

The views represented in this paper are those of the authors and do not necessarily represent those of DARPA, DoD or the US Government. This research has been supported under a number of DARPA contracts.

Table of Contents

List of Figures

Abstract

The military requirement for computer communications between heterogeneous computers on heterogeneous networks has driven the development of a standard suite of protocols to permit such communications to take place in a robust and flexible manner. These protocols support an architecture consisting of multiple packet switched networks interconnected by gateways. The DARPA experimental internet system consists of satellite, terrestrial, radio and local networks, all interconnected through a system of gateways and a set of common protocols.

In this paper, the suite of protocols supporting this internet system is described.

1. Introduction

The rapid proliferation of computers and other signal processing elements throughout the military coupled with their need for reliable and efficient exchange of information has driven the development of a number of computer networking technologies. The differences in both requirements and environments for these networks has resulted in different network designs. Furthermore, differences in requirements coupled with changing technologies has resulted in many different computer types being fielded. These different computers, although located on different networks, still have a requirement to communicate with each other.

Beginning with the ARPANET (the first packet-switched network) [5], DARPA (Defense Advanced Research Projects Agency) has sponsored the development of a number of packet switched networking technologies designed to provide robust and reliable computer communications. These networks have included the primarily land-line based ARPANET, packet radio networks [11, 12], and satellite networks [10, 16]. In addition, the use of other available technologies such as local area networks and public data networks has also been investigated.

As mentioned above, there is a significant need to be able to interconnect these various packet-switched networks so that computers on the various networks can communicate. Furthermore, this communication must be reliable and robust, making use of whatever communication facilities are available to accomplish end-to-end connectivity. To this end, DARPA initiated a program to investigate the issues in interconnection of different packet-switched networks. This effort has resulted in an architecture and set of protocols to accomplish this robust system of interconnected networks.

In this paper, the current status of the DARPA Experimental Internet System (the Internet for short) in terms of the architecture and set of protocols is described. The first section gives an overview of the internet architecture, describing the key elements of the system and their relation to each other. Following that, the set of protocols is

described. Next, experiences in the test and development of the internet system are discussed. Finally, a summary and conclusions is given.

Throughout the reading of this paper, one should keep in mind that the internet system is still under development. Although a number of protocols have been standardized within the research community and are either currently Defense Department standards [6, 7] or in the process of becoming standards, the internet system is constantly evolving with new functions and new protocols being developed to meet the ever-changing military requirements.

2. Architectural Overview

The DARPA Internet protocol suite is designed to support communication between heterogeneous hosts on heterogeneous networks as shown in Figure 2-1. A number of packet-switched networks are interconnected with gateways. Each of these networks are assumed to be designed separately in accordance with some specific requirements and environmental considerations (e.g. radio line-of-sight, local cable networks, etc.). However, it is assumed that each network is capable of accepting a packet of information (data with appropriate network headers) and delivering it to a specified destination on that particular network. It is specifically not assumed that the network guarantees delivery of the packet. Specific networks may or may not have end-to-end reliability built into them.

Thus, two hosts connected to the same network are capable of sending packets of information between them. Should two hosts on different networks wish to communicate, the source host would send packets to the appropriate gateway, which then would route each packet through the system of gateways and networks until it reaches a gateway connected to the same network as the final host. At this point, the gateway sends the packet to the destination host.

The internet system can therefore be viewed as a set of hosts and gateways interconnected by networks. Each network can act as a link between the gateways and hosts residing on it, and a gateway looks like a typical host to any network. Packets

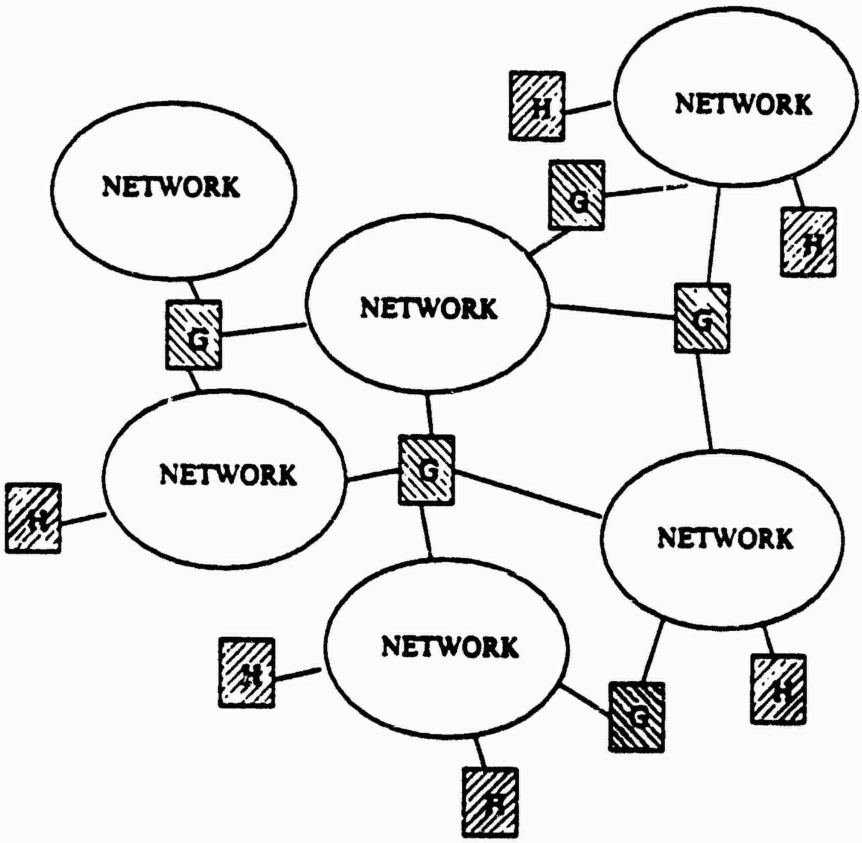


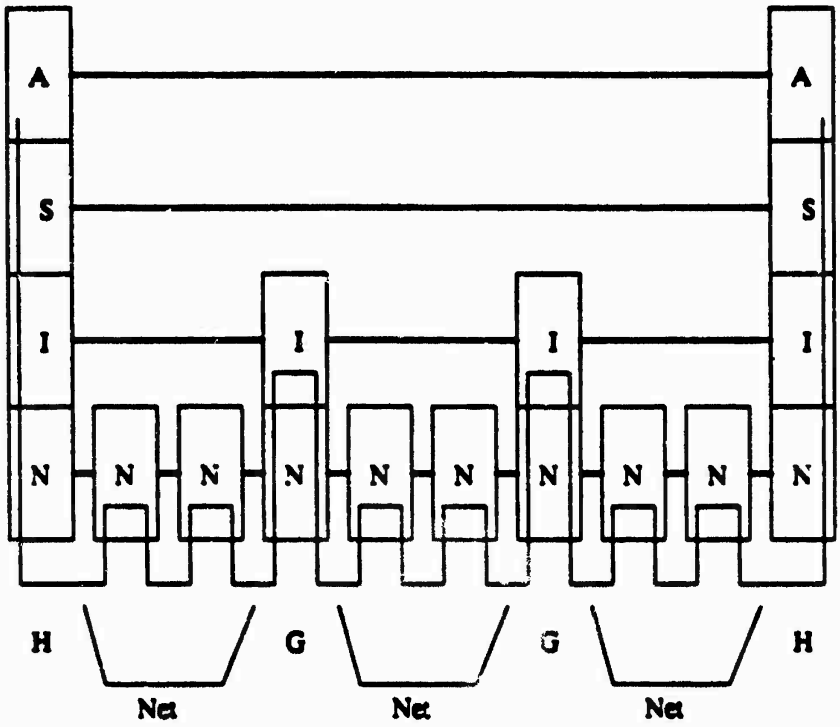
Figure 2-1: An Internet System

are suitably routed between the hosts and gateways so as to use the correct networks to traverse the system from source to destination.

Taking this view, it is clear that the service required from each network is simply the ability to carry packets between attached hosts. Gateways attach to networks as hosts. Since mechanisms must be built into the system to provide end-to-end reliability even in the face of network failures (by, for example, routing packets through alternate networks), the only service required from the network is a datagram delivery service. This means that, given a packet with a destination address on the network, the network will attempt to deliver the packet to that destination.

The overall internet architecture can therefore be described as four layers. At the bottom layer, individual networks and mechanisms for connecting hosts to those networks are present. At the next layer, the internet layer, are the mechanisms for connecting the various networks and gateways into a system capable of delivering packets from source to destination. At the next layer, end-to-end communication services are built in, including mechanisms such as end-to-end reliability and network control. Finally, at the upper layer, applications services are provided such as file transfer, virtual terminal and mail.

To describe the internet architecture, it is useful to trace a typical packet as it traverses the system from source host to destination host. Figure 2-2 shows the flow of a packet through the internet system. Data originates at an application layer and needs to be transported to the corresponding layer at the other end. Using the appropriate utility protocol and transport protocol, it packages the data into internet packets. These packets are treated as data in the transmission through each of the individual networks, so that internet packets move from host to gateway, from gateway to gateway, and from final gateway to destination host, in each case looking like just a normal network packet on each network. The interface between the network and the hosts and gateways are defined by the individual networks, and the hosts and gateways are responsible for packaging the internet packets into network packets.



N = Network Protocols
I = Internet Protocol
S = Service Protocols
A = Application Protocols

H = Host computer
G = Gateway

Figure 2-2: The Layered Protocol Architecture

It should be noted at this point that this approach, known as encapsulation, has some distinct advantages in the interconnection of networks. It is never necessary to build a "translation" device mapping one network protocol into another. The internet layer provides a common language for communication between hosts and gateways, and can be treated as simple data by each network. This eliminates the "N x N" problem of building translating devices for each possible pair of networks, as it is only necessary to build the interface between the internet layer and each individual network. Thus, hosts only need know about their local network and the internet protocols.

3. The Internet Protocol Suite

To implement the above architecture, a set of protocols has been developed within the DARPA research community. These protocols have been developed with the above architecture in mind (namely a layered architecture with certain functionalities in the host-host protocols, and others in the gateways and networks). As additional functionalities have been required, either new protocols or modifications to existing ones were developed. It is anticipated that this will continue and therefore the description of the protocol suite given here represents the current state rather than a permanent set of "standards".

Figure 3-1 shows the various protocols currently being used and their relation to one another.

3.1. Network Layer Protocols

At the lowest levels are the Physical, Link and Network protocols. These correspond to the Network layer mentioned above and provide the means for a host accessing the network. (Note that these normally describe the protocol for a host to connect to a network and not the protocol used in the network itself, i.e. between the switches of the network. That is of concern only to the network designer.) The key point to be remembered here is that the internet system accepts networks as they are and utilizes them in an interconnected system of networks to achieve the required end-to-end communication capability. Thus, the primary areas of concern to the internet system

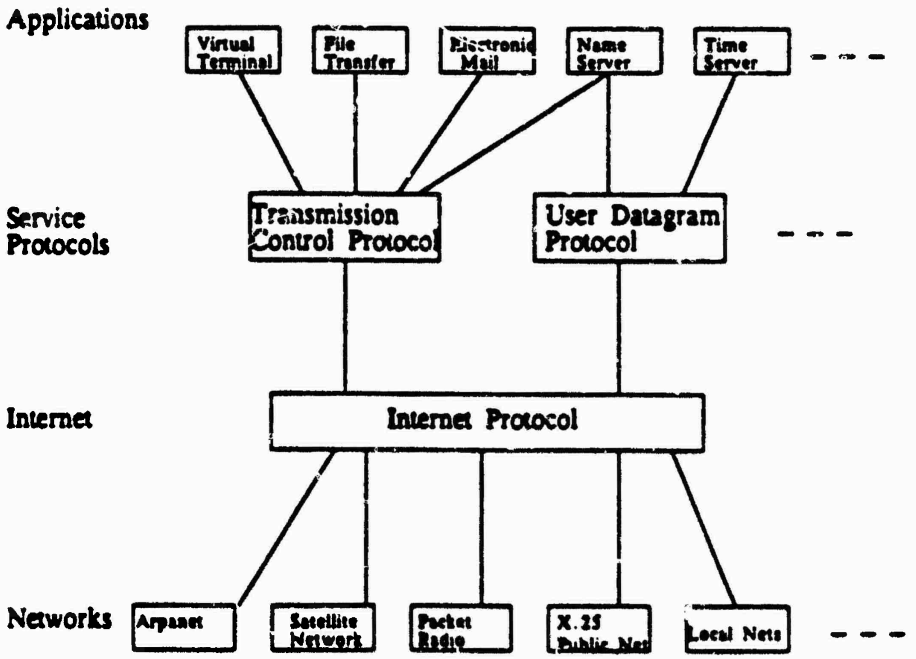


Figure 3-1: The Internet Protocol Suite

are the interface to the network and the performance (e.g. throughput and delay) offered by the network.

3.2. Internet Protocol

The Internet Protocol [2, 6] is the lynch pin of the internet system. It is this protocol that insulates applications programs from needing to know specifics about the networks. The Internet Protocol (IP) unifies the available network services into a uniform internet datagram service. The IP includes such functions as a global addressing structure, provision for type of service requests (to allow selection of appropriate network level services where required), and provision for fragmentation of packets and reassembly at the destination host in the event that a packet's size is larger than the maximum packet size of the network through which the packet is about to traverse. The decision on what to put into IP and what to leave out was made on the basis of the question "Do gateways need to know it?". The key feature of IP is the Internet Address, a address scheme independent of the addresses used in the particular networks used to create the Internet.

As can be seen from Figure 2-2, the IP is used for communication between hosts and gateways, between gateways themselves, and between hosts on an end-to-end basis. It allows hosts to send packets through the internet system without regard to the network on which the destination host resides, by having the host send the packet to a gateway on the same network as the source host and letting the gateways take responsibility for determining how to deliver the packet to the final destination network and thereby destination host. Thus, the IP is critical to the proper operation of the internet system and the gateways in particular.

3.3. Service Protocols

The internet protocol and layer provides an end-to-end datagram delivery service, permitting a host to inject a packet into the internet system and have it delivered with some degree of confidence to the desired destination. The customer application, however, typically requires a specific level of service. This may involve specification of

reliability, error rate, delay, etc. or some combination of those characteristics. Rather than have each application develop its own end-to-end service protocols, it is desirable to have a number of standard services available upon which applications can build.

Currently, the DARPA experimental internet system has two standard service protocols, the Transmission Control Protocol (TCP) [7] and the User Datagram Protocol (UDP) [17]. Other protocols are likely to be developed at this level.

In addition to end-to-end service protocols, there is a requirement for control of the internet system. An adjunct to the IP has been developed called the Internet Control Message Protocol (ICMP) [19] to serve this need.

3.3.1. Transmission Control Protocol

One of the prime uses for computer communication networks is the ability to reliably transmit and receive files and electronic mail. The characteristics of such use is the necessity to pass a fairly large amount of data (typically more than would fit into a single network packet) reliably and be able to reconstruct the data in sequence. To support such internet services, the TCP was developed.

TCP provides an end-to-end reliable data stream service. It contains mechanisms to provide reliable transmission of data. These mechanisms include sequence numbers, checksums, timers, acknowledgments, and retransmission procedures. The intent of TCP is to allow the design of applications that can assume reliable, sequenced delivery of data.

3.3.2. User Datagram Protocol

Many applications do not require a reliable stream service. Sometimes, the basic datagram service of the internet is sufficient for applications if enhanced by such services as multiplexing different addresses onto the same IP address and checksumming for data integrity. The UDP provides these services and permits individual datagrams to be sent between hosts. This supports applications requiring such a transaction-oriented service.

3.3.3. Internet Control Message Protocol

In systems as large and complex as the Internet, it is necessary to have monitoring and control capabilities, permitting hosts to interact with gateways, as well as both interacting with internet monitoring and control centers. The ICMP provides the facility to carry out this control activity. It includes functions such as redirect messages to permit gateways to notify hosts that they should send packets to a different gateway as well as error reporting.

3.4. Application Protocols

Clearly, the purpose of the Internet system is to provide host to host and user to host computer communications service, thereby supporting the required applications. To accomplish this, the communicating hosts must agree on the protocol to be used for each application. A number of application protocols have been agreed upon in the DARPA Experimental Internet System, ranging from the very basic terminal access protocol to permit timesharing over the internet through the provision of such services as name servers and time servers.

3.4.1. Remote Terminal Protocol

TELNET [21] is the remote terminal access protocol in the DARPA Protocol Suite. TELNET allows the use of a terminal on one host with a program on another host. TELNET is based on three ideas: a network virtual terminal, negotiated options, and the symmetry of processes. TELNET is built on the services provided by TCP.

The network virtual terminal idea is used to define an imaginary terminal as the standard terminal. Then all real terminals are mapped by the TELNET implementations into or out of this imaginary standard. All the data traversing the Internet in TELNET applications is in terms of the imaginary standard terminal.

The negotiated options idea calls for a base level of capability as the default operation. Then enhancements may be negotiated via the exchange of requests between the two hosts. One nice feature of this mechanism is that a request can be rejected with out needing to know the semantics of the request.

The symmetry of processes suggests that the TELNET protocol should work the same both ways. That the protocol is mostly used for connecting Terminals to remote programs should not drive the protocol to be too specialized to that. It should also work to link two terminals, or to support process to process communication.

3.4.2. File Transfer Protocol

The File Transfer Protocol (FTP) [18] is based on a model of files having a few attributes, and a mechanism of commands and replies. The command and reply mechanism is used to establish the parameters for a file transfer and then to actually invoke the transfer. Like TELNET, FTP runs over TCP and thus assumes the service level provided by TCP.

3.4.3. Mail Transfer Protocol

An important use of computer networks is the support of electronic mail. In fact, one could attribute the success of the DARPA packet-switching research in large part to the availability of electronic mail facilities (first over the ARPANET and then over the Internet) to the researchers involved in the effort.

The Simple Mail Transfer Protocol (SMTP) [20] is similar to the FTP protocol in that it uses the same mechanism of commands and replies. The SMTP is simpler than the FTP though, in that the data exchanged is restricted to just one of the many possible combination of attributes allowed under FTP. The main concerns in the SMTP protocol are the provision for negotiating the recipients of a message, and confirming that the receiving host has taken full responsibility for the message. Like FTP, SMTP is built on TCP services.

3.4.4. Other Application Protocols

To illustrate some of the other application protocols that are available as part of the Internet, we describe two simple applications; a time server and a name server.

The time server [22] provides a very simple service that returns the time of day when ever it receives a request. This service may be implemented either on TCP or on UDP. On TCP, if a TCP connection is opened to the server, the server sends the time of day

and closed the connection. On UDP, if the server received a datagram, the server sends back a datagram carrying the time of day.

In order that users not be required to know the address of each internet host and to facilitate the movement of hosts to different addresses as part of normal network operations, a host name server [15] is part of the Internet. The host name to address lookup service is a transaction style service implemented on UDP. It expects to receive datagrams containing the name of an Internet host (e.g., USC-ISIF). When such a datagram arrives it adds the Internet address to the information and sends back a datagram carrying all that information (e.g., USC-ISIF = 10.2.0.52).

3.5. Gateway Protocols

As mentioned in the architectural overview, packets flow through the system through the use of gateways located between the networks. Thus, it is necessary that the gateways communicate with each other, both for passing data packets and for accomplishing the control of the internet, as such control is fully distributed to the gateways.

Datagrams are exchanged between networks via gateways, each of which belongs to one of several Autonomous Systems (AS). The gateways of each AS operate an Interior Gateway Protocol (IGP) in order to exchange network reachability and routing information within the AS; however, each AS may operate a different IGP suited to its architecture and operating requirements. The Gateway-Gateway Protocol (GGP), used for some time in the present Internet system, is an example of an IGP. The Exterior Gateway Protocol (EGP) is operated between selected gateways in each AS in order to exchange network reachability and routing information. Each gateway operating EGP or an IGP maintains a data base that selects the next gateway hop on the path to each destination network, of which there are now over 65 in the Internet system.

All gateways support the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP), which are datagram protocols requiring only minimal state storage in the gateway itself. IP support includes fragmentation, for those networks that require it,

along with several options including an explicit gateway routing override for special applications. ICMP support provides notification messages to the sender in cases of misrouted traffic, excessive flows and special maintenance messages.

4. Summary of Experiences

It cannot be over emphasized that the system described in this paper is not merely a set of standards but rather has been in operation and used on a daily basis supporting research in networking, command and control and other areas of computer science for over a decade. Figure 4-1 shows a sample indicating the breadth and heterogeneity of the Internet system. The system consists of land-line networks such as the ARPANET and X.25 public networks, several phases of packet radio networks, a number of local area networks and two different satellite packet switched networks. There are currently roughly 100 networks and 60 gateways, all interconnected into a unified system to provide the robust and reliable computer communications service required by both military and commercial users.

The Internet has been used to support a number of applications and experiments. Interestingly enough, due to its experimental nature, perhaps its most important use in the past decade has been the support of research into networking and other computer science areas.. By permitting the easy and rapid exchange of information (through both electronic mail and file transfers) as well as permitting the distributed development of software, rapid progress in these fields has been encouraged and facilitated.

The Internet has also been used to explore the implications of advanced computer communications technologies on military concepts and doctrine. In cooperation with the US Army, a testbed has been established at Ft Bragg, NC, which is investigating the application of advanced communications and distributed processing technologies in the support of Army concepts in distributed command and control [8]. In cooperation with the Strategic Air Command (SAC), the Defense Communications Agency (DCA) and Rome Air Development Center (RADC) of the Air Force, a testbed has been established at Offutt AFB, NE, to investigate the use of the Internet technology to

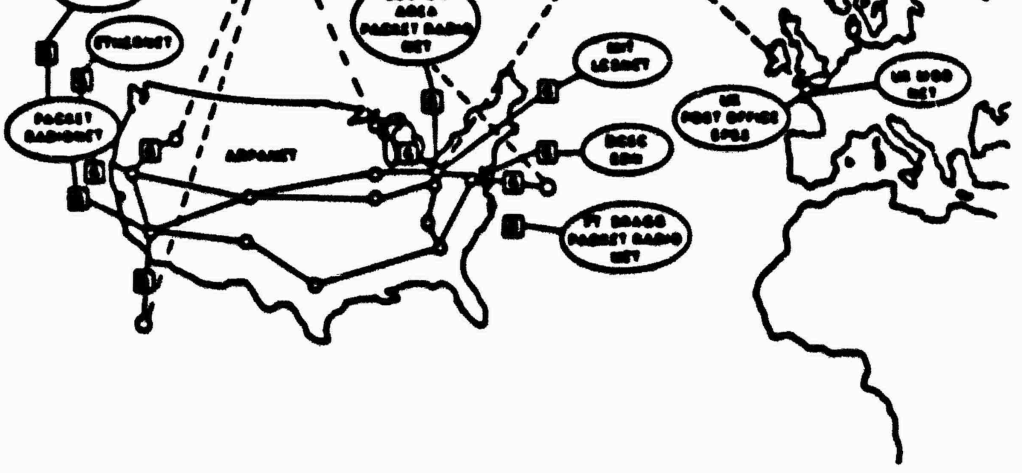
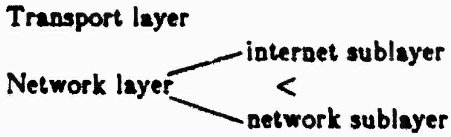


Figure 4-1: The DARPA Experimental Internet System

support strategic reconstitution efforts [8]. The Internet system provides the communications heart of a joint activity between the United States, United Kingdom, Germany, Norway and Canada investigating command and control interoperability. In addition, a number of experiments have been carried out with the US Navy using the Internet to demonstrate distributed command and control technologies. Clearly, none of these activities could have been performed with such effectiveness if it were not for the Internet system providing a unified and interoperable communication structure.

At present the International Standards Organization (ISO) is discussing a proposal to use datagrams as the main mechanism for inter-networking. The internetwork protocols will fit into a sublayer at the top of the network layer, just below the transport layer.



ISO have adopted X.25 as their main network sublayer protocol and have proposed their own protocol for the transport layer [23, 9]. The DARPA TCP is functionally similar to the ISO proposal for a transport protocol and can be considered equivalent.

Two groups are currently using the TCP as a transport protocol, the IP as an internet protocol, and X.25 as the network protocol in a manner which mirrors the ISO proposals. Both groups use the X.25 network as only one component of the path between the hosts, other networks include various local area networks and the other constituent networks of the DARPA Internet.

The CSNET group uses TELENET to provide connections between a number of hosts in Computer Science Departments throughout the US [4]. The second group is a number of European research sites, the main user being University College London (UCL).

UCL provides a relay service for mail and remote login that enables US and UK research workers to access each others facilities [1]. A single international X.25 connection is used to connect hosts at UCL, in England, to various Internet hosts in the US [3]. The primary protocols used on the international and US sides are TCP and IP. These are carried on the international public X.25 services.

Another effective use of the Internet system has turned out to be the measurement of network performance. TCP and IP can be used in network and inter-network measurements in a particularly effective manner. The protocols give two advantages:

1. The same protocols can be used over a number of networks and therefore different types of networks. This can allow comparison of network media.

2. The datagram nature of the IP layer enables network saturation measurements, while the controlled TCP allows measurement of a more conventional nature.

By using a single system to carry out measurements on very different networks the bias due to implementation can be eliminated. For instance in a study to compare the response to overloading in two different satellite systems [13].

The datagram based IP enables measurements to be made of the maximum throughput a network can provide to a user. Then using the TCP protocol it is possible to determine how much of that throughput can be utilized by an end user, and what techniques can be used to optimize the throughput [14].

5. Summary and Conclusions

An experimental system and set of protocols has been described which permits communications between heterogeneous host computers on heterogeneous networks. The Internet has evolved over the past fifteen years and has resulted in a set of proven and tested protocols to support the military requirements for robust and reliable computer communications. As those requirements evolve through the development of both new technologies and new military concepts and doctrine, it is anticipated that the Internet system will also continue to evolve, developing new protocols and technologies

to meet those ever-changing requirements.

Acknowledgment

The Internet System has evolved over the years through the dedication and hard work of a large number of researchers. Known as the Internet Research Group, it is their efforts that made the program a success. The authors would also like to thank the members of the Internet Configuration Control Board for their many helpful comments through the preparation of the paper. Finally, we would like to acknowledge Dr. Robert E. Kahn and Dr. Vinton G. Cerf for their vision and guidance in the carrying out of the Internet research activity.

References

1. R. Braden and R. Cole. Some problems in the Interconnection of Computer Networks. Proc ICCC 82, Sept., 1982, pp. 969-974.

2. V.G. Cerf and R.E. Kahn. "A protocol for packet network intercommunication". *IEEE Transactions on Communications COM-22* (May 1974).

3. R.H. Cole. User Experience and Evaluation of International X.25 Services. Proc. Business Telecom Conf., March, 1984.

4. D. Comer. "A computer science research network CSNET: a history and status report". *Communications ACM 26.10* (October 1983), 747-753.

5. DARPA. A History of the Arpanet: The First Decade. Defense Advanced Research Projects Agency, April, 1981. (Defense Tech. Info. Center AD A1 15440).

6. Defense Communications Agency. *MIL STD 1777: Internet Protocol.* 1983.

7. Defense Communications Agency. *MIL STD 1778: Transmission Control Protocol.* 1983.

8. M. Frankel. Advanced technology testbeds for distributed, survivable command , control and communications (C3). Proc. MILCOM82, 1982. paper 10.2.

9. ISO/TC97/SC16/WG6. "Transport protocol specification (N1169)". *Computer Communication Review* (October 1982).

10. I.M. Jacobs, et. al. "General purpose satellite networks". *Proc. IEEE* (Nov. 1978), 1448-1467.

11. R.E. Kahn et. al. "Advances in packet radio technology". *Proc. IEEE* (November 1978), 1468-1496.

12. K. Klemba et. al. Packet Radio Network Executive Summary. DARPA, 1983.

13. P. Lloyd and R. Cole. Transport Protocol Performance over Concatenated Local Area and Satellite Networks. Proc. Conf. Data Networks with Satellites, Sept., 1982.

14. P. Lloyd and R. Cole. A Comparative Study of Protocol Performance On the UNIVERSE and SATNET Satellite Systems. Proc Conf on Satellite and Computer Communications, April, 1983, pp. 353-368.

15. P. Mockapetris. Domain Names - Concepts and Facilities. USC Information Sciences Institute, 1983. RFC-882.

16. L. Palmer, et. al. "SATNET packet data transmission". *COMSAT Technical Review* (Spring 1982), 395-404.

17. J. Postel. User Datagram Protocol. USC Information Sciences Institute, 1980. RFCxxx.

18. J. Postel. File Transfer Protocol. USC Information Sciences Institute, 1980. RFC-765.

19. J. Postel. Internet Control Message Protocol. USC Information Sciences Institute, 1981. RFC-792.

20. J. Postel. Simple Mail Transfer Protocol. USC Information Sciences Institute, 1982. RFC-821.

21. J. Postel and J. Reynolds. Telnet Protocol Specification. USC Information Sciences Institute, 1983. RFC-854.

22. J. Postel and K. Harrenstien. Time Protocol. USC Information Sciences Institute, 1983. RFC-868.

23. H. Zimmerman. "OSI reference model - the ISO model of architecture for open systems interconnection". *IEEE Transactions on Communications* (April 1980), 425-432.

SECTION 3. PROTOCOL REVIEW AND ACCEPTANCE FOR THE DARPA INTERNET

3.1 Request for Comments (RFCs)

Before a proposed protocol is accepted for use on the DARPA Internet, it is discussed, reviewed, and often revised by members of the Internet Advisory Board, its Task Forces, and other interested parties. This dialogue is captured in a set of technical notes known as Requests for Comments or RFCs. RFCs may be submitted online to the Editor-in-Chief, currently Dr. Jonathan B. Postel (POSTEL@USC-ISIF.ARPA). Contributors are requested to foliow the format guidelines outlined in *Instructions for Authors of RFCs*, available online at the NIC in the file RFC:AUTHOR-INSTRUCT.TXT.

RFCs are available online or in hardcopy from the NIC. See Section 4 below for instructions on how to obtain copies of the RFCs and other online NIC files.

3.2 Special Interest Group Discussions

The DARPA Internet also provides a forum for online discussions in several special fields of interest (SIGs). Many of these discussions take place among implementors of the network protocols. One such discussion addresses TCP/IP protocol development. Users of the network can take part in this SIG by joining an online mailing list, called TCP/IP, which is maintained by the NIC. To become a subscriber send a message to TCP-IP-REQUEST@SRI-NIC.ARPA.

For a list of other SIGs, FTP the file NETINFO:INTEREST-GROUPS.TXT from the SRI-NIC host.

3.3 The Internet Advisory Board

The DARPA Internet Research Program is directed by DARPA IPTO with the assistance of an Internet Advisory Board (IAB) and a set of IPTO-appointed Task Forces (technical working committees). The IAB consists of the chairmen of the Task Forces, the DARPA Program Manager, the Chairman of the IAB (the Internet Architect), the Deputy Chairman. and the Secretary of the IAB.

The IAB guides and reviews the work of the Task Forces, and ensures proper cross communication among them. The IAB may from time to time create new, or disband existing, Task Forces.

The Task Forces are expected to generate and develop new ideas, to monitor the

technical work of the Internet program, and to recommend additional research activity. The role of the Task Forces is seminal and advisory, and very important to the advancement of the research goals of the Internet program.

Members of each Task Force are chosen by its chairman, and they are expected to make a moderate commitment of time to the work of the Task Force. Most Task Forces also have mailing lists for persons interested in following the work of a given Task Force. Current Task Forces and chairmen are:

Task Force	Chairman	Organization
Applications	Bob Thomas	BBNCC
Gateway Algorithms and Data Structures	Dave Mills	M/A-COM
Interoperability and Autonomous Systems	Robert Cole	UCL
New End to End Services	Bob Braden	UCLA
Privacy	Steve Kent	BBNCC
Robustness and Survivability	Jim Mathis	SRI
Security	Ray McFarland	DOD
Tactical Internetting	David Hartmann	MITRE
Testing	Ed Cain	DCEC

IAB officers are:

Position	Occupant	Organization
Internet Architect	Dave Clark	MIT
Deputy Internet Architect	Jon Postel	ISI
DARPA Program Manager	Dennis Perry	DARPA
IAB Secretary	Chris Perry	MITRE

BBNCC - BBN Communications Corporation
DARPA - Defense Advanced Research Projects Agency
DCEC - Defense Communications Engineering Center
DOD - Department of Defense
ISI - University of Southern California, Information Sciences Institute
M/A-COM - Linkabit Coporation
MIT - Massachusetts Institute of Technology
MITRE - Mitre Corporation
SRI - SRI International
UCL - University College London
UCLA - University of California at Los Angeles

Phone numbers for IAB members are available from DARPA or through the NIC WHOIS server.

SECTION 4. OBTAINING PROTOCOL INFORMATION

4.1 Military Standards

MIL STD protocols can be ordered from:

Naval Publications and Forms Center, Code 3015
5801 Tabor Drive
Philadelphia, PA 19120

4.2 The DDN Protocol Handbook

Additional copies of this 1985 DDN Protocol Handbook can be ordered from:

DDN Network Information Center
SRI International, Room EJ291
333 Ravenswood Avenue
Menlo Park, CA 94025
Telephone: (800) 235-3155

The price for the three-volume set is $110.00, prepaid, to cover the cost of reproduction and handling. Checks should be made payable to SRI International. Copies of the handbook will also be deposited at DTIC.

4.3 Requests for Comments (RFCs)

RFCs are available online or in hardcopy from the NIC. For network users, the online versions can be obtained via FTP from the SRI-NIC host computer (26.0.0.73 on MILNET and 10.0.0.51 on ARPANET) using username "anonymous" and password "guest" and the pathname of RFC:RFCxxx.TXT, where "xxx" equals the number of the RFC desired. An online index is also available with pathname RFC:RFC-INDEX.TXT. Individuals who wish to be added to the RFC notification list should send a message to NIC@SRI-NIC.ARPA requesting that their name be added to the online distribution list. Hardcopies of RFCs may be obtained from the DDN Network Information Center by sending a check or purchase order made payable to SRI International in the amount of $5.00 for each copy under 100 pages, or $10.00 for 100 pages and above.

4.4 DDN Management Bulletins and Newsletters

The DDN Management Bulletins and informal DDN Newsletters are available for FTP from the SRI-NIC machine using username "anonymous" and password "guest" and pathnames of the type DDN-NEWS:DDN-MGT-BULLETIN-xx.TXT and DDN-NEWS:DDN-NEWS-xx.TXT, where "xx" is the number of the bulletin or newsletter desired. All of the newsletters that are still current are online on the NIC machine.

Special quarterly issues of the DDN Newsletter are published both online and in hardcopy. The hardcopy versions are distributed to appropriate military agencies by the DDN PMO. Additional copies are available from the NIC.

Both DDN Management Bulletins and DDN Newsletters can also be read using the TACNEWS service described above.

4.5 NIC Services

The DDN Network Information Center (NIC) assists users in obtaining information pertaining to DoD protocols. The NIC publishes the DDN Protocol Handbook and maintains a NIC Repository of DoD and related protocol documents. It houses the DDN Management Bulletins, the DDN Newsletters, the Requests for Comments (RFC) technical note series, and also produces the TCP/IP Protocol Implementation and Vendors Guide. The NIC is a good place to start if you need information.

(800) 235-3155

is the toll-free telephone number to call for user assistance. Service is available Monday through Friday, 7 am to 4 pm, Pacific time.

The NIC host computer is a DEC-2065 running the TOPS-20 operating system with the hostname SRI-NIC and host addresses, 26.0.0.73 (MILNET) and 10.0.0.51 (ARPANET). NIC online services are available 24 hours a day, 7 days a week. Operations personnel are in attendance from 4 am - 11 pm weekdays, and 8 am - 12 pm weekends, Pacific time.

Send online mail to:

NIC@SRI-NIC.ARPA

Send U.S. mail to:

DDN Network Information Center
SRI International, Room EJ291
333 Ravenswood Avenue
Menlo Park, CA 94025

4.6 Other Protocol Information Sources

A subscription to the DoD Index of Specifications and Standards (DODISS) can be ordered from:

Naval Publications and Printing Service Office
Fourth Naval District
700 Robbins Avenue
Philadelphia, PA 19111

FIPS Standards can be ordered from:

National Technical Information Service (NTIS)
U.S. Dept. of Commerce
5285 Port Royal Road
Springfield, VA 22161
Telephone: (703) 487-4630

ANSI Standards can be ordered from:

American National Standards Institute (ANSI)
Sales Department
1430 Broadway
New York, NY 10018
Telephone: (212) 354-3300

IEEE documents are available from:

Institution of Electrical and Electronic Engineers
445 Hoes Lane
Piscataway, NJ 08854

CCITT documents can be ordered from:

International Telecommunications Union
General Secretariat, Sales Section
Place des Nations
CH-1211 Geneva 20
SWITZERLAND

SECTION 5. CURRENT OFFICIAL ARPANET PROTOCOLS

This section contains RFC 944, "Official ARPA-Internet Protocols", which identifies the documents specifying the official protocols used in the internet.

Network Working Group
Request for Comments: 961

J. Reynolds
J. Postel
ISI

Obsoletes: RFCs 944, 924, 901, 880, 840

December 1985

OFFICIAL ARPA-INTERNET PROTOCOLS

STATUS OF THIS MEMO

This memo is an official status report on the protocols used in the ARPA-Internet community. Distribution of this memo is unlimited.

INTRODUCTION

This RFC identifies the documents specifying the official protocols used in the Internet. Comments indicate any revisions or changes planned.

To first order, the official protocols are those in the "Internet Protocol Transition Workbook" (IPTW) dated March 1982. There are several protocols in use that are not in the IPTW. A few of the protocols in the IPTW have been revised. Notably, the mail protocols have been revised and issued as a volume titled "Internet Mail Protocols" dated November 1982. Telnet and the most useful Telnet options have been revised and issued as a volume titled "Internet Telnet Protocol and Options" (ITP) dated June 1983. The File Transfer Protocol has been revised most recently as RFC 959 which is not yet included in any collection. Some protocols have not been revised for many years, these are found in the old "ARPANET Protocol Handbook" (APH) dated January 1978. There is also a volume of protocol related information called the "Internet Protocol Implementers Guide" (IPIG) dated August 1982.

This document is organized as a sketchy outline. The entries are protocols (e.g., Transmission Control Protocol). In each entry there are notes on status, specification, comments, other references, dependencies, and contact.

The STATUS is one of: required, recommended, elective, or experimental.

The SPECIFICATION identifies the protocol defining documents.

The COMMENTS describe any differences from the specification or problems with the protocol.

The OTHER REFERENCES identify documents that comment on or expand on the protocol.

The DEPENDENCIES indicate what other protocols are called upon by this protocol.

The CONTACT indicates a person who can answer questions about the protocol.

In particular, the status may be:

required

- all hosts must implement the required protocol,

recommended

- all hosts are encouraged to implement the recommended protocol,

elective

- hosts may implement or not the elective protocol,

experimental

- hosts should not implement the experimental protocol unless they are participating in the experiment and have coordinated their use of this protocol with the contact person, and

none

- this is not a protocol.

For further information about protocols in general, please contact:

Joyce Reynolds
USC - Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292-6695

Phone: (213) 822-1511

ARPA mail: JKREYNOLDS@USC-ISIB.ARPA

OVERVIEW

Catenet Model ----------

STATUS: None

SPECIFICATION: IEN 48 (in IPTW)

COMMENTS:

Gives an overview of the organization and principles of the Internet.

Could be revised and expanded.

OTHER REFERENCES:

Leiner, B., Cole R., Postel, J., and D. Mills, "The DARPA Protocol Suite", IEEE INFOCOM 85, Washington, D.C., March 1985. Also in IEEE Communications Magazine, and as ISI/RS-85-153, March 1985.

Postel, J., "Internetwork Applications Using the DARPA Protocol Suite", IEEE INFOCOM 85, Washington, D.C., March 1985. Also in IEEE Communications Magazine, and as ISI/RS-85-151, April 1985.

Padlipsky, M.A., "The Elements of Networking Style and other Essays and Animadversions on the Art of Intercomputer Networking", Prentice-Hall, New Jersey, 1985.

RFC 871 - A Perspective on the ARPANET Reference Model

DEPENDENCIES:

CONTACT: Postel@USC-ISIB.ARPA

NETWORK LEVEL

Internet Protocol ---------- (IP)

STATUS: Required

SPECIFICATION: RFC 791 (in IPTW)

COMMENTS:

This is the universal protocol of the Internet. This datagram protocol provides the universal addressing of hosts in the Internet.

A few minor problems have been noted in this document.

The most serious is a bit of confusion in the route options. The route options have a pointer that indicates which octet of the route is the next to be used. The confusion is between the phrases "the pointer is relative to this option" and "the smallest legal value for the pointer is 4". If you are confused, forget about the relative part, the pointer begins at 4.

Another important point is the alternate reassembly procedure suggested in RFC 815.

Some changes are in the works for the security option.

Note that ICMP is defined to be an integral part of IP. You have not completed an implementation of IP if it does not include ICMP.

OTHER REFERENCES:

RFC 815 (in IPIG) - IP Datagram Reassembly Algorithms

RFC 814 (in IPIG) - Names, Addresses, Ports, and Routes

RFC 816 (in IPIG) - Fault Isolation and Recovery

RFC 817 (in IPIG) - Modularity and Efficiency in Protocol Implementation

MIL-STD-1777 - Military Standard Internet Protocol

RFC 963 - Some Problems with the Specification of the Military Standard Internet Protocol

DEPENDENCIES:

CONTACT: Postel@USC-ISIB.ARPA

Internet Control Message Protocol ---------- (ICMP)

STATUS: Required

SPECIFICATION: RFC 792 (in IPTW)

COMMENTS:

The control messages and error reports that go with the Internet Protocol.

A few minor errors in the document have been noted. Suggestions have been made for additional types of redirect message and additional destination unreachable messages.

A proposal for two additional ICMP message types is made in RFC 950 "Internet Subnets", Address Mask Request (A1=17), and Address Mask Reply (A2=18). The details of these ICMP types are subject to change. Use of these ICMP types is experimental.

Note that ICMP is defined to be an integral part of IP. You have not completed an implementation of IP if it does not include ICMP.

OTHER REFERENCES: RFC 950

DEPENDENCIES: Internet Protocol

CONTACT: Postel@USC-ISIB.ARPA

HOST LEVEL

User Datagram Protocol ---------- (UDP)

STATUS: Recommended

SPECIFICATION: RFC 768 (in IPTW)

COMMENTS:

Provides a datagram service to applications. Adds port addressing to the IP services.

The only change noted for the UDP specification is a minor clarification that if in computing the checksum a padding octet is used for the computation it is not transmitted or counted in the length.

OTHER REFERENCES:

DEPENDENCIES: Internet Protocol

CONTACT: Postel@USC-ISIB.ARPA

Transmission Control Protocol ---------- (TCP)

STATUS: Recommended

SPECIFICATION: RFC 793 (in IPTW)

COMMENTS:

Provides reliable end-to-end data stream service.

Many comments and corrections have been received for the TCP specification document. These are primarily document bugs rather than protocol bugs.

Event Processing Section: There are many minor corrections and clarifications needed in this section.

Push: There are still some phrases in the document that give a "record mark" flavor to the push. These should be further clarified. The push is *not* a record mark.

Urgent: Page 17 is wrong. The urgent pointer points to the last octet of urgent data (not to the first octet of non-urgent data).

Listening Servers: Several comments have been received on difficulties with contacting listening servers. There should be some discussion of implementation issues for servers, and some notes on alternative models of system and process organization for servers.

Maximum Segment Size: The maximum segment size option should be generalized and clarified. It can be used to either increase or decrease the maximum segment size from the default. The TCP Maximum Segment Size is the IP Maximum Datagram Size minus forty. The default IP Maximum Datagram Size is 576. The default TCP Maximum Segment Size is 536. For further discussion, see RFC 879.

Idle Connections: There have been questions about automatically closing idle connections. Idle connections are ok, and should not be closed. There are several cases where idle connections arise, for example, in Telnet when a user is thinking for a long time following a message from the server computer before his next input. There is no TCP "probe" mechanism, and none is needed.

Queued Receive Data on Closing: There are several points where it is not clear from the description what to do about data received by the TCP but not yet passed to the user, particularly when the connection is being closed. In general, the data is to be kept to give to the user if he does a RECV call.

Out of Order Segments: The description says that segments that arrive out of order, that is, are not exactly the next segment to be processed, may be kept on hand. It should also point out that there is a very large performance penalty for not doing so.

User Time Out: This is the time out started on an open or send call. If this user time out occurs the user should be notified, but the connection should not be closed or the TCB deleted. The user should explicitly ABORT the connection if he wants to give up.

OTHER REFERENCES:

RFC 813 (in IPIG) - Window and Acknowledgement Strategy in TCP

RFC 814 (in IPIG) - Names, Addresses, Ports, and Routes

RFC 816 (in IPIG) - Fault Isolation and Recovery

RFC 817 (in IPIG) - Modularity and Efficiency in Protocol Implementation

RFC 879 - TCP Maximum Segment Size

RFC 889 - Internet Delay Experiments

RFC 896 - TCP/IP Congestion Control

MIL-STD-1778 - Military Standard Transmission Control Protocol

RFC 964 - Some Problems with the Specification of the Military Standard Transmission Control Protocol

DEPENDENCIES: Internet Protocol

CONTACT: Postel@USC-ISIB.ARPA

Host Monitoring Protocol ---------- (HMP)

STATUS: Elective

SPECIFICATION: RFC 869

COMMENTS:

This is a good tool for debugging protocol implementations in remotely located computers.

This protocol is used to monitor Internet gateways and the TACs.

OTHER REFERENCES:

DEPENDENCIES: Internet Protocol

CONTACT: Hinden@BBN-UNIX.ARPA

Cross Net Debugger ---------- (XNET)

STATUS: Elective

SPECIFICATION: IEN 158

COMMENTS:

A debugging protocol, allows debugger like access to remote systems.

This specification should be updated and reissued as an RFC.

OTHER REFERENCES: RFC 643

DEPENDENCIES: Internet Protocol

CONTACT: Postel@USC-ISIB.ARPA

"Stub" Exterior Gateway Protocol ---------- (EGP)

STATUS: Recommended for Gateways

SPECIFICATION: RFC 888, RFC 904

COMMENTS:

The protocol used between gateways of different administrations to exchange routing information.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES: RFC 827, RFC 890

DEPENDENCIES: Internet Protocol

CONTACT: Mills@USC-ISID.ARPA

Gateway Gateway Protocol ---------- (GGP)

STATUS: Experimental

SPECIFICATION: RFC 823

COMMENTS:

The gateway protocol now used in the core gateways.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Internet Protocol

CONTACT: Brescia@BBN-UNIX.ARPA

Multiplexing Protocol ---------- (MUX)

STATUS: Experimental

SPECIFICATION: IEN 90

COMMENTS:

Defines a capability to combine several segments from different higher level protocols in one IP datagram.

No current experiment in progress. There is some question as to the extent to which the sharing this protocol envisions can actually take place. Also, there are some issues about the information captured in the multiplexing header being (a) insufficient, or (b) over specific.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Internet Protocol

CONTACT: Postel@USC-ISIB.ARPA

Stream Protocol ---------- (ST)

STATUS: Experimental

SPECIFICATION: IEN 119

COMMENTS:

A gateway resource allocation protocol designed for use in multihost real time applications.

The implementation of this protocol has evolved and may no longer be consistent with this specification. The document should be updated and issued as an RFC.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Internet Protocol

CONTACT: jwf@LL-EN.ARPA

Network Voice Protocol ---------- (NVP-II)

STATUS: Experimental

SPECIFICATION: ISI Internal Memo

COMMENTS:

Defines the procedures for real time voice conferencing.

The specification is an ISI Internal Memo which should be updated and issued as an RFC.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES: RFC 741

DEPENDENCIES: Internet Protocol, Stream Protocol

CONTACT: Casner@USC-ISIB.ARPA

Reliable Data Protocol ---------- (RDP)

STATUS: Experimental

SPECIFICATION: RFC 908

COMMENTS:

This protocol is designed to efficiently support the bulk transfer of data for such host monitoring and control applications as loading/dumping and remote debugging. The protocol is intended to be simple to implement but still be efficient in environments where there may be long transmission delays and loss or non-sequential delivery of message segments.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Internet Protocol

CONTACT: CWelles@BBN-UNIX.ARPA

Internet Reliable Transaction Protocol ---------- (IRTP)

STATUS: Experimental

SPECIFICATION: RFC 938

COMMENTS:

This protocol is a transport level host to host protocol designed for an internet environment. While the issues discussed may not be directly relevant to the research problems of the DARPA community, they may be interesting to a number of researchers and implementors.

OTHER REFERENCES:

DEPENDENCIES: Internet Protocol

CONTACT: Trudy@ACC.ARPA

APPLICATION LEVEL

Telnet Protocol ---------- (TELNET)

STATUS: Recommended

SPECIFICATION: RFC 854 (in "Internet Telnet Protocol and Options")

COMMENTS:

The protocol for remote terminal access.

This has been revised since the IPTW. RFC 764 in IPTW is now obsolete.

OTHER REFERENCES:

MIL-STD-1782 - Telnet Protocol

DEPENDENCIES: Transmission Control Protocol

CONTACT: Postel@USC-ISIB.ARPA

Telnet Options ---------- (TELNET-OPTIONS)

STATUS: Elective

SPECIFICATION: General description of options: RFC 855
(in "Internet Telnet Protocol and Options")

Number	Name	RFC	NIC	ITP	APH	USE
0	Binary Transmission	856	-----	yes	obs	yes
1	Echo	857	-----	yes	obs	yes
2	Reconnection	...	15391	no	yes	no
3	Suppress Go Ahead	858	-----	yes	obs	yes
4	Approx Message Size Negotiation	...	15393	no	yes	no
5	Status	859	-----	yes	obs	yes
6	Timing Mark	860	-----	yes	obs	yes
7	Remote Controlled Trans and Echo	726	39237	no	yes	no
8	Output Line Width	...	20196	no	yes	no
9	Output Page Size	...	20197	no	yes	no
10	Output Carriage-Return Disposition	652	31155	no	yes	no
11	Output Horizontal Tabstops	653	31156	no	yes	no
12	Output Horizontal Tab Disposition	654	31157	no	yes	no
13	Output Formfeed Disposition	655	31158	no	yes	no
14	Output Vertical Tabstops	656	31159	no	yes	no
15	Output Vertical Tab Disposition	657	31160	no	yes	no
16	Output Linefeed Disposition	658	31161	no	yes	no
17	Extended ASCII	698	32964	no	yes	no
18	Logout	727	40025	no	yes	no
19	Byte Macro	735	42083	no	yes	no
20	Data Entry Terminal	732	41762	no	yes	no
21	SUPDUP 734	736	42213	no	yes	no
22	SUPDUP Output	749	45449	no	no	no
23	Send Location	779	-----	no	no	no
24	Terminal Type	930	-----	no	no	no
25	End of Record	885	-----	no	no	no
26	TACACS User Identification	927	-----	no	no	no
27	Output Marking	933	-----	no	no	no
28	Terminal Location Number	946	-----	no	no	no
255	Extended-Options-List	861	-----	yes	obs	yes

(obs = obsolete)

The ITP column indicates if the specification is included in the Internet Telnet Protocol and Options. The APH column indicates if the specification is included in the ARPANET Protocol Handbook. The USE column of the table above indicates which options are in general use.

COMMENTS:

The Binary Transmission, Echo, Suppress Go Ahead, Status, Timing Mark, and Extended Options List options have been recently updated and reissued. These are the most frequently implemented options.

The remaining options should be reviewed and the useful ones should be revised and reissued. The others should be eliminated.

The following are recommended: Binary Transmission, Echo, Suppress Go Ahead, Status, Timing Mark, and Extended Options List.

OTHER REFERENCES:

DEPENDENCIES: Telnet

CONTACT: Postel@USC-ISIB.ARPA

File Transfer Protocol ---------- (FTP)

STATUS: Recommended

SPECIFICATION: RFC 959

COMMENTS:

The protocol for moving files between Internet hosts. Provides for access control and negotiation of file parameters.

The following new optional commands are included in this edition of the specification: Change to Parent Directory (CDUP), Structure Mount (SMNT), Store Unique (STOU), Remove Directory (RMD), Make Directory (MKD), Print Directory (PWD), and System (SYST). Note that this specification is compatible with the previous edition (RFC 765).

OTHER REFERENCES:

RFC 678 - Document File Format Standards

MIL-STD-1780 - File Transfer Protocol

DEPENDENCIES: Transmission Control Protocol

CONTACT: Postel@USC-ISIB.ARPA

Trivial File Transfer Protocol ---------- (TFTP)

STATUS: Elective

SPECIFICATION: RFC 783 (in IPTW)

COMMENTS:

A very simple file moving protocol, no access control is provided.

This is in use in several local networks.

Ambiguities in the interpretation of several of the transfer modes should be clarified, and additional transfer modes could be defined. Additional error codes could be defined to more clearly identify problems.

OTHER REFERENCES:

DEPENDENCIES: User Datagram Protocol

CONTACT: Postel@USC-ISIB.ARPA

Simple File Transfer Protocol ---------- (SFTP)

STATUS: Experimental

SPECIFICATION: RFC 913

COMMENTS:

SFTP is a simple file transfer protocol. It fills the need of people wanting a protocol that is more useful than TFTP but easier to implement (and less powerful) than FTP. SFTP supports user access control, file transfers, directory listing, directory changing, file renaming and deleting.

SFTP can be implemented with any reliable 8-bit byte stream oriented protocol, this document describes its TCP specification. SFTP uses only one TCP connection; whereas TFTP implements a connection over UDP, and FTP uses two TCP connections (one using the TELNET protocol).

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol

CONTACT: MKL@SRI-NIC.ARPA

Simple Mail Transfer Protocol ---------- (SMTP)

STATUS: Recommended

SPECIFICATION: RFC 821 (in "Internet Mail Protocols")

COMMENTS:

The procedure for transmitting computer mail between hosts.

This has been revised since the IPTW, it is in the "Internet Mail Protocols" volume of November 1982. RFC 788 (in IPTW) is obsolete.

There have been many misunderstandings and errors in the early implementations. Some documentation of these problems can be found in the file [ISIB]<SMTP>MAIL.ERRORS.

Some minor differences between RFC 821 and RFC 822 should be resolved.

OTHER REFERENCES:

RFC 822 - Mail Header Format Standards

This has been revised since the IPTW, it is in the "Internet Mail Protocols" volume of November 1982. RFC 733 (in IPTW) is obsolete. Further revision of RFC 822 is needed to correct some minor errors in the details of the specification.

MIL-STD-1781 - Simple Mail Transfer Protocol (SMTP)

DEPENDENCIES: Transmission Control Protocol

CONTACT: Postel@USC-ISIB.ARPA

Resource Location Protocol ---------- (RLP)

STATUS: Elective

SPECIFICATION: RFC 887

COMMENTS:

A resource location protocol for use in the ARPA-Internet. This protocol utilizes the User Datagram Protocol (UDP) which in turn calls on the Internet Protocol to deliver its datagrams.

OTHER REFERENCES:

DEPENDENCIES: User Datagram Protocol

CONTACT: Accetta@CMU-CS-A.ARPA

Loader Debugger Protocol ---------- (LDP)

STATUS: Experimental

SPECIFICATION: RFC 909

COMMENTS:

Specifies a protocol for loading, dumping and debugging target machines from hosts in a network environment. It is also designed to accommodate a variety of target CPU types. It provides a powerful set of debugging services, while at the same time, it is structured so that a simple subset may be implemented in applications like boot loading where efficiency and space are at a premium.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Reliable Data Protocol

CONTACT: Hinden@BBN-UNIX.ARPA

Remote Job Entry ---------- (RJE)

STATUS: Elective

SPECIFICATION: RFC 407 (in APH)

COMMENTS:

The general protocol for submitting batch jobs and retrieving the results.

Some changes needed for use with TCP.

No known active implementations.

OTHER REFERENCES:

DEPENDENCIES: File Transfer Protocol
Transmission Control Protocol

CONTACT: Postel@USC-ISIB.ARPA

Remote Job Service ---------- (NETRJS)

STATUS: Elective

SPECIFICATION: RFC 740 (in APH)

COMMENTS:

A special protocol for submitting batch jobs and retrieving the results used with the UCLA IBM OS system.

Please discuss any plans for implementation or use of this protocol with the contact.

Revision in progress.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol

CONTACT: Braden@UCLA-CCN.ARPA

Remote Telnet Service ---------- (RTELNET)

STATUS: Elective

SPECIFICATION: RFC 818

COMMENTS:

Provides special access to user Telnet on a remote system.

OTHER REFERENCES:

DEPENDENCIES: Telnet, Transmission Control Protocol

CONTACT: Postel@USC-ISIB.ARPA

Graphics Protocol ---------- (GRAPHICS)

STATUS: Elective

SPECIFICATION: NIC 24308 (in APH)

COMMENTS:

The protocol for vector graphics.

Very minor changes needed for use with TCP.

No known active implementations.

OTHER REFERENCES:

DEPENDENCIES: Telnet, Transmission Control Protocol

CONTACT: Postel@USC-ISIB.ARPA

Official ARPA-Internet Protocols RFC 961

Echo Protocol ---------- (ECHO)

STATUS: Recommended

SPECIFICATION: RFC 862

COMMENTS:

Debugging protocol, sends back whatever you send it.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol
or User Datagram Protocol

CONTACT: Postel@USC-ISIB.ARPA

Discard Protocol ---------- (DISCARD)

STATUS: Elective

SPECIFICATION: RFC 863

COMMENTS:

Debugging protocol, throws away whatever you send it.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol
or User Datagram Protocol

CONTACT: Postel@USC-ISIB.ARPA

Character Generator Protocol ---------- (CHARGEN)

STATUS: Elective

SPECIFICATION: RFC 864

COMMENTS:

Debugging protocol, sends you ASCII data.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol
or User Datagram Protocol

CONTACT: Postel@USC-ISIB.ARPA

Quote of the Day Protocol ---------- (QUOTE)

STATUS: Elective

SPECIFICATION: RFC 865

COMMENTS:

Debugging protocol, sends you a short ASCII message.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol
or User Datagram Protocol

CONTACT: Postel@USC-ISIB.ARPA

Active Users Protocol ---------- (USERS)

STATUS: Elective

SPECIFICATION: RFC 866

COMMENTS:

Lists the currently active users.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol
or User Datagram Protocol

CONTACT: Postel@USC-ISIB.ARPA

Finger Protocol ---------- (FINGER)

STATUS: Elective

SPECIFICATION: RFC 742 (in APH)

COMMENTS:

Provides information on the current or most recent activity of a user.

Some extensions have been suggested.

Some changes are are needed for TCP.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol

CONTACT: Postel@USC-ISIB.ARPA

WhoIs Protocol ---------- (NICNAME)

STATUS: Elective

SPECIFICATION: RFC 954

COMMENTS:

Accesses the ARPANET Directory database. Provides a way to find out about people, their addresses, phone numbers, organizations, and mailboxes.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol

CONTACT: Feinler@SRI-NIC.ARPA

Domain Name Protocol ---------- (DOMAIN)

STATUS: Recommended

SPECIFICATION: RFC 881, 882, 883

COMMENTS:

OTHER REFERENCES:

RFC 920 - Domain Requirements

RFC 921 - Domain Name Implementation Schedule - Revised

DEPENDENCIES: Transmission Control Protocol
or User Datagram Protocol

CONTACT: Mockapetris@USC-ISIB.ARPA

HOSTNAME Protocol ---------- (HOSTNAME)

STATUS: Elective

SPECIFICATION: RFC 953

COMMENTS:

Accesses the Registered Internet Hosts database (HOSTS.TXT). Provides a way to find out about a host in the Internet, its Internet Address, and the protocols it implements.

OTHER REFERENCES:

RFC 952 - Host Table Specification

DEPENDENCIES: Transmission Control Protocol

CONTACT: Feinler@SRI-NIC.ARPA

Host Name Server Protocol ---------- (NAMESERVER)

STATUS: Experimental

SPECIFICATION: IEN 116 (in IPTW)

COMMENTS:

Provides machine oriented procedure for translating a host name to an Internet Address.

This specification has significant problems: 1) The name syntax is out of date. 2) The protocol details are ambiguous, in particular, the length octet either does or doesn't include itself and the op code. 3) The extensions are not supported by any known implementation.

This protocol is now abandoned in favor of the DOMAIN protocol. Further implementations of this protocol are not advised.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: User Datagram Protocol

CONTACT: Postel@USC-ISIB.ARPA

CSNET Mailbox Name Server Protocol ---------- (CSNET-NS)

STATUS: Experimental

SPECIFICATION: CS-DN-2

COMMENTS:

Provides access to the CSNET data base of users to give information about users names, affiliations, and mailboxes.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol

CONTACT: Solomon@UWISC.ARPA

Daytime Protocol ---------- (DAYTIME)

STATUS: Elective

SPECIFICATION: RFC 867

COMMENTS:

Provides the day and time in ASCII character string.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol
or User Datagram Protocol

CONTACT: Postel@USC-ISIB.ARPA

Network Time Protocol ---------- (NTP)

STATUS: Experimental

SPECIFICATION: RFC 958

COMMENTS:

A proposed protocol for synchronizing a set of network clocks using a set of distributed clients and servers.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES: RFC 778, RFC 891, RFC 956, and RFC 957.

DEPENDENCIES: User Datagram Protocol

CONTACT: Mills@USC-ISID.ARPA

Time Server Protocol ---------- (TIME)

STATUS: Elective

SPECIFICATION: RFC 868

COMMENTS:

Provides the time as the number of seconds from a specified reference time.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol
or User Datagram Protocol

CONTACT: Postel@USC-ISIB.ARPA

DCNET Time Server Protocol ---------- (CLOCK)

STATUS: Experimental

SPECIFICATION: RFC 778

COMMENTS:

Provides a mechanism for keeping synchronized clocks.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Internet Control Message Protocol

CONTACT: Mills@USC-ISID.ARPA

SUPDUP Protocol ---------- (SUPDUP)

STATUS: Elective

SPECIFICATION: RFC 734 (in APH)

COMMENTS:

A special Telnet like protocol for display terminals.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol

CONTACT: Crispin@SU-SCORE.ARPA

Internet Message Protocol ---------- (MPM)

STATUS: Experimental

SPECIFICATION: RFC 759

COMMENTS:

This is an experimental multimedia mail transfer protocol. The implementation is called a Message Processing Module or MPM.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

RFC 767 - Structured Document Formats

DEPENDENCIES: Transmission Control Protocol

CONTACT: Postel@USC-ISIB.ARPA

Post Office Protocol - Version 2 ---------- (POP2)

STATUS: Experimental

SPECIFICATION: RFC 937

COMMENTS:

The intent of the Post Office Protocol - Version 2 (POP2) is to allow a user's workstation to access mail from a mailbox server. It is expected that mail will be posted from the workstation to the mailbox server via the Simple Mail Transfer Protocol (SMTP).

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES: Obsoletes RFC 918

DEPENDENCIES: Transmission Control Protocol

CONTACT: JKReynolds@USC-ISIB.ARPA

Network Standard Text Editor ---------- (NETED)

STATUS: Elective

SPECIFICATION: RFC 569

COMMENTS:

Describes a simple line editor which could be provided by every Internet host.

OTHER REFERENCES:

DEPENDENCIES:

CONTACT: Postel@USC-ISIB.ARPA

APPENDICES

Assigned Numbers ----------

STATUS: None

SPECIFICATION: RFC 960

COMMENTS:

Describes the fields of various protocols that are assigned specific values for actual use, and lists the currently assigned values.

Issued November 1985, replaces RFC 943, RFC 790 in IPTW, and RFC 923.

OTHER REFERENCES:

CONTACT: JKReynolds@USC-ISIB.ARPA

Pre-emption ----------

STATUS: Elective

SPECIFICATION: RFC 794 (in IPTW)

COMMENTS:

Describes how to do pre-emption of TCP connections.

OTHER REFERENCES:

CONTACT: Postel@USC-ISIB.ARPA

Authentication Service ---------- (AUTH)

STATUS: Experimental

SPECIFICATION: RFC 931

COMMENTS:

This server provides a means to determine the identity of a user of a particular TCP connection. Given a TCP port number pair, it returns a character string which identifies the owner of that connection on the server's system.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES: Supercedes RFC 912

DEPENDENCIES: Transmission Control Protocol

CONTACT: StJohns@MIT-Multics.ARPA

Bootstrap Protocol ---------- (BOOTP)

STATUS: Experimental

SPECIFICATION: RFC 951

COMMENTS:

This proposed protocol provides an IP/UDP bootstrap protocol which allows a diskless client machine to discover its own IP address, the address of a server host, and the name of a file to be loaded into memory and executed.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Internet Protocol, User Datagram Protocol

CONTACT: Croft@SUMEX-AIM.ARPA

Service Mappings ----------

STATUS: None

SPECIFICATION: RFC 795 (in IPTW)

COMMENTS:

Describes the mapping of the IP type of service field onto the parameters of some specific networks.

Out of date, needs revision.

OTHER REFERENCES:

CONTACT: Postel@USC-ISIB.ARPA

Address Mappings ----------

STATUS: None

SPECIFICATION: RFC 796 (in IPTW)

COMMENTS:

Describes the mapping between Internet Addresses and the addresses of some specific networks.

Out of date, needs revision.

OTHER REFERENCES:

CONTACT: Postel@USC-ISIB.ARPA

Document Formats ----------

STATUS: None

SPECIFICATION: RFC 678

COMMENTS:

Describes standard format rules for several types of documents.

OTHER REFERENCES:

CONTACT: Postel@USC-ISIB.ARPA

Bitmap Formats ----------

STATUS: None

SPECIFICATION: RFC 797

COMMENTS:

Describes a standard format for bitmap data.

OTHER REFERENCES:

CONTACT: Postel@USC-ISIB.ARPA

Facsimile Formats ----------

STATUS: None

SPECIFICATION: RFC 804

COMMENTS:

Describes a standard format for facsimile data.

OTHER REFERENCES:

CONTACT: Postel@USC-ISIB.ARPA

Host-Front End Protocol ---------- (HFEP)

STATUS: Experimental

SPECIFICATION: RFC 929

COMMENTS:

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES: RFC 928

DEPENDENCIES:

CONTACT: Padlipsky@USC-ISI.ARPA

Internet Protocol on X.25 Networks ---------- (IP-X25)

STATUS: Recommended

SPECIFICATION: RFC 877

COMMENTS:

Describes a standard for the transmission of IP Datagrams over Public Data Networks.

OTHER REFERENCES:

CONTACT: jtk@PURDUE.ARPA

Internet Protocol on DC Networks ---------- (IP-DC)

STATUS: Elective

SPECIFICATION: RFC 891

COMMENTS:

OTHER REFERENCES:

RFC 778 - DCNET Internet Clock Service

CONTACT: Mills@USC-ISID.ARPA

Internet Protocol on Ethernet Networks ---------- (IP-E)

STATUS: Recommended

SPECIFICATION: RFC 894

COMMENTS:

OTHER REFERENCES: RFC 893

CONTACT: Postel@USC-ISIB.ARPA

Internet Protocol on Experimental Ethernet Networks -------- (IP-EE)

STATUS: Recommended

SPECIFICATION: RFC 895

COMMENTS:

OTHER REFERENCES:

CONTACT: Postel@USC-ISIB.ARPA

Internet Protocol on IEEE 802.3 ---------- (IP-IEEE)

STATUS: Recommended

SPECIFICATION: RFC 948

COMMENTS: A proposed protocol of two methods of encapsulating Internet Protocol (IP) datagrams on an IEEE 802.3 network.

OTHER REFERENCES:

CONTACT: Ira@UPENN.CSNET

Internet Subnet Protocol ---------- (IP-SUB)

STATUS: Recommended

SPECIFICATION: RFC 950

COMMENTS:

Specifies procedures for the use of subnets, including the ultility of "subnets" of Internet networks, which are logically visible sub-sections of a single Internet. Recommended in the sense of "if you do subnetting at all do it this way".

OTHER REFERENCES: RFC 940, RFC 917, RFC 925, RFC 932, RFC 936, RFC 922

DEPENDENCIES:

CONTACT: Mogul@SU-SCORE.ARPA

Broadcasting Internet Datagrams ---------- (IP-BROAD)

STATUS: Experimental

SPECIFICATION: RFC 919

COMMENTS:

A proposed protocol of simple rules for broadcasting Internet datagrams on local networks that support broadcast, for addressing broadcasts, and for how gateways should handle them.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES: RFC 922

DEPENDENCIES:

CONTACT: Mogul@SU-SCORE.ARPA

Address Resolution Protocol ---------- (ARP)

STATUS: Recommended

SPECIFICATION: RFC 826

COMMENTS:

This is a procedure for finding the network hardware address corresponding to an Internet Address.

OTHER REFERENCES:

CONTACT: Postel@USC-ISIB.ARPA

A Reverse Address Resolution Protocol ---------- (RARP)

STATUS: Elective

SPECIFICATION: RFC 903

COMMENTS:

This is a procedure for workstations to dynamically find their protocol address (e.g., their Internet Address), when they only only know their hardware address (e.g., their attached physical network address).

OTHER REFERENCES:

CONTACT: Mogul@SU-SCORE.ARPA

Multi-LAN Address Resolution Protocol ---------- (MARP)

STATUS: Experimental

SPECIFICATION: RFC 925

COMMENTS:

Discussion of the various problems and potential solutions of "transparent subnets" in a multi-LAN environment.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES: RFC 917, RFC 826

DEPENDENCIES:

CONTACT: Postel@USC-ISIB.ARPA

Broadcasting Internet Datagrams with Subnets --------- (IP-SUB-BROAD)

STATUS: Experimental

SPECIFICATION: RFC 922

COMMENTS:

A proposed protocol of simple rules for broadcasting Internet datagrams on local networks that support broadcast, for addressing broadcasts, and for how gateways should handle them.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES:

CONTACT: Mogul@SU-SCORE.ARPA

Host Access Protocol ---------- (HAP)

STATUS: Recommended

SPECIFICATION: RFC 907

COMMENTS:

This protocol specifies the network-access level communication between an arbitrary computer, called a host, and a packet-switched satellite network, e.g., SATNET or WBNET.

Note: Implementations of HAP should be performed in coordination with satellite network development and operations personnel.

OTHER REFERENCES:

DEPENDENCIES:

CONTACT: Schoen@BBN-UNIX.ARPA

Reliable Asynchronous Transfer Protocol ---------- (RATP)

STATUS: Experimental

SPECIFICATION: RFC 916

COMMENTS:

This paper specifies a protocol which allows two programs to reliably communicate over a communication link. It ensures that the data entering one end of the link if received arrives at the other end intact and unaltered. This proposed protocol is designed to operate over a full duplex point-to-point connection. It contains some features which tailor it to the RS-232 links now in current use.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES: Transmission Control Protocol

CONTACT: Finn@USC-ISIB.ARPA

Thinwire Protocol ---------- (THINWIRE)

STATUS: Experimental

SPECIFICATION: RFC 914

COMMENTS:

This paper discusses a Thinwire Protocol for connecting personal computers to the ARPA-Internet. It primarily focuses on the particular problems in the ARPA-Internet of low speed network interconnection with personal computers, and possible methods of solution.

Please discuss any plans for implementation or use of this protocol with the contact.

OTHER REFERENCES:

DEPENDENCIES:

CONTACT: Farber@ROCHESTER.ARPA

SECTION 6. NETWORK LEVEL PROTOCOLS

This section contains the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP).

RFC: 791

INTERNET PROTOCOL

DARPA INTERNET PROGRAM

PROTOCOL SPECIFICATION

September 1981

prepared for

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

by

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

September 1981

Internet Protocol

TABLE OF CONTENTS

PREFACE

This document specifies the DoD Standard Internet Protocol. This document is based on six earlier editions of the ARPA Internet Protocol Specification, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition revises aspects of addressing, error handling, option codes, and the security, precedence, compartments, and handling restriction features of the internet protocol.

Jon Postel

Editor

September 1981

RFC: 791
Replaces: RFC 760
IENs 128, 123, 111,
80, 54, 44, 41, 28, 26

INTERNET PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

1. INTRODUCTION

1.1. Motivation

The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. Such a system has been called a "catenet" [1]. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

1.2. Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service.

1.3. Interfaces

This protocol is called on by host-to-host protocols in an internet environment. This protocol calls on local network protocols to carry the internet datagram to the next gateway or destination host.

For example, a TCP module would call on the internet module to take a TCP segment (including the TCP header and user data) as the data portion of an internet datagram. The TCP module would provide the addresses and other parameters in the internet header to the internet module as arguments of the call. The internet module would then create an internet datagram and call on the local network interface to transmit the internet datagram.

In the ARPANET case, for example, the internet module would call on a

local net module which would add the 1822 leader [2] to the internet datagram creating an ARPANET message to transmit to the IMP. The ARPANET address would be derived from the internet address by the local network interface and would be the address of some host in the ARPANET, that host might be a gateway to other networks.

1.4. Operation

The internet protocol implements two basic functions: addressing and fragmentation.

The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing.

The internet modules use fields in the internet header to fragment and reassemble internet datagrams when necessary for transmission through "small packet" networks.

The model of operation is that an internet module resides in each host engaged in internet communication and in each gateway that interconnects networks. These modules share common rules for interpreting address fields and for fragmenting and assembling internet datagrams. In addition, these modules (especially in gateways) have procedures for making routing decisions and other functions.

The internet protocol treats each internet datagram as an independent entity unrelated to any other internet datagram. There are no connections or logical circuits (virtual or otherwise).

The internet protocol uses four key mechanisms in providing its service: Type of Service, Time to Live, Options, and Header Checksum.

The Type of Service is used to indicate the quality of the service desired. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the internet. This type of service indication is to be used by gateways to select the actual transmission parameters for a particular network, the network to be used for the next hop, or the next gateway when routing an internet datagram.

The Time to Live is an indication of an upper bound on the lifetime of an internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live can be thought of as a self destruct time limit.

The Options provide for control functions needed or useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, security, and special routing.

The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error.

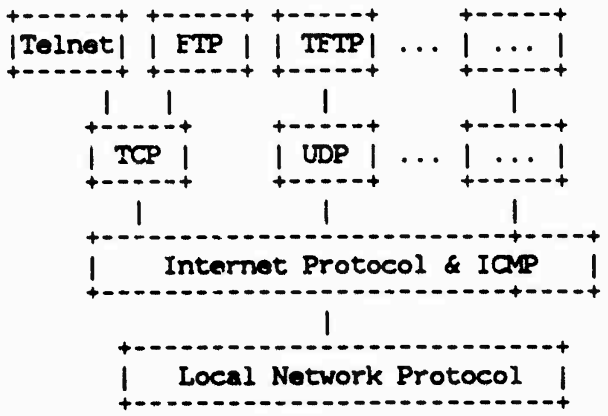
The internet protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control.

Errors detected may be reported via the Internet Control Message Protocol (ICMP) [3] which is implemented in the internet protocol module.

2. OVERVIEW

2.1. Relation to Other Protocols

The following diagram illustrates the place of the internet protocol in the protocol hierarchy:



Protocol Relationships

Figure 1.

Internet protocol interfaces on one side to the higher level host-to-host protocols and on the other side to the local network protocol. In this context a "local network" may be a small network in a building or a large network such as the ARPANET.

2.2. Model of Operation

The model of operation for transmitting a datagram from one application program to another is illustrated by the following scenario:

We suppose that this transmission will involve one intermediate gateway.

The sending application program prepares its data and calls on its local internet module to send that data as a datagram and passes the destination address and other parameters as arguments of the call.

The internet module prepares a datagram header and attaches the data to it. The internet module determines a local network address for this internet address. In this case it is the address of a gateway.

It sends this datagram and the local network address to the local network interface.

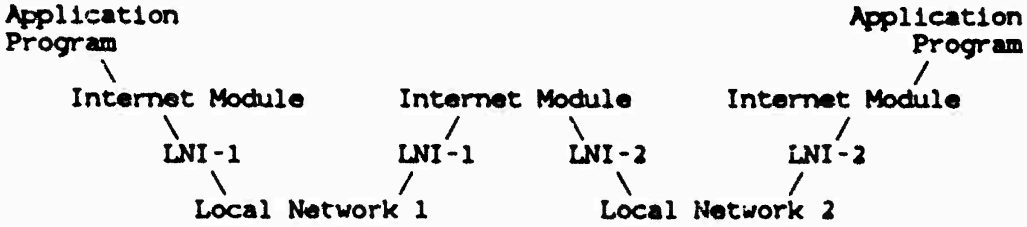
The local network interface creates a local network header, and attaches the datagram to it, then sends the result via the local network.

The datagram arrives at a gateway host wrapped in the local network header, the local network interface strips off this header, and turns the datagram over to the internet module. The internet module determines from the internet address that the datagram is to be forwarded to another host in a second network. The internet module determines a local net address for the destination host. It calls on the local network interface for that network to send the datagram.

This local network interface creates a local network header and attaches the datagram sending the result to the destination host.

At this destination host the datagram is stripped of the local net header by the local network interface and handed to the internet module.

The internet module determines that the datagram is for an application program in this host. It passes the data to the application program in response to a system call, passing the source address and other parameters as results of the call.



Transmission Path

Figure 2

2.3. Function Description

The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached. The internet modules reside in hosts and gateways in the internet system. The datagrams are routed from one internet module to another through individual networks based on the interpretation of an internet address. Thus, one important mechanism of the internet protocol is the internet address.

In the routing of messages from one internet module to another, datagrams may need to traverse a network whose maximum packet size is smaller than the size of the datagram. To overcome this difficulty, a fragmentation mechanism is provided in the internet protocol.

Addressing

A distinction is made between names, addresses, and routes [4]. A name indicates what we seek. An address indicates where it is. A route indicates how to get there. The internet protocol deals primarily with addresses. It is the task of higher level (i.e., host-to-host or application) protocols to make the mapping from names to addresses. The internet module maps internet addresses to local net addresses. It is the task of lower level (i.e., local net or gateways) procedures to make the mapping from local net addresses to routes.

Addresses are fixed length of four octets (32 bits). An address begins with a network number, followed by local address (called the "rest" field). There are three formats or classes of internet addresses: in class a, the high order bit is zero, the next 7 bits are the network, and the last 24 bits are the local address; in class b, the high order two bits are one-zero, the next 14 bits are the network and the last 16 bits are the local address; in class c, the high order three bits are one-one-zero, the next 21 bits are the network and the last 8 bits are the local address.

Care must be taken in mapping internet addresses to local net addresses; a single physical host must be able to act as if it were several distinct hosts to the extent of using several distinct internet addresses. Some hosts will also have several physical interfaces (multi-homing).

That is, provision must be made for a host to have several physical interfaces to the network with each having several logical internet addresses.

Examples of address mappings may be found in "Address Mappings" [5].

Fragmentation

Fragmentation of an internet datagram is necessary when it originates in a local net that allows a large packet size and must traverse a local net that limits packets to a smaller size to reach its destination.

An internet datagram can be marked "don't fragment." Any internet datagram so marked is not to be internet fragmented under any circumstances. If internet datagram marked don't fragment cannot be delivered to its destination without fragmenting it, it is to be discarded instead.

Fragmentation, transmission and reassembly across a local network which is invisible to the internet protocol module is called intranet fragmentation and may be used [6].

The internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed. The fragment offset field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The more-fragments flag indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams.

The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

To fragment a long internet datagram, an internet protocol module (for example, in a gateway), creates two new internet datagrams and copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on a 8 octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be). Call the number of 8 octet blocks in the first portion NFB (for Number of Fragment Blocks). The first portion of the data is placed in the first new internet datagram, and the total length field is set to the length of the first

datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new internet datagram is set to the value of that field in the long datagram plus NFB.

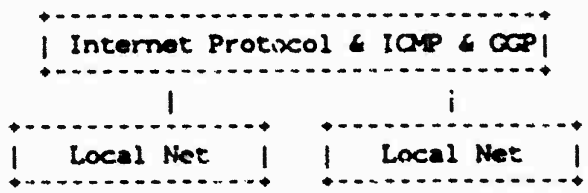
This procedure can be generalized for an n-way split, rather than the two-way split described.

To assemble the fragments of an internet datagram, an internet protocol module (for example at a destination host) combines internet datagrams that all have the same value for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero.

2.4. Gateways

Gateways implement internet protocol to forward datagrams between networks. Gateways also implement the Gateway to Gateway Protocol (GGP) [7] to coordinate routing and other internet control information.

In a gateway the higher level protocols need not be implemented and the GGP functions are added to the IP module.



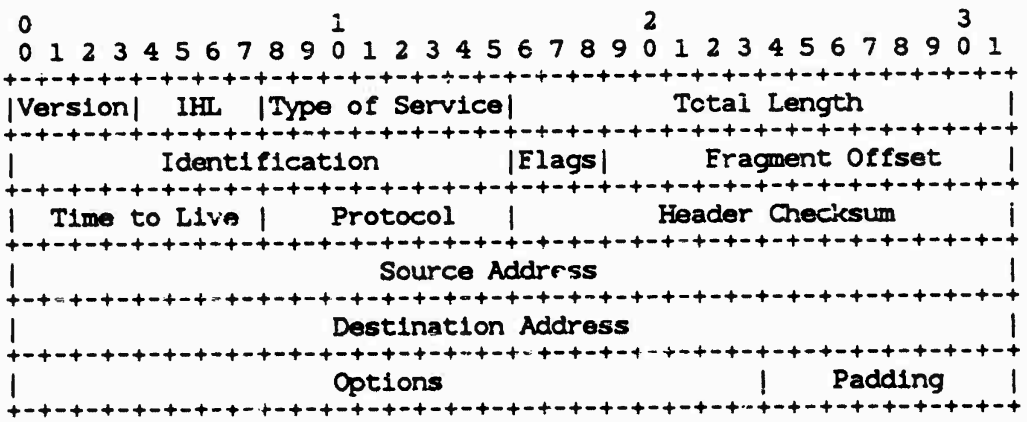
Gateway Protocols

Figure 3.

3. SPECIFICATION

3.1. Internet Header Format

A summary of the contents of the internet header follows:



Example Internet Datagram Header

Figure 4.

Note that each tick mark represents one bit position.

Version: 4 bits

The Version field indicates the format of the internet header. This document describes version 4.

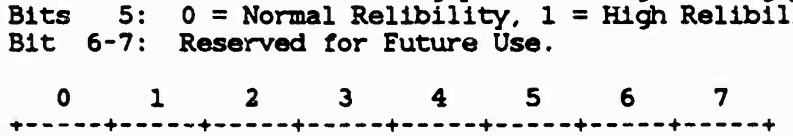
IHL: 4 bits

Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

Type of Service: 8 bits

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

```
Bits 0-2:  Precedence.
Bit    3:  0 = Normal Delay,       1 = Low Delay.
Bits   4:  0 = Normal Throughput,  1 = High Throughput.
Bits   5:  0 = Normal Relibility,  1 = High Relibility.
Bit  6-7:  Reserved for Future Use.
```



Precedence

```
111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine
```

The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set.

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET is given in "Service Mappings" [8].

The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.

Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

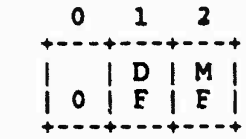
Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

Flags: 3 bits

Various Control Flags.

Bit 0: reserved, must be zero
Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.



Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs.

The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in "Assigned Numbers" [9].

Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

Source Address: 32 bits

The source address. See section 3.2.

Destination Address: 32 bits

The destination address. See section 3.2.

Options: variable

The options may appear or not in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation.

In some environments the security option may be required in all datagrams.

The option field is variable in length. There may be zero or more options. There are two cases for the format of an option:

Case 1: A single octet of option-type.

Case 2: An option-type octet, an option-length octet, and the actual option-data octets.

The option-length octet counts the option-type octet and the option-length octet as well as the option-data octets.

The option-type octet is viewed as having 3 fields:

1 bit copied flag,
2 bits option class,
5 bits option number.

The copied flag indicates that this option is copied into all fragments on fragmentation.

0 = not copied
1 = copied

The option classes are:

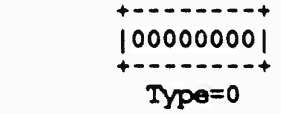
0 = control
1 = reserved for future use
2 = debugging and measurement
3 = reserved for future use

The following internet options are defined:

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0	-	End of Option list. This option occupies only 1 octet; it has no length octet.
0	1	-	No Operation. This option occupies only 1 octet; it has no length octet.
0	2	11	Security. Used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DOD requirements.
0	3	var.	Loose Source Routing. Used to route the internet datagram based on information supplied by the source.
0	9	var.	Strict Source Routing. Used to route the internet datagram based on information supplied by the source.
0	7	var.	Record Route. Used to trace the route an internet datagram takes.
0	8	4	Stream ID. Used to carry the stream identifier.
2	4	var.	Internet Timestamp.

Specific Option Definitions

End of Option List

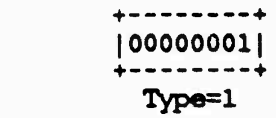


Type=0

This option indicates the end of the option list. This might not coincide with the end of the internet header according to the internet header length. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the internet header.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

No Operation

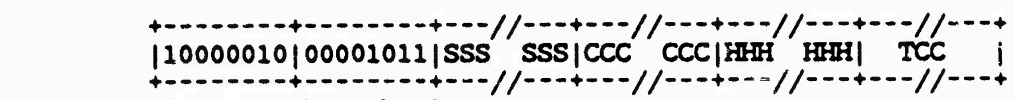


This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

Security

This option provides a way for hosts to send security, compartmentation, handling restrictions, and TCC (closed user group) parameters. The format for this option is as follows:



Security (S field): 16 bits

Specifies one of 16 levels of security (eight of which are reserved for future use).

00000000 00000000 - Unclassified
11110001 00110101 - Confidential
01111000 10011010 - EFTO
10111100 01001101 - MMMM
01011110 00100110 - PROG
10101111 00010011 - Restricted
11010111 10001000 - Secret
01101011 11000101 - Top Secret
00110101 11100010 - (Reserved for future use)
10011010 11110001 - (Reserved for future use)
01001101 01111000 - (Reserved for future use)
00100100 10111101 - (Reserved for future use)
00010011 01011110 - (Reserved for future use)
10001001 10101111 - (Reserved for future use)
11000100 11010110 - (Reserved for future use)
11100010 01101011 - (Reserved for future use)

Compartments (C field): 16 bits

An all zero value is used when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency.

Handling Restrictions (H field): 16 bits

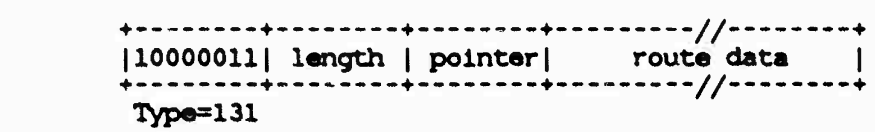
The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 65-19, "Standard Security Markings".

Transmission Control Code (TCC field): 24 bits

Provides a means to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs, and are available from HQ DCA Code 530.

Must be copied on fragmentation. This option appears at most once in a datagram.

Loose Source and Record Route



The loose source and record route (LSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a loose source route because the gateway or host IP is allowed to use any route of any number of other intermediate gateways to reach the next address in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Strict Source and Record Route

The strict source and record route (SSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the

recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

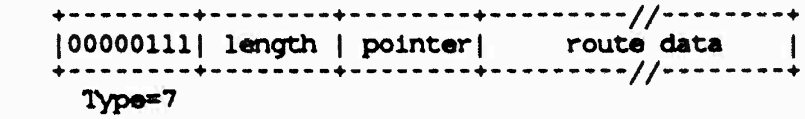
The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a strict source route because the gateway or host IP must send the datagram directly to the next address in the source route through only the directly connected network indicated in the next address to reach the next gateway or host specified in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Record Route



Type=7

The record route option provides a means to record the route of an internet datagram.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next area to store a route address. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A recorded route is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is

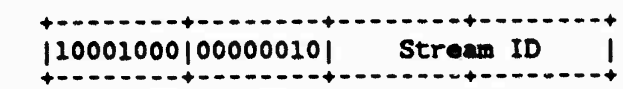
greater than the length, the recorded route data area is full. The originating host must compose this option with a large enough route data area to hold all the address expected. The size of the option does not change due to adding addresses. The intitial contents of the route data area must be zero.

When an internet module routes a datagram it checks to see if the record route option is present. If it is, it inserts its own internet address as known in the environment into which this datagram is being forwarded into the recorded route begining at the octet indicated by the pointer, and increments the pointer by four.

If the route data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the address into the recorded route. If there is some room but not enough room for a full address to be inserted, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

Not copied on fragmentation, goes in first fragment only. Appears at most once in a datagram.

Stream Identifier

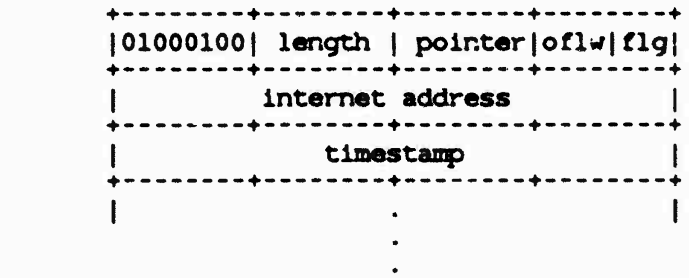


Type=136 Length=4

This option provides a way for the 16-bit SATNET stream identifier to be carried through networks that do not support the stream concept.

Must be copied on fragmentation. Appears at most once in a datagram.

Internet Timestamp



Type = 68

The Option Length is the number of octets in the option counting the type, length, pointer, and overflow/flag octets (maximum length 40).

The Pointer is the number of octets from the beginning of this option to the end of timestamps plus one (i.e., it points to the octet beginning the space for next timestamp). The smallest legal value is 5. The timestamp area is full when the pointer is greater than the length.

The Overflow (oflw) [4 bits] is the number of IP modules that cannot register timestamps due to lack of space.

The Flag (flg) [4 bits] values are

0 -- time stamps only, stored in consecutive 32-bit words,

1 -- each timestamp is preceded with internet address of the registering entity,

3 -- the internet address fields are prespecified. An IP module only registers its timestamp if it matches its own address with the next specified internet address.

The Timestamp is a right-justified, 32-bit timestamp in milliseconds since midnight UT. If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time may be inserted as a timestamp provided the high order bit of the timestamp field is set to one to indicate the use of a non-standard value.

The originating host must compose this option with a large enough timestamp data area to hold all the timestamp information expected. The size of the option does not change due to adding

timestamps. The intitial contents of the timestamp data area must be zero or internet address/zero pairs.

If the timestamp data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the timestamp, but the overflow count is incremented by one.

If there is some room but not enough room for a full timestamp to be inserted, or the overflow count itself overflows, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

The timestamp option is not copied upon fragmentation. It is carried in the first fragment. Appears at most once in a datagram.

Padding: variable

The internet header padding is used to ensure that the internet header ends on a 32 bit boundary. The padding is zero.

3.2. Discussion

The implementation of a protocol must be robust. Each implementation must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the protocol there is the possibility of differing interpretations. In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear).

The basic internet service is datagram oriented and provides for the fragmentation of datagrams at gateways, with reassembly taking place at the destination internet protocol module in the destination host. Of course, fragmentation and reassembly of datagrams within a network or by private agreement between the gateways of a network is also allowed since this is transparent to the internet protocols and the higher-level protocols. This transparent type of fragmentation and reassembly is termed "network-dependent" (or intranet) fragmentation and is not discussed further here.

Internet addresses distinguish sources and destinations to the host level and provide a protocol field as well. It is assumed that each protocol will provide for whatever multiplexing is necessary within a host.

Addressing

To provide for flexibility in assigning address to networks and allow for the large number of small to intermediate sized networks the interpretation of the address field is coded to specify a small number of networks with a large number of host, a moderate number of networks with a moderate number of hosts, and a large number of networks with a small number of hosts. In addition there is an escape code for extended addressing mode.

Address Formats:

High Order Bits	Format	Class
0	7 bits of net, 24 bits of host	a
10	14 bits of net, 16 bits of host	b
110	21 bits of net, 8 bits of host	c
111	escape to extended addressing mode	

A value of zero in the network field means this network. This is only used in certain ICMP messages. The extended addressing mode is undefined. Both of these features are reserved for future use.

The actual values assigned for network addresses is given in "Assigned Numbers" [9].

The local address, assigned by the local network, must allow for a single physical host to act as several distinct internet hosts. That is, there must be a mapping between internet host addresses and network/host interfaces that allows several internet addresses to correspond to one interface. It must also be allowed for a host to have several physical interfaces and to treat the datagrams from several of them as if they were all addressed to a single host.

Address mappings between internet addresses and addresses for ARPANET, SATNET, PRNET, and other networks are described in "Address Mappings" [5].

Fragmentation and Reassembly.

The internet identification field (ID) is used together with the source and destination address, and the protocol fields, to identify datagram fragments for reassembly.

The More Fragments flag bit (MF) is set if the datagram is not the last fragment. The Fragment Offset field identifies the fragment location, relative to the beginning of the original unfragmented datagram. Fragments are counted in units of 8 octets. The

fragmentation strategy is designed so than an unfragmented datagram has all zero fragmentation information (MF = 0, fragment offset = 0). If an internet datagram is fragmented, its data portion must be broken on 8 octet boundaries.

This format allows 2**13 = 8192 fragments of 8 octets each for a total of 65,536 octets. Note that this is consistent with the the datagram total length field (of course, the header is counted in the total length and not in the fragments).

When fragmentation occurs, some options are copied, but others remain with the first fragment only.

Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets.

Every internet destination must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled.

The fields which may be affected by fragmentation include:

(1) options field
(2) more fragments flag
(3) fragment offset
(4) internet header length field
(5) total length field
(6) header checksum

If the Don't Fragment flag (DF) bit is set, then internet fragmentation of this datagram is NOT permitted, although it may be discarded. This can be used to prohibit fragmentation in cases where the receiving host does not have sufficient resources to reassemble internet fragments.

One example of use of the Don't Fragment feature is to down line load a small host. A small host could have a boot strap program that accepts a datagram stores it in memory and then executes it.

The fragmentation and reassembly procedures are most easily described by examples. The following procedures are example implementations.

General notation in the following pseudo programs: "=<" means "less than or equal", "#" means "not equal", "=" means "equal", "<-" means "is set to". Also, "x to y" includes x and excludes y; for example, "4 to 7" would include 4, 5, and 6 (but not 7).

An Example Fragmentation Procedure

The maximum sized datagram that can be transmitted through the next network is called the maximum transmission unit (MTU).

If the total length is less than or equal the maximum transmission unit then submit this datagram to the next step in datagram processing; otherwise cut the datagram into two fragments, the first fragment being the maximum size, and the second fragment being the rest of the datagram. The first fragment is submitted to the next step in datagram processing, while the second fragment is submitted to this procedure in case it is still too large.

Notation:

```
FO    - Fragment Offset
IHL   - Internet Header Length
DF    - Don't Fragment flag
MF    - More Fragments flag
TL    - Total Length
OFO   - Old Fragment Offset
OIHL  - Old Internet Header Length
OMF   - Old More Fragments flag
OTL   - Old Total Length
NFB   - Number of Fragment Blocks
MTU   - Maximum Transmission Unit
```

Procedure:

```
IF TL =< MTU THEN Submit this datagram to the next step
     in datagram processing ELSE IF DF = 1 THEN discard the
datagram ELSE
To produce the first fragment:
(1)  Copy the original internet header;
(2)  OIHL <- IHL; OTL <- TL; OFO <- FO; OMF <- MF;
(3)  NFB <- (MTU-IHL*4)/8;
(4)  Attach the first NFB*8 data octets;
(5)  Correct the header:
     MF <- 1;  TL <- (IHL*4)+(NFB*8);
     Recompute Checksum;
(6)  Submit this fragment to the next step in
     datagram processing;
To produce the second fragment:
(7)  Selectively copy the internet header (some options
     are not copied, see option definitions);
(8)  Append the remaining data;
(9)  Correct the header:
     IHL <- (((OIHL*4)-(length of options not copied))+3)/4;
```

```
      TL <- OTL - NFB*8 - (OIHL-IHL)*4);
      FO <- OFO + NFB;  MF <- OMF;  Recompute Checksum;
(10)  Submit this fragment to the fragmentation test; DONE.
```

In the above procedure each fragment (except the last) was made the maximum allowable size. An alternative might produce less than the maximum size datagrams. For example, one could implement a fragmentation procedure that repeatly divided large datagrams in half until the resulting fragments were less than the maximum transmission unit size.

An Example Reassembly Procedure

For each datagram the buffer identifier is computed as the concatenation of the source, destination, protocol, and identification fields. If this is a whole datagram (that is both the fragment offset and the more fragments fields are zero), then any reassembly resources associated with this buffer identifier are released and the datagram is forwarded to the next step in datagram processing.

If no other fragment with this buffer identifier is on hand then reassembly resources are allocated. The reassembly resources consist of a data buffer, a header buffer, a fragment block bit table, a total data length field, and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length, and bits are set in the fragment block bit table corresponding to the fragment blocks received.

If this is the first fragment (that is the fragment offset is zero) this header is placed in the header buffer. If this is the last fragment (that is the more fragments field is zero) the total data length is computed. If this fragment completes the datagram (tested by checking the bits set in the fragment block table), then the datagram is sent to the next step in datagram processing; otherwise the timer is set to the maximum of the current timer value and the value of the time to live field from this fragment; and the reassembly routine gives up control.

If the timer runs out, the all reassembly resources for this buffer identifier are released. The initial setting of the timer is a lower bound on the reassembly waiting time. This is because the waiting time will be increased if the Time to Live in the arriving fragment is greater than the current timer value but will not be decreased if it is less. The maximum this timer value could reach is the maximum time to live (approximately 4.25 minutes). The current recommendation for the initial timer setting is 15 seconds. This may be changed as experience with

this protocol accumulates. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium; that is, data rate times timer value equals buffer size (e.g., 10Kb/s X 15s = 150Kb).

Notation:

```
FO    -  Fragment Offset
IHL   -  Internet Header Length
MF    -  More Fragments flag
TTL   -  Time To Live
NFB   -  Number of Fragment Blocks
TL    -  Total Length
TDL   -  Total Data Length
BUFID -  Buffer Identifier
RCVBT -  Fragment Received Bit Table
TLB   -  Timer Lower Bound
```

Procedure:

```
(1)  BUFID <- source|destination|protocol|identification;
(2)  IF FO = 0 AND MF = 0
(3)     THEN IF buffer with BUFID is allocated
(4)             THEN flush all reassembly for this BUFID;
(5)          Submit datagram to next step; DONE.
(6)     ELSE IF no buffer with BUFID is allocated
(7)             THEN allocate reassembly resources
                     with BUFID;
                     TIMER <- TLB; TDL <- 0;
(8)          put data from fragment into data buffer with
             BUFID from octet FO*8 to
                                  octet (TL-(IHL*4))+FO*8;
(9)          set RCVBT bits from FO
                                to FO+((TL-(IHL*4)+7)/8);
(10)         IF MF = 0 THEN TDL <- TL-(IHL*4)+(FO*8)
(11)         IF FO = 0 THEN put header in header buffer
(12)         IF TDL # 0
(13)          AND all RCVBT bits from 0
                                       to (TDL+7)/8 are set
(14)            THEN TL <- TDL+(IHL*4)
(15)                 Submit datagram to next step;
(16)                 free all reassembly resources
                     for this BUFID; DONE.
(17)         TIMER <- MAX(TIMER,TTL);
(18)         give up until next fragment or timer expires;
(19) timer expires: flush all reassembly with this BUFID; DONE.
```

In the case that two or more fragments contain the same data

either identically or through a partial overlap, this procedure will use the more recently arrived copy in the data buffer and datagram delivered.

Identification

The choice of the Identifier for a datagram is based on the need to provide a way to uniquely identify the fragments of a particular datagram. The protocol module assembling fragments judges fragments to belong to the same datagram if they have the same source, destination, protocol, and Identifier. Thus, the sender must choose the Identifier to be unique for this source, destination pair and protocol for the time the datagram (or any fragment of it) could be alive in the internet.

It seems then that a sending protocol module needs to keep a table of Identifiers, one entry for each destination it has communicated with in the last maximum packet lifetime for the internet.

However, since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is appropriate for some higher level protocols to choose the identifier. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment.

Type of Service

The type of service (TOS) is for internet service quality selection. The type of service is specified along the abstract parameters precedence, delay, throughput, and reliability. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses.

Precedence. An independent measure of the importance of this datagram.

Delay. Prompt delivery is important for datagrams with this indication.

Throughput. High data rate is important for datagrams with this indication.

Reliability. A higher level of effort to ensure delivery is important for datagrams with this indication.

For example, the ARPANET has a priority bit, and a choice between "standard" messages (type 0) and "uncontrolled" messages (type 3), (the choice between single packet and multipacket messages can also be considered a service parameter). The uncontrolled messages tend to be less reliably delivered and suffer less delay. Suppose an internet datagram is to be sent through the ARPANET. Let the internet type of service be given as:

Precedence:	5
Delay:	0
Throughput:	1
Reliability:	1

In this example, the mapping of these parameters to those available for the ARPANET would be to set the ARPANET priority bit on since the Internet precedence is in the upper half of its range, to select standard messages since the throughput and reliability requirements are indicated and delay is not. More details are given on service mappings in "Service Mappings" [8].

Time to Live

The time to live is set by the sender to the maximum time the datagram is allowed to be in the internet system. If the datagram is in the internet system longer than the time to live, then the datagram must be destroyed.

This field must be decreased at each point that the internet header is processed to reflect the time spent processing the datagram. Even if no local information is available on the time actually spent, the field must be decremented by 1. The time is measured in units of seconds (i.e. the value 1 means one second). Thus, the maximum time to live is 255 seconds or 4.25 minutes. Since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Some higher level reliable connection protocols are based on assumptions that old duplicate datagrams will not arrive after a certain time elapses. The TTL is a way for such protocols to have an assurance that their assumption is met.

Options

The options are optional in each datagram, but required in implementations. That is, the presence or absence of an option is the choice of the sender, but each internet module must be able to parse every option. There can be several options present in the option field.

The options might not end on a 32-bit boundary. The internet header must be filled out with octets of zeros. The first of these would be interpreted as the end-of-options option, and the remainder as internet header padding.

Every internet module must be able to act on every option. The Security Option is required if classified, restricted, or compartmented traffic is to be passed.

Checksum

The internet header checksum is recomputed if the internet header is changed. For example, a reduction of the time to live, additions or changes to internet options, or due to fragmentation. This checksum at the internet level is intended to protect the internet header fields from transmission errors.

There are some applications where a few data bit errors are acceptable while retransmission delays are not. If the internet protocol enforced data correctness such applications could not be supported.

Errors

Internet protocol errors may be reported via the ICMP messages [3].

3.3. Interfaces

The functional description of user interfaces to the IP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different IP implementations may have different user interfaces. However, all IPs must provide a certain minimum set of services to guarantee that all IP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all IP implementations.

Internet protocol interfaces on one side to the local network and on the other side to either a higher level protocol or an application program. In the following, the higher level protocol or application

program (or even a gateway program) will be called the "user" since it is using the internet module. Since internet protocol is a datagram protocol, there is minimal memory or state maintained between datagram transmissions, and each call on the internet protocol module by the user supplies all information necessary for the IP to perform the service requested.

An Example Upper Level Interface

The following two example calls satisfy the requirements for the user to internet protocol module communication ("=>" means returns):

```
SEND (src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt => result)

  where:

    src = source address
    dst = destination address
    prot = protocol
    TOS = type of service
    TTL = time to live
    BufPTR = buffer pointer
    len = length of buffer
    Id  = Identifier
    DF = Don't Fragment
    opt = option data
    result = response
      OK = datagram sent ok
      Error = error in arguments or local network error

  Note that the precedence is included in the TOS and the
  security/compartment is passed as an option.

RECV (BufPTR, prot, => result, src, dst, TOS, len, opt)

  where:

    BufPTR = buffer pointer
    prot = protocol
    result = response
      OK = datagram received ok
      Error = error in arguments
    len = length of buffer
    src = source address
    dst = destination address
    TOS = type of service
    opt = option data
```

When the user sends a datagram, it executes the SEND call supplying all the arguments. The internet protocol module, on receiving this call, checks the arguments and prepares and sends the message. If the arguments are good and the datagram is accepted by the local network, the call returns successfully. If either the arguments are bad, or the datagram is not accepted by the local network, the call returns unsuccessfully. On unsuccessful returns, a reasonable report must be made as to the cause of the problem, but the details of such reports are up to individual implementations.

When a datagram arrives at the internet protocol module from the local network, either there is a pending RECV call from the user addressed or there is not. In the first case, the pending call is satisfied by passing the information from the datagram to the user. In the second case, the user addressed is notified of a pending datagram. If the user addressed does not exist, an ICMP error message is returned to the sender, and the data is discarded.

The notification of a user may be via a pseudo interrupt or similar mechanism, as appropriate in the particular operating system environment of the implementation.

A user's RECV call may then either be immediately satisfied by a pending datagram, or the call may be pending until a datagram arrives.

The source address is included in the send call in case the sending host has several addresses (multiple physical connections or logical addresses). The internet module must check to see that the source address is one of the legal address for this host.

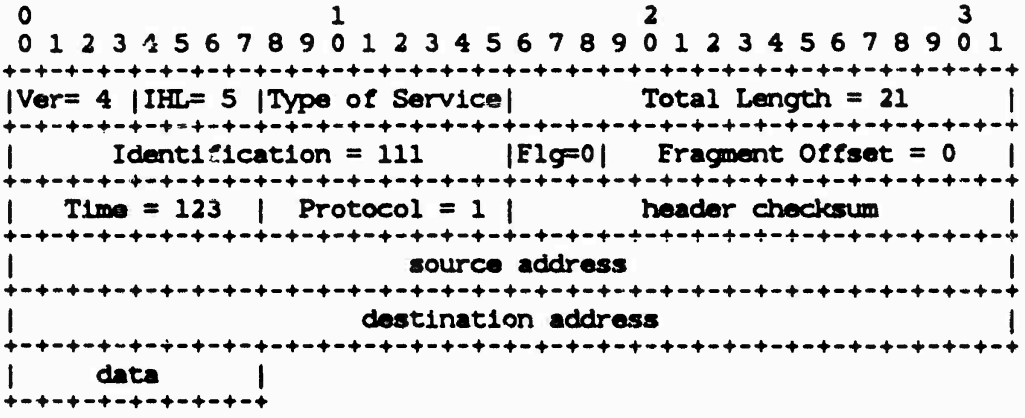
An implementation may also allow or require a call to the internet module to indicate interest in or reserve exclusive use of a class of datagrams (e.g., all those with a certain value in the protocol field).

This section functionally characterizes a USER/IP interface. The notation used is similar to most procedure of function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs), or any other form of interprocess communication.

APPENDIX A: Examples & Scenarios

Example 1:

This is an example of the minimal data carrying internet datagram:



Example Internet Datagram

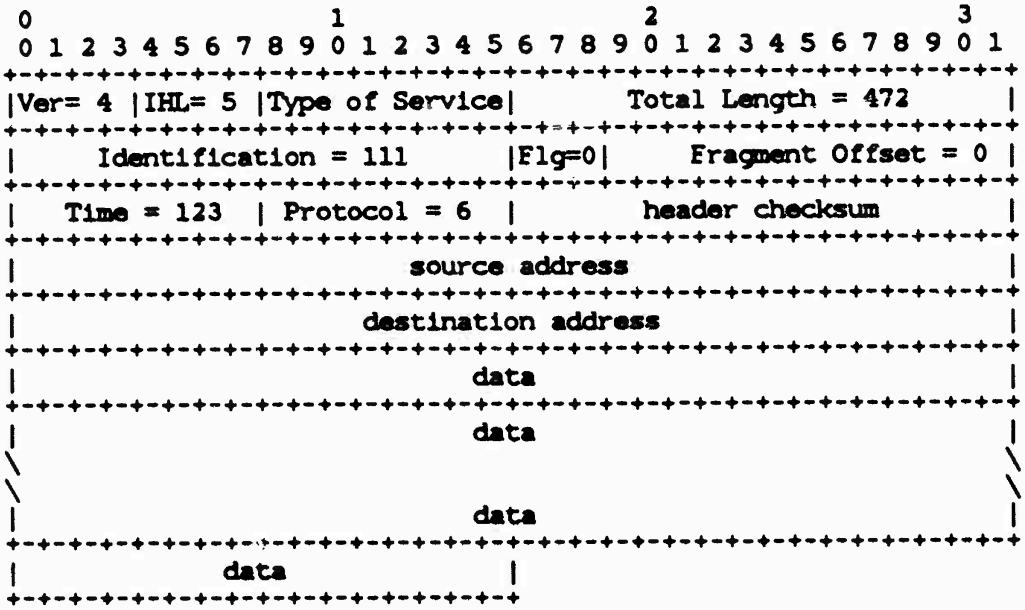
Figure 5.

Note that each tick mark represents one bit position.

This is a internet datagram in version 4 of internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 21 octets. This datagram is a complete datagram (not a fragment).

Example 2:

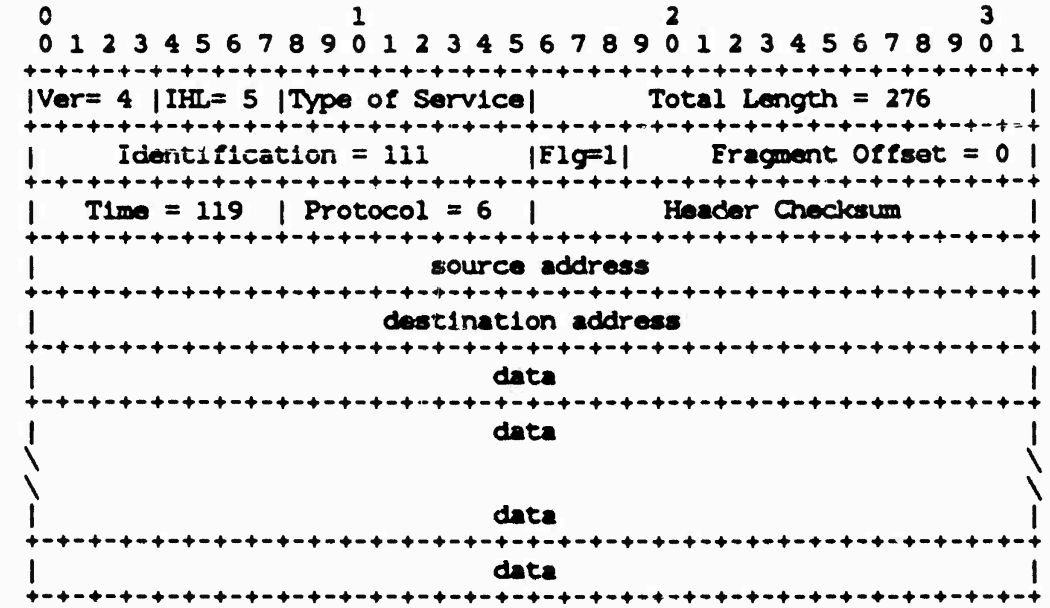
In this example, we show first a moderate size internet datagram (452 data octets), then two internet fragments that might result from the fragmentation of this datagram if the maximum sized transmission allowed were 280 octets.



Example Internet Datagram

Figure 6.

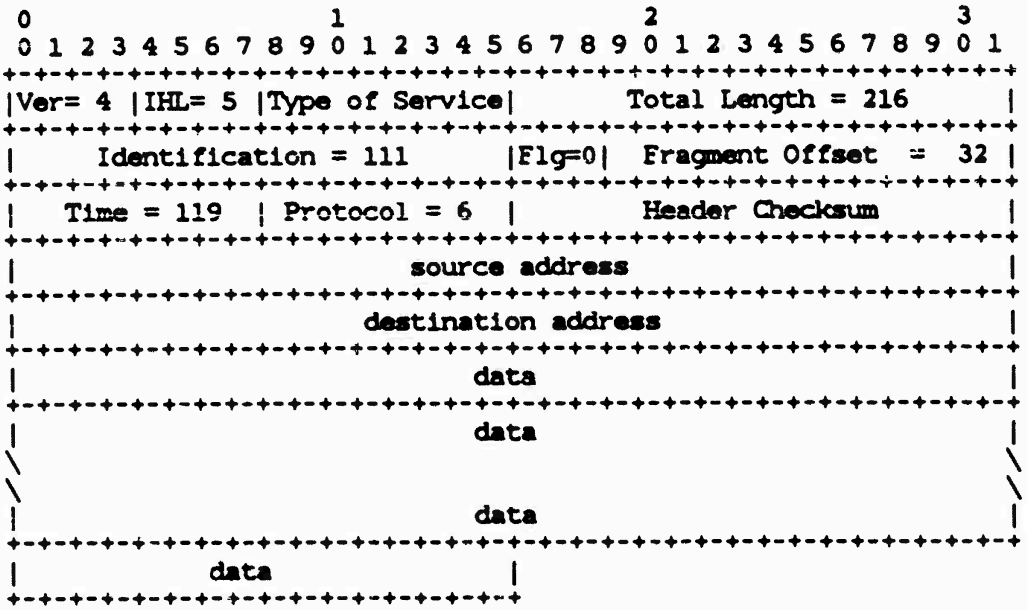
Now the first fragment that results from splitting the datagram after 256 data octets.



Example Internet Fragment

Figure 7.

And the second fragment.

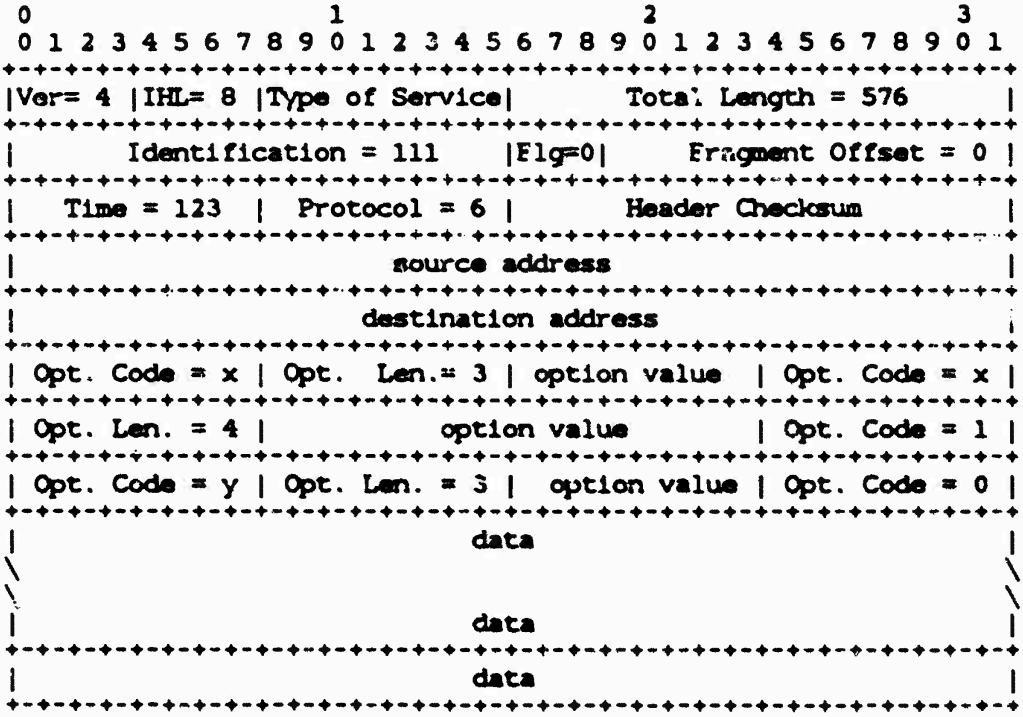


Example Internet Fragment

Figure 8.

Example 3:

Here, we show an example of a datagram containing options:

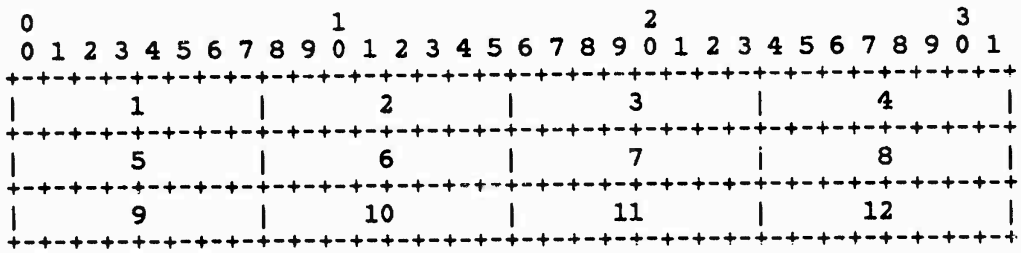


Example Internet Datagram

Figure 9.

APPENDIX B: Data Transmission Order

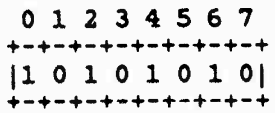
The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they are numbered.



Transmission Order of Bytes

Figure 10.

Whenever an octet represents a numeric quantity the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).



Significance of Bits

Figure 11.

Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

GLOSSARY

1822
BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ARPANET leader
The control information on an ARPANET message at the host-IMP interface.

ARPANET message
The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet
A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

Destination
The destination address, an internet header field.

DF
The Don't Fragment bit carried in the flags field.

Flags
An internet header field carrying various control flags.

Fragment Offset
This internet header field indicates where in the internet datagram a fragment belongs.

GGP
Gateway to Gateway Protocol, the protocol used primarily between gateways to control routing and other gateway functions.

header
Control information at the beginning of a message, segment, datagram, packet or block of data.

ICMP
Internet Control Message Protocol, implemented in the internet module, the ICMP is used from gateways to hosts and between hosts to report errors and make routing suggestions.

September 1981

Internet Protocol
Glossary

Identification
An internet header field carrying the identifying value assigned by the sender to aid in assembling the fragments of a datagram.

IHL
The internet header field Internet Header Length is the length of the internet header measured in 32 bit words.

IMP
The Interface Message Processor, the packet switch of the ARPANET.

Internet Address
A four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

internet datagram
The unit of data exchanged between a pair of internet modules (includes the internet header).

internet fragment
A portion of the data of an internet datagram with an internet header.

Local Address
The address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

MF
The More-Fragments Flag carried in the internet header flags field.

module
An implementation, usually in software, of a protocol or other procedure.

more-fragments flag
A flag indicating whether or not this internet datagram contains the end of an internet datagram, carried in the internet header Flags field.

NFB
The Number of Fragment Blocks in a the data portion of an internet fragment. That is, the length of a portion of data measured in 8 octet units.

September 1981

Internet Protocol
Glossary

octet
An eight bit byte.

Options
The internet header Options field may contain several options, and each option may be several octets in length.

Padding
The internet header Padding field is used to ensure that the data begins on 32 bit word boundary. The padding is zero.

Protocol
In this document, the next higher level protocol identifier, an internet header field.

Rest
The local address portion of an Internet Address.

Source
The source address, an internet header field.

TCP
Transmission Control Protocol: A host-to-host protocol for reliable communication in internet environments.

TCP Segment
The unit of data exchanged between TCP modules (including the TCP header).

TFTP
Trivial File Transfer Protocol: A simple file transfer protocol built on UDP.

Time to Live
An internet header field which indicates the upper bound on how long this internet datagram may exist.

TOS
Type of Service

Total Length
The internet header field Total Length is the length of the datagram in octets including internet header and data.

TTL
Time to Live

September 1981

Internet Protocol
Glossary

Type of Service
An internet header field which indicates the type (or quality) of service for this internet datagram.

UDP
User Datagram Protocol: A user level protocol for transaction oriented applications.

User
The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.

Version
The Version field indicates the format of the internet header.

REFERENCES

[1] Cerf, V., "The Catenet Model for Internetworking," Information Processing Techniques Office, Defense Advanced Research Projects Agency, IEN 48, July 1978.

[2] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," BBN Technical Report 1822, Revised May 1978.

[3] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," RFC 792, USC/Information Sciences Institute, September 1981.

[4] Shoch, J., "Inter-Network Naming, Addressing, and Routing," COMPCON, IEEE Computer Society, Fall 1978.

[5] Postel, J., "Address Mappings," RFC 796, USC/Information Sciences Institute, September 1981.

[6] Shoch, J., "Packet Fragmentation in Inter-Network Protocols," Computer Networks, v. 3, n. 1, February 1979.

[7] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.

[8] Postel, J., "Service Mappings," RFC 795, USC/Information Sciences Institute, September 1981.

[9] Postel, J., "Assigned Numbers," RFC 790, USC/Information Sciences Institute, September 1981.

Network Working Group
Request for Comments: 792

J. Postel
ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

INTERNET CONTROL MESSAGE PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

Introduction

The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

The Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols that use IP must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. Also ICMP messages are only sent about errors in handling fragment zero of fragemented datagrams. (Fragment zero has the fragment offeset equal zero).

Message Formats

ICMP messages are sent using the basic IP header. The first octet of the data portion of the datagram is a ICMP type field; the value of this field determines the format of the remaining data. Any field labeled "unused" is reserved for later extensions and must be zero when sent, but receivers should not use these fields (except to include them in the checksum). Unless otherwise noted under the individual format descriptions, the values of the internet header fields are as follows:

Version

4

IHL

Internet header length in 32-bit words.

Type of Service

0

Total Length

Length of internet header and data in octets.

Identification, Flags, Fragment Offset

Used in fragmentation, see [1].

Time to Live

Time to live in seconds; as this field is decremented at each machine in which the datagram is processed, the value in this field should be at least as great as the number of gateways which this datagram will traverse.

Protocol

ICMP = 1

Header Checksum

The 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

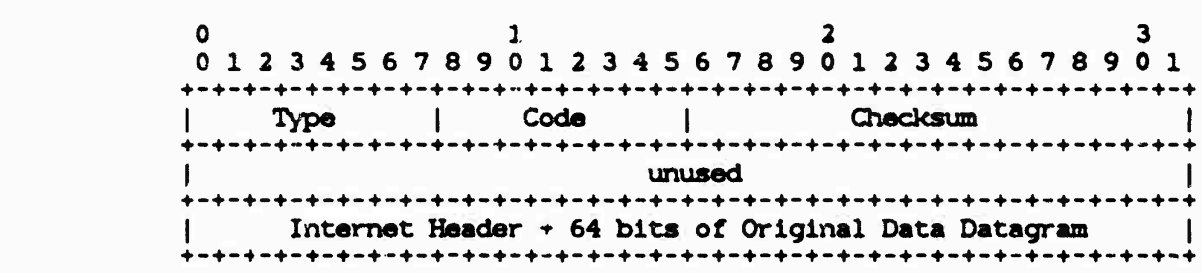
Source Address

The address of the gateway or host that composes the ICMP message. Unless otherwise noted, this can be any of a gateway's addresses.

Destination Address

The address of the gateway or host to which the message should be sent.

Destination Unreachable Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

3

Code

0 = net unreachable;

1 = host unreachable;

2 = protocol unreachable;

3 = port unreachable;

4 = fragmentation needed and DF set;

5 = source route failed.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original

datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

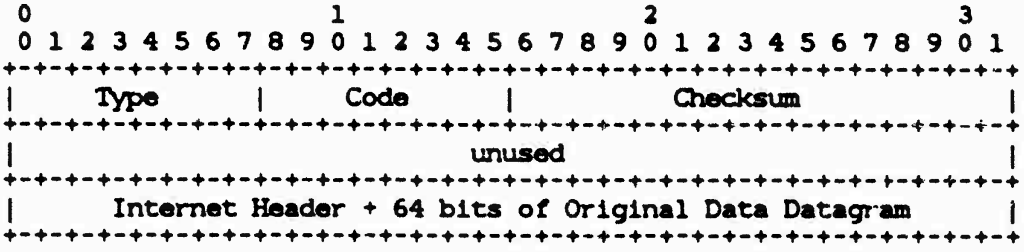
If, according to the information in the gateway's routing tables, the network specified in the internet destination field of a datagram is unreachable, e.g., the distance to the network is infinity, the gateway may send a destination unreachable message to the internet source host of the datagram. In addition, in some networks, the gateway may be able to determine if the internet destination host is unreachable. Gateways in these networks may send destination unreachable messages to the source host when the destination host is unreachable.

If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host.

Another case is when a datagram must be fragmented to be forwarded by a gateway yet the Don't Fragment flag is on. In this case the gateway must discard the datagram and may return a destination unreachable message.

Codes 0, 1, 4, and 5 may be received from a gateway. Codes 2 and 3 may be received from a host.

Time Exceeded Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

11

Code

0 = time to live exceeded in transit;

1 = fragment reassembly time exceeded.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

If the gateway processing a datagram finds the time to live field

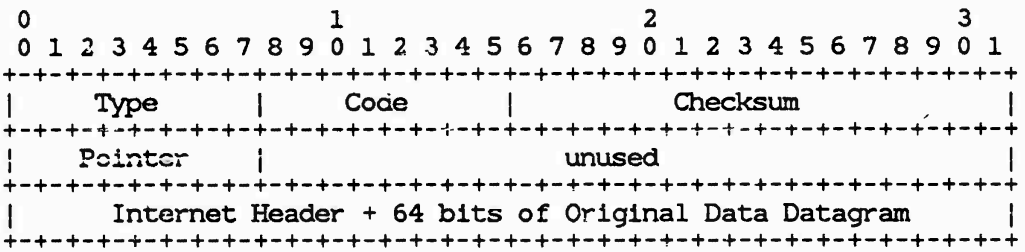
is zero it must discard the datagram. The gateway may also notify the source host via the time exceeded message.

If a host reassembling a fragmented datagram cannot complete the reassembly due to missing fragments within its time limit it discards the datagram, and it may send a time exceeded message.

If fragment zero is not available then no time exceeded need be sent at all.

Code 0 may be received from a gateway. Code 1 may be received from a host.

Parameter Problem Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

12

Code

0 = pointer indicates the error.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Pointer

If code = 0, identifies the octet where an error was detected.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

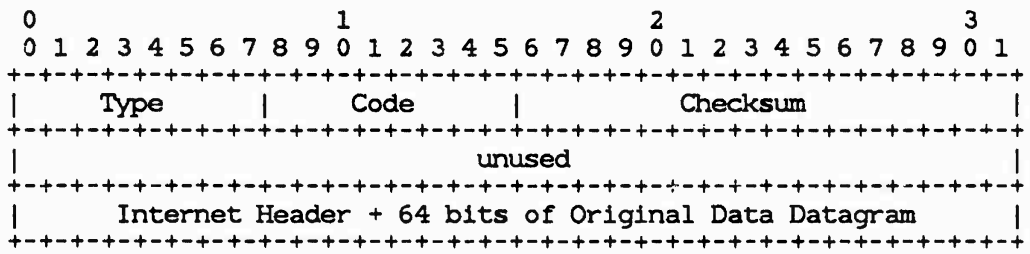
Description

If the gateway or host processing a datagram finds a problem with the header parameters such that it cannot complete processing the datagram it must discard the datagram. One potential source of such a problem is with incorrect arguments in an option. The gateway or host may also notify the source host via the parameter problem message. This message is only sent if the error caused the datagram to be discarded.

The pointer identifies the octet of the original datagram's header where the error was detected (it may be in the middle of an option). For example, 1 indicates something is wrong with the Type of Service, and (if there are options present) 20 indicates something is wrong with the type code of the first option.

Code 0 may be received from a gateway or a host.

Source Quench Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

4

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

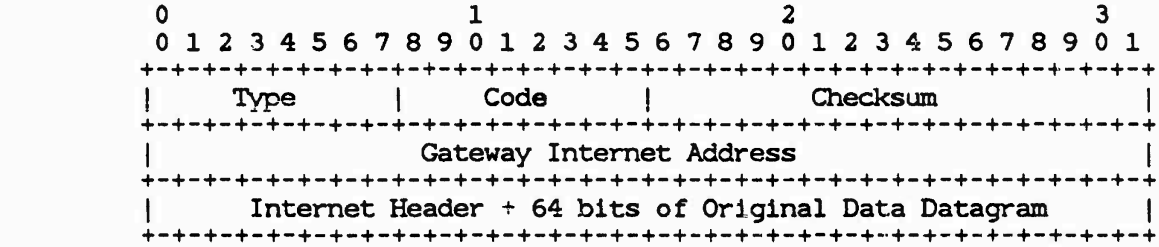
A gateway may discard internet datagrams if it does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination network. If a gateway

discards a datagram, it may send a source quench message to the internet source host of the datagram. A destination host may also send a source quench message if datagrams arrive too fast to be processed. The source quench message is a request to the host to cut back the rate at which it is sending traffic to the internet destination. The gateway may send a source quench message for every message that it discards. On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the gateway. The source host can then gradually increase the rate at which it sends traffic to the destination until it again receives source quench messages.

The gateway or host may send the source quench message when it approaches its capacity limit rather than waiting until the capacity is exceeded. This means that the data datagram which triggered the source quench message may be delivered.

Code 0 may be received from a gateway or a host.

Redirect Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

5

Code

0 = Redirect datagrams for the Network.

1 = Redirect datagrams for the Host.

2 = Redirect datagrams for the Type of Service and Network.

3 = Redirect datagrams for the Type of Service and Host.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Gateway Internet Address

Address of the gateway to which traffic for the network specified in the internet destination network field of the original datagram's data should be sent.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

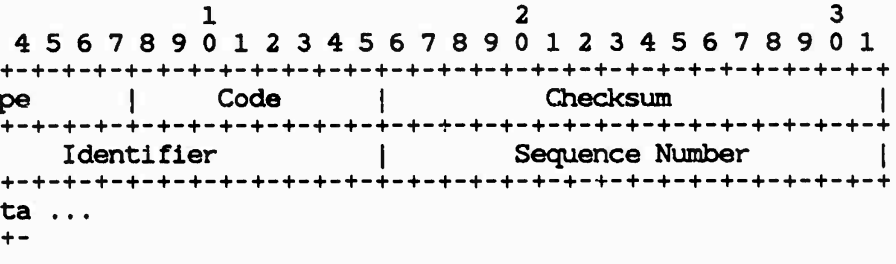
Description

The gateway sends a redirect message to a host in the following situation. A gateway, G1, receives an internet datagram from a host on a network to which the gateway is attached. The gateway, G1, checks its routing table and obtains the address of the next gateway, G2, on the route to the datagram's internet destination network, X. If G2 and the host identified by the internet source address of the datagram are on the same network, a redirect message is sent to the host. The redirect message advises the host to send its traffic for network X directly to gateway G2 as this is a shorter path to the destination. The gateway forwards the original datagram's data to its internet destination.

For datagrams with the IP source route options and the gateway address in the destination address field, a redirect message is not sent even if there is a better route to the ultimate destination than the next address in the source route.

Codes 0, 1, 2, and 3 may be received from a gateway.

Echo or Echo Reply Message



IP Fields:

Addresses

The address of the source in an echo message will be the destination of the echo reply message. To form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed.

IP Fields:

Type

8 for echo message;

0 for echo reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. If the total length is odd, the received data is padded with one octet of zeros for computing the checksum. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching echos and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching echos and replies, may be zero.

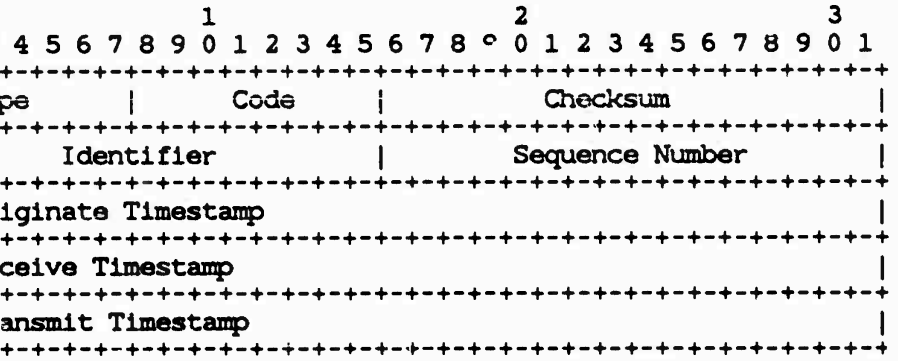
Description

The data received in the echo message must be returned in the echo reply message.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply.

Code 0 may be received from a gateway or a host.

Timestamp or Timestamp Reply Message



IP Fields:

Addresses

The address of the source in a timestamp message will be the destination of the timestamp reply message. To form a timestamp reply message, the source and destination addresses are simply reversed, the type code changed to 14, and the checksum recomputed.

IP Fields:

Type

13 for timestamp message;

14 for timestamp reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complemant of the one's complement sum of the ICMP message starting with the ICMP Type. For computing che checksum , the checksum field should be zero. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching timestamp and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching timestamp and replies, may be zero.

Description

The data received (a timestamp) in the message is returned in the reply together with an additional timestamp. The timestamp is 32 bits of milliseconds since midnight UT. One use of these timestamps is described by Mills [5].

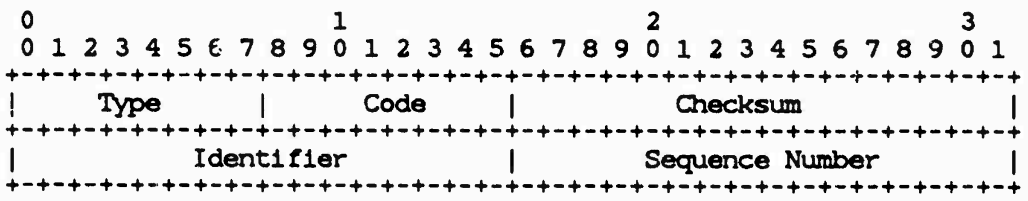
The Originate Timestamp is the time the sender last touched the message before sending it, the Receive Timestamp is the time the echoer first touched it on receipt, and the Transmit Timestamp is the time the echoer last touched the message on sending it.

If the time is not available in miliseconds or cannot be provided with respect to midnight UT then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

Information Request or Information Reply Message



IP Fields:

Addresses

The address of the source in a information request message will be the destination of the information reply message. To form a information reply message, the source and destination addresses are simply reversed, the type code changed to 16, and the checksum recomputed.

IP Fields:

Type

15 for information request message;

16 for information reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching request and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching request and replies, may be zero.

Description

This message may be sent with the source network in the IP header source and destination address fields zero (which means "this" network). The replying IP module should send the reply with the addresses fully specified. This message is a way for a host to find out the number of the network it is on.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

Summary of Message Types

0 Echo Reply

3 Destination Unreachable

4 Source Quench

5 Redirect

8 Echo

11 Time Exceeded

12 Parameter Problem

13 Timestamp

14 Timestamp Reply

15 Information Request

16 Information Reply

References

[1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

[2] Cerf, V., "The Catenet Model for Internetworking," IEN 48, Information Processing Techniques Office, Defense Advanced Research Projects Agency, July 1978.

[3] Strazisar, V., "Gateway Routing: An Implementation Specification", IEN 30, Bolt Beranek and Newman, April 1979.

[4] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.

[5] Mills, D., "DCNET Internet Clock Service," RFC 778, COMSAT Laboratories, April 1981.

SECTION 7. HOST LEVEL PROTOCOLS

This section contains RFCs pertaining to major host protocols, minor host protocols, and gateway protocols.

RFC 768

J. Postel
ISI
28 August 1980

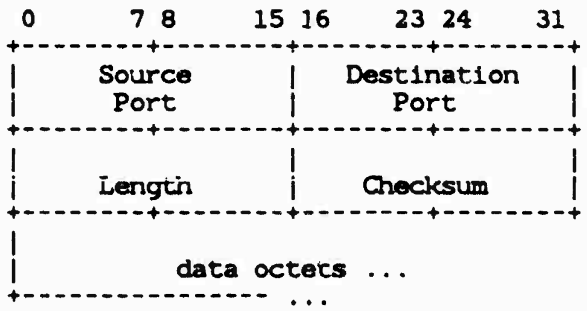
User Datagram Protocol

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format



User Datagram Header Format

Fields

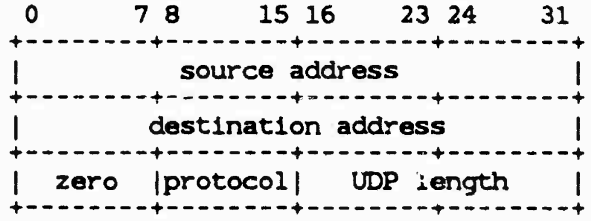
Source Port is an optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

Destination Port has a meaning within the context of a particular internet destination address.

Length is the length in octets of this user datagram including this header and the data. (This means the minimum value of the length is eight.)

Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

The pseudo header conceptually prefixed to the UDP header contains the source address, the destination address, the protocol, and the UDP length. This information gives protection against misrouted datagrams. This checksum procedure is the same as is used in TCP.



If the computed checksum is zero, it is transmitted as all ones (the equivalent in one's complement arithmetic). An all zero transmitted checksum value means that the transmitter generated no checksum (for debugging or for higher level protocols that don't care).

User Interface

A user interface should allow

the creation of new receive ports,

receive operations on the receive ports that return the data octets and an indication of source port and source address,

and an operation that allows a datagram to be sent, specifying the data, source and destination ports and addresses to be sent.

IP Interface

The UDP module must be able to determine the source and destination internet addresses and the protocol field from the internet header. One possible UDP/IP interface would return the whole internet datagram including all of the internet header in response to a receive operation. Such an interface would also allow the UDP to pass a full internet datagram complete with header to the IP to send. The IP would verify certain fields for consistency and compute the internet header checksum.

Protocol Application

The major uses of this protocol is the Internet Name Server [3], and the Trivial File Transfer [4].

Protocol Number

This is protocol 17 (21 octal) when used in the Internet Protocol. Other protocol numbers are listed in [5].

References
---

[1] Postel, J., "Internet Protocol," RFC 760, USC/Information Sciences Institute, January 1980.

[2] Postel, J., "Transmission Control Protocol," RFC 761, USC/Information Sciences Institute. January 1980.

[3] Postel, J., "Internet Name Server," USC/Information Sciences Institute, IEN 116, August 1979.

[4] Sollins, K., "The TFTP Protocol," Massachusetts Institute of Technology, IEN 133, January 1980.

[5] Postel, J., "Assigned Numbers," USC/Information Sciences Institute, RFC 762, January 1980.

RFC: 793

TRANSMISSION CONTROL PROTOCOL

DARPA INTERNET PROGRAM

PROTOCOL SPECIFICATION

September 1981

prepared for

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

by

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

TABLE OF CONTENTS

September 1981

Transmission Control Protocol

PREFACE

This document describes the DoD Standard Transmission Control Protocol (TCP). There have been nine earlier editions of the ARPA TCP specification on which this standard is based, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition clarifies several details and removes the end-of-letter buffer-size adjustments, and redescribes the letter mechanism as a push function.

Jon Po. .el

Editor

RFC: 793
Replaces: RFC 761
IENs: 129, 124, 112, 81,
55, 44, 40, 27, 21, 5

TRANSMISSION CONTROL PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

1. INTRODUCTION

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the Transmission Control Protocol, the program that implements it, and its interface to programs or users that require its services.

1.1. Motivation

Computer communication systems are playing an increasingly important role in military, government, and civilian environments. This document focuses its attention primarily on military computer communication requirements, especially robustness in the presence of communication unreliability and availability in the presence of congestion, but many of these problems are found in the civilian and government sector as well.

As strategic and tactical computer communication networks are developed and deployed, it is essential to provide means of interconnecting them and to provide standard interprocess communication protocols which can support a broad range of applications. In anticipation of the need for such standards, the Deputy Undersecretary of Defense for Research and Engineering has declared the Transmission Control Protocol (TCP) described herein to be a basis for DoD-wide inter-process communication protocol standardization.

TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes it can obtain a simple, potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks.

September 1981

Transmission Control Protocol
Introduction

TCP is based on concepts first described by Cerf and Kahn in [1]. The TCP fits into a layered protocol architecture just above a basic Internet Protocol [2] which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram "envelopes". The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentation or reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting gateways. The internet protocol also carries information on the precedence, security classification and compartmentation of the TCP segments, so this information can be communicated end-to-end across multiple networks.

Protocol Layering

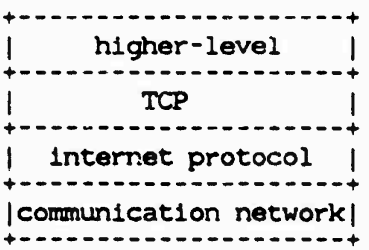


Figure 1

Much of this document is written in the context of TCP implementations which are co-resident with higher level protocols in the host computer. Some computer systems will be connected to networks via front-end computers which house the TCP and internet protocol layers, as well as network specific software. The TCP specification describes an interface to the higher level protocols which appears to be implementable even for the front-end case, as long as a suitable host-to-front end protocol is implemented.

1.2. Scope

The TCP is intended to provide a reliable process-to-process communication service in a multinetwork environment. The TCP is intended to be a host-to-host protocol in common use in multiple networks.

1.3. About this Document

This document represents a specification of the behavior required of any TCP implementation, both in its interactions with higher level protocols and in its interactions with other TCPs. The rest of this

section offers a very brief view of the protocol interfaces and operation. Section 2 summarizes the philosophical basis for the TCP design. Section 3 offers both a detailed description of the actions required of TCP when various events occur (arrival of new segments, user calls, errors, etc.) and the details of the formats of TCP segments.

1.4. Interfaces

The TCP interfaces on one side to user or application processes and on the other side to a lower level protocol such as Internet Protocol.

The interface between an application process and the TCP is illustrated in reasonable detail. This interface consists of a set of calls much like the calls an operating system provides to an application process for manipulating files. For example, there are calls to open and close connections and to send and receive data on established connections. It is also expected that the TCP can asynchronously communicate with application programs. Although considerable freedom is permitted to TCP implementors to design interfaces which are appropriate to a particular operating system environment, a minimum functionality is required at the TCP/user interface for any valid implementation.

The interface between TCP and lower level protocol is essentially unspecified except that it is assumed there is a mechanism whereby the two levels can asynchronously pass information to each other. Typically, one expects the lower level protocol to specify this interface. TCP is designed to work in a very general environment of interconnected networks. The lower level protocol which is assumed throughout this document is the Internet Protocol [2].

1.5. Operation

As noted above, the primary purpose of the TCP is to provide reliable, securable logical circuit or connection service between pairs of processes. To provide this service on top of a less reliable internet communication system requires facilities in the following areas:

Basic Data Transfer
Reliability
Flow Control
Multiplexing
Connections
Precedence and Security

The basic operation of the TCP in each of these areas is described in the following paragraphs.

September 1981

Transmission Control Protocol
Introduction

Basic Data Transfer:

The TCP is able to transfer a continuous stream of octets in each direction between its users by packaging some number of octets into segments for transmission through the internet system. In general, the TCPs decide when to block and forward data at their own convenience.

Sometimes users need to be sure that all the data they have submitted to the TCP has been transmitted. For this purpose a push function is defined. To assure that data submitted to a TCP is actually transmitted the sending user indicates that it should be pushed through to the receiving user. A push causes the TCPs to promptly forward and deliver data up to that point to the receiver. The exact push point might not be visible to the receiving user and the push function does not supply a record boundary marker.

Reliability:

The TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the internet communication system. This is achieved by assigning a sequence number to each octet transmitted, and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.

As long as the TCPs continue to function properly and the internet system does not become completely partitioned, no transmission errors will affect the correct delivery of data. TCP recovers from internet communication system errors.

Flow Control:

TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.

Multiplexing:

To allow for many processes within a single Host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a socket. A pair of sockets uniquely identifies each connection. That is, a socket may be simultaneously used in multiple connections.

The binding of ports to processes is handled independently by each Host. However, it proves useful to attach frequently used processes (e.g., a "logger" or timesharing service) to fixed sockets which are made known to the public. These services can then be accessed through the known addresses. Establishing and learning the port addresses of other processes may involve more dynamic mechanisms.

Connections:

The reliability and flow control mechanisms described above require that TCPs initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is uniquely specified by a pair of sockets identifying its two sides.

When two processes wish to communicate, their TCP's must first establish a connection (initialize the status information on each side). When their communication is complete, the connection is terminated or closed to free the resources for other uses.

Since connections must be established between unreliable hosts and over the unreliable internet communication system, a handshake mechanism with clock-based sequence numbers is used to avoid erroneous initialization of connections.

Precedence and Security:

The users of TCP may indicate the security and precedence of their communication. Provision is made for default values to be used when these features are not needed.

September 1981

Transmission Control Protocol

[Page 6]

2. PHILOSOPHY

2.1. Elements of the Internetwork System

The internetwork environment consists of hosts connected to networks which are in turn interconnected via gateways. It is assumed here that the networks may be either local networks (e.g., the ETHERNET) or large networks (e.g., the ARPANET), but in any case are based on packet switching technology. The active agents that produce and consume messages are processes. Various levels of protocols in the networks, the gateways, and the hosts support an interprocess communication system that provides two-way data flow on logical connections between process ports.

The term packet is used generically here to mean the data of one transaction between a host and its network. The format of data blocks exchanged within the a network will generally not be of concern to us.

Hosts are computers attached to a network, and from the communication network's point of view, are the sources and destinations of packets. Processes are viewed as the active elements in host computers (in accordance with the fairly common definition of a process as a program in execution). Even terminals and files or other I/O devices are viewed as communicating with each other through the use of processes. Thus, all communication is viewed as inter-process communication.

Since a process may need to distinguish among several communication streams between itself and another process (or processes), we imagine that each process may have a number of ports through which it communicates with the ports of other processes.

2.2. Model of Operation

Processes transmit data by calling on the TCP and passing buffers of data as arguments. The TCP packages the data from these buffers into segments and calls on the internet module to transmit each segment to the destination TCP. The receiving TCP places the data from a segment into the receiving user's buffer and notifies the receiving user. The TCPs include control information in the segments which they use to ensure reliable ordered data transmission.

The model of internet communication is that there is an internet protocol module associated with each TCP which provides an interface to the local network. This internet module packages TCP segments inside internet datagrams and routes these datagrams to a destination internet module or intermediate gateway. To transmit the datagram through the local network, it is embedded in a local network packet.

The packet switches may perform further packaging, fragmentation, or

other operations to achieve the delivery of the local packet to the destination internet module.

At a gateway between networks, the internet datagram is "unwrapped" from its local packet and examined to determine through which network the internet datagram should travel next. The internet datagram is then "wrapped" in a local packet suitable to the next network and routed to the next gateway, or to the final destination.

A gateway is permitted to break up an internet datagram into smaller internet datagram fragments if this is necessary for transmission through the next network. To do this, the gateway produces a set of internet datagrams; each carrying a fragment. Fragments may be further broken into smaller fragments at subsequent gateways. The internet datagram fragment format is designed so that the destination internet module can reassemble fragments into internet datagrams.

A destination internet module unwraps the segment from the datagram (after reassembling the datagram, if necessary) and passes it to the destination TCP.

This simple model of the operation glosses over many details. One important feature is the type of service. This provides information to the gateway (or internet module) to guide it in selecting the service parameters to be used in traversing the next network. Included in the type of service information is the precedence of the datagram. Datagrams may also carry security information to permit host and gateways that operate in multilevel secure environments to properly segregate datagrams for security considerations.

2.3. The Host Environment

The TCP is assumed to be a module in an operating system. The users access the TCP much like they would access the file system. The TCP may call on other operating system functions, for example, to manage data structures. The actual interface to the network is assumed to be controlled by a device driver module. The TCP does not call on the network device driver directly, but rather calls on the internet datagram protocol module which may in turn call on the device driver.

The mechanisms of TCP do not preclude implementation of the TCP in a front-end processor. However, in such an implementation, a host-to-front-end protocol must provide the functionality to support the type of TCP-user interface described in this document.

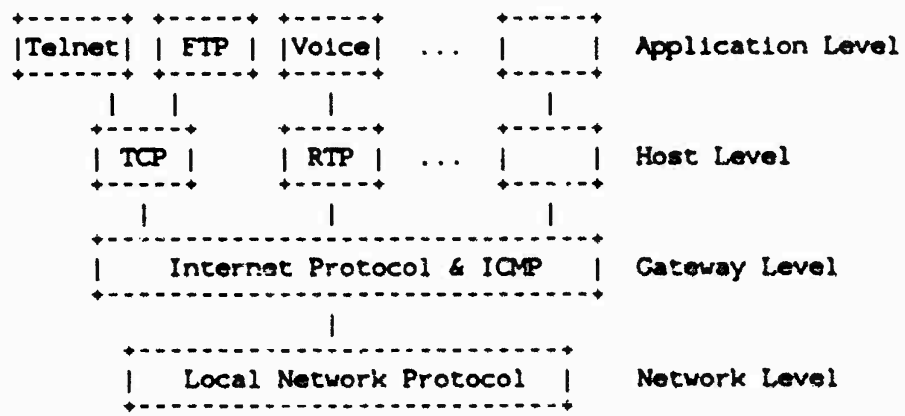
2.4. Interfaces

The TCP/user interface provides for calls made by the user on the TCP to OPEN or CLOSE a connection, to SEND or RECEIVE data, or to obtain STATUS about a connection. These calls are like other calls from user programs on the operating system, for example, the calls to open, read from, and close a file.

The TCP/internet interface provides calls to send and receive datagrams addressed to TCP modules in hosts anywhere in the internet system. These calls have parameters for passing the address, type of service, precedence, security, and other control information.

2.5. Relation to Other Protocols

The following diagram illustrates the place of the TCP in the protocol hierarchy:



Protocol Relationships

Figure 2.

It is expected that the TCP will be able to support higher level protocols efficiently. It should be easy to interface higher level protocols like the ARPANET Telnet or AUTODIN II THP to the TCP.

2.6. Reliable Communication

A stream of data sent on a TCP connection is delivered reliably and in order at the destination.

Transmission is made reliable via the use of sequence numbers and acknowledgments. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first octet of data in a segment is transmitted with that segment and is called the segment sequence number. Segments also carry an acknowledgment number which is the sequence number of the next expected data octet of transmissions in the reverse direction. When the TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted.

An acknowledgment by TCP does not guarantee that the data has been delivered to the end user, but only that the receiving TCP has taken the responsibility to do so.

To govern the flow of data between TCPs, a flow control mechanism is employed. The receiving TCP reports a "window" to the sending TCP. This window specifies the number of octets, starting with the acknowledgment number, that the receiving TCP is currently prepared to receive.

2.7. Connection Establishment and Clearing

To identify the separate data streams that a TCP may handle, the TCP provides a port identifier. Since port identifiers are selected independently by each TCP they might not be unique. To provide for unique addresses within each TCP, we concatenate an internet address identifying the TCP with a port identifier to create a socket which will be unique throughout all networks connected together.

A connection is fully specified by the pair of sockets at the ends. A local socket may participate in many connections to different foreign sockets. A connection can be used to carry data in both directions, that is, it is "full duplex".

TCPs are free to associate ports with processes however they choose. However, several basic concepts are necessary in any implementation. There must be well-known sockets which the TCP associates only with the "appropriate" processes by some means. We envision that processes may "own" ports, and that processes can initiate connections only on the ports they own. (Means for implementing ownership is a local issue, but we envision a Request Port user command, or a method of uniquely allocating a group of ports to a given process, e.g., by associating the high order bits of a port name with a given process.)

A connection is specified in the OPEN call by the local port and foreign socket arguments. In return, the TCP supplies a (short) local

connection name by which the user refers to the connection in subsequent calls. There are several things that must be remembered about a connection. To store this information we imagine that there is a data structure called a Transmission Control Block (TCB). One implementation strategy would have the local connection name be a pointer to the TCB for this connection. The OPEN call also specifies whether the connection establishment is to be actively pursued, or to be passively waited for.

A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case a foreign socket of all zeros is used to denote an unspecified socket. Unspecified foreign sockets are allowed only on passive OPENs.

A service process that wished to provide services for unknown other processes would issue a passive OPEN request with an unspecified foreign socket. Then a connection could be made with any process that requested a connection to this local socket. It would help if this local socket were known to be associated with this service.

Well-known sockets are a convenient mechanism for a priori associating a socket address with a standard service. For instance, the "Telnet-Server" process is permanently assigned to a particular socket, and other sockets are reserved for File Transfer, Remote Job Entry, Text Generator, Echoer, and Sink processes (the last three being for test purposes). A socket address might be reserved for access to a "Look-Up" service which would return the specific socket at which a newly created service would be provided. The concept of a well-known socket is part of the TCP specification, but the assignment of sockets to services is outside this specification. (See [4].)

Processes can issue passive OPENs and wait for matching active OPENs from other processes and be informed by the TCP when connections have been established. Two processes which issue active OPENs to each other at the same time will be correctly connected. This flexibility is critical for the support of distributed computing in which components act asynchronously with respect to each other.

There are two principal cases for matching the sockets in the local passive OPENs and an foreign active OPENs. In the first case, the local passive OPENs has fully specified the foreign socket. In this case, the match must be exact. In the second case, the local passive OPENs has left the foreign socket unspecified. In this case, any foreign socket is acceptable as long as the local sockets match. Other possibilities include partially restricted matches.

If there are several pending passive OPENs (recorded in TCBs) with the same local socket, an foreign active OPEN will be matched to a TCB with the specific foreign socket in the foreign active OPEN, if such a TCB exists, before selecting a TCB with an unspecified foreign socket.

The procedures to establish connections utilize the synchronize (SYN) control flag and involves an exchange of three messages. This exchange has been termed a three-way hand shake [3].

A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting TCB entry each created by a user OPEN command. The matching of local and foreign sockets determines when a connection has been initiated. The connection becomes "established" when sequence numbers have been synchronized in both directions.

The clearing of a connection also involves the exchange of segments, in this case carrying the FIN control flag.

2.8. Data Communication

The data that flows on a connection may be thought of as a stream of octets. The sending user indicates in each SEND call whether the data in that call (and any preceeding calls) should be immediately pushed through to the receiving user by the setting of the PUSH flag.

A sending TCP is allowed to collect data from the sending user and to send that data in segments at its own convenience, until the push function is signaled, then it must send all unsent data. When a receiving TCP sees the PUSH flag, it must not wait for more data from the sending TCP before passing the data to the receiving process.

There is no necessary relationship between push functions and segment boundaries. The data in any particular segment may be the result of a single SEND call, in whole or part, or of multiple SEND calls.

The purpose of push function and the PUSH flag is to push data through from the sending user to the receiving user. It does not provide a record service.

There is a coupling between the push function and the use of buffers of data that cross the TCP/user interface. Each time a PUSH flag is associated with data placed into the receiving user's buffer, the buffer is returned to the user for processing even if the buffer is not filled. If data arrives that fills the user's buffer before a PUSH is seen, the data is passed to the user in buffer size units.

TCP also provides a means to communicate to the receiver of data that at some point further along in the data stream than the receiver is

currently reading there is urgent data. TCP does not attempt to define what the user specifically does upon being notified of pending urgent data, but the general notion is that the receiving process will take action to process the urgent data quickly.

2.9. Precedence and Security

The TCP makes use of the internet protocol type of service field and security option to provide precedence and security on a per connection basis to TCP users. Not all TCP modules will necessarily function in a multilevel secure environment; some may be limited to unclassified use only, and others may operate at only one security level and compartment. Consequently, some TCP implementations and services to users may be limited to a subset of the multilevel secure case.

TCP modules which operate in a multilevel secure environment must properly mark outgoing segments with the security, compartment, and precedence. Such TCP modules must also provide to their users or higher level protocols such as Telnet or THP an interface to allow them to specify the desired security level, compartment, and precedence of connections.

2.10. Robustness Principle

TCP implementations will follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.

September 1981

Transmission Control Protocol

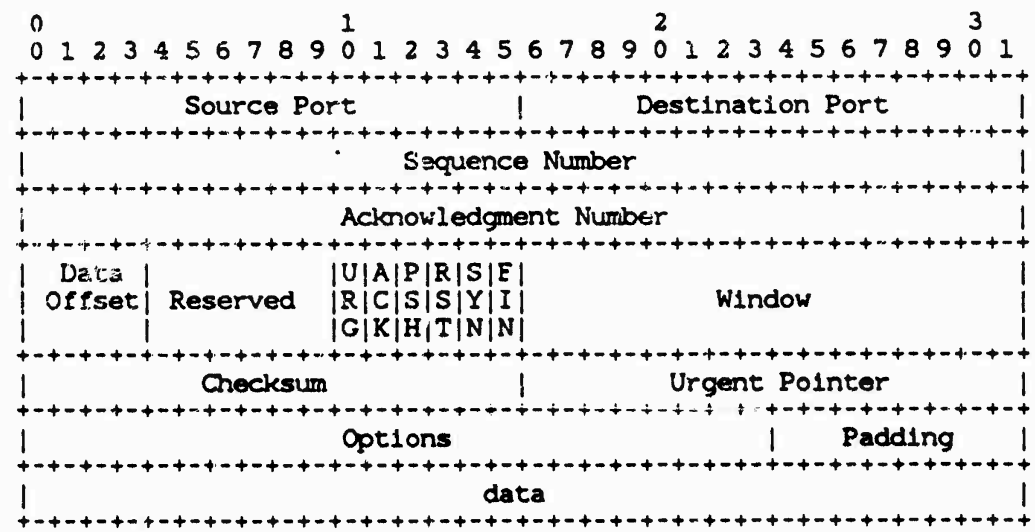
[Page 14]

3. FUNCTIONAL SPECIFICATION

3.1. Header Format

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses [2]. A TCP header follows the internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.

TCP Header Format



TCP Header Format

Note that one tick mark represents one bit position.

Figure 3.

Source Port: 16 bits

The source port number.

Destination Port: 16 bits

The destination port number.

Sequence Number: 32 bits

The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits

If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Data Offset: 4 bits

The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

Reserved: 6 bits

Reserved for future use. Must be zero.

Control Bits: 6 bits (from left to right):

URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

Window: 16 bits

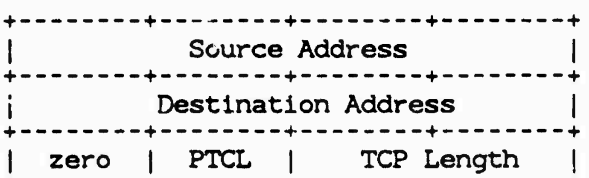
The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Checksum: 16 bits

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a 96 bit pseudo header conceptually

prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.



The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.

Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

A TCP must implement all options.

Currently defined options include (kind indicated in octal):

Kind	Length	Meaning
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.

Specific Option Definitions

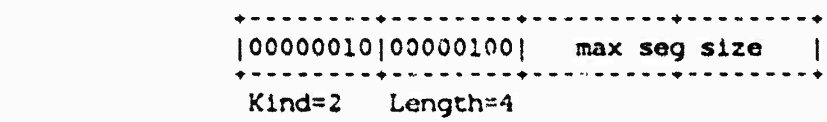
End of Option List

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Maximum Segment Size



Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

Padding: variable

The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

3.2. Terminology

Before we can discuss very much about the operation of the TCP we need to introduce some detailed terminology. The maintenance of a TCP connection requires the remembering of several variables. We conceive of these variables being stored in a connection record called a Transmission Control Block or TCB. Among the variables stored in the TCB are the local and remote socket numbers, the security and precedence of the connection, pointers to the user's send and receive buffers, pointers to the retransmit queue and to the current segment. In addition several variables relating to the send and receive sequence numbers are stored in the TCB.

Send Sequence Variables

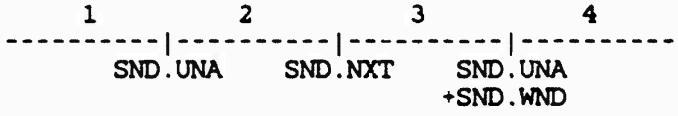
SND.UNA - send unacknowledged
SND.NXT - send next
SND.WND - send window
SND.UP - send urgent pointer
SND.WL1 - segment sequence number used for last window update
SND.WL2 - segment acknowledgment number used for last window update
ISS - initial send sequence number

Receive Sequence Variables

RCV.NXT - receive next
RCV.WND - receive window
RCV.UP - receive urgent pointer
IRS - initial receive sequence number

The following diagrams may help to relate some of these variables to the sequence space.

Send Sequence Space



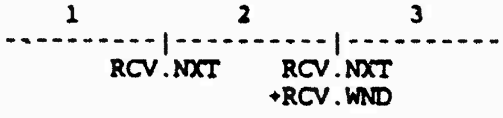
1 - old sequence numbers which have been acknowledged
2 - sequence numbers of unacknowledged data
3 - sequence numbers allowed for new data transmission
4 - future sequence numbers which are not yet allowed

Send Sequence Space

Figure 4.

The send window is the portion of the sequence space labeled 3 in figure 4.

Receive Sequence Space



1 - old sequence numbers which have been acknowledged
2 - sequence numbers allowed for new reception
3 - future sequence numbers which are not yet allowed

Receive Sequence Space

Figure 5.

The receive window is the portion of the sequence space labeled 2 in figure 5.

There are also some variables used frequently in the discussion that take their values from the fields of the current segment.

Current Segment Variables

SEG.SEQ - segment sequence number
SEG.ACK - segment acknowledgment number
SEG.LEN - segment length
SEG.WND - segment window
SEG.UP - segment urgent pointer
SEG.PRC - segment precedence value

A connection progresses through a series of states during its lifetime. The states are: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. CLOSED is fictional because it represents the state when there is no TCB, and therefore, no connection. Briefly the meanings of the states are:

LISTEN - represents waiting for a connection request from any remote TCP and port.

SYN-SENT - represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED - represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED - represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1 - represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2 - represents waiting for a connection termination request from the remote TCP.

CLOSE-WAIT - represents waiting for a connection termination request from the local user.

CLOSING - represents waiting for a connection termination request acknowledgment from the remote TCP.

LAST-ACK - represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

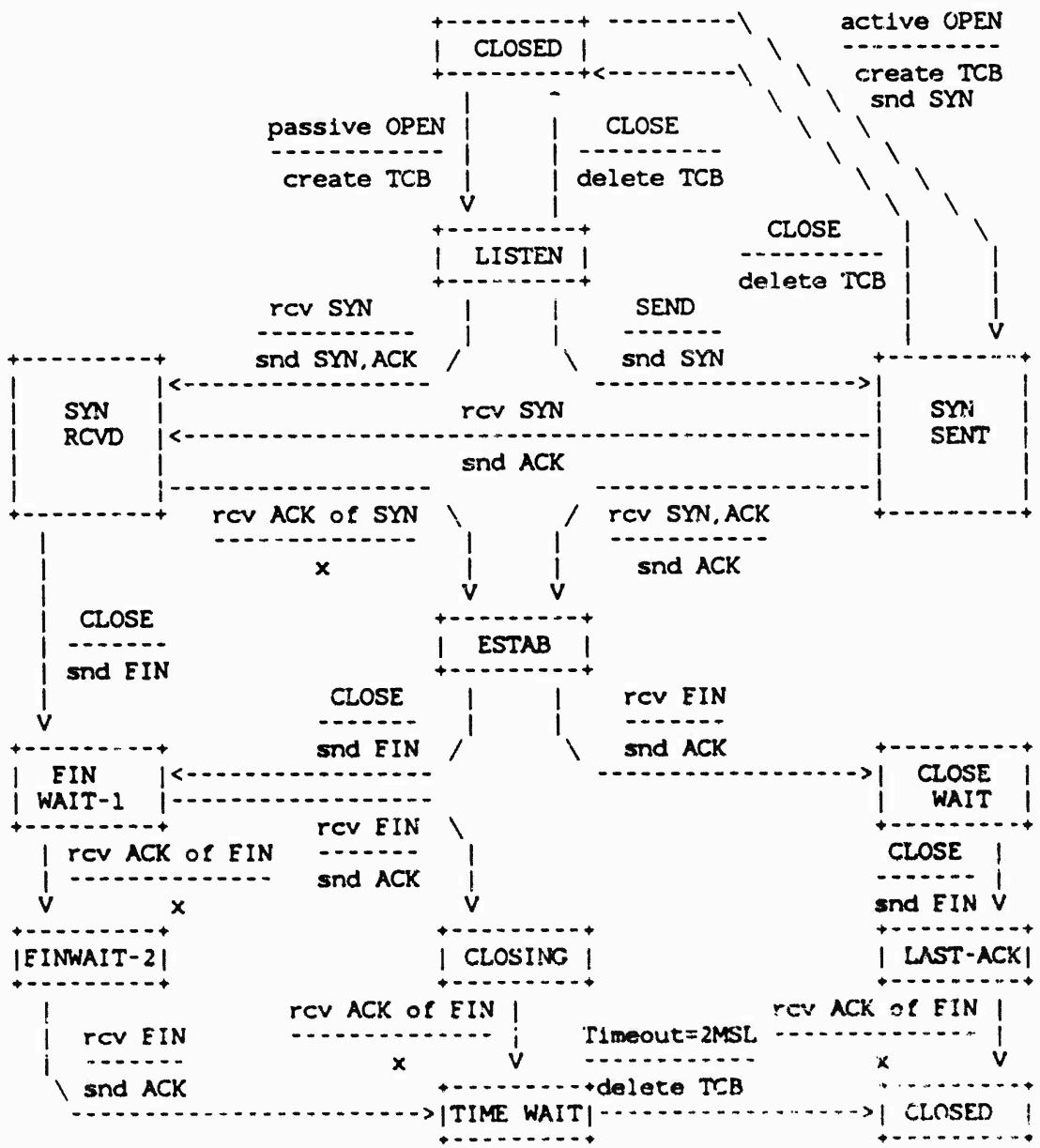
TIME-WAIT - represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request.

CLOSED - represents no connection state at all.

A TCP connection progresses from one state to another in response to events. The events are the user calls, OPEN, SEND, RECEIVE, CLOSE, ABORT, and STATUS; the incoming segments, particularly those containing the SYN, ACK, RST and FIN flags; and timeouts.

The state diagram in figure 6 illustrates only state changes, together with the causing events and resulting actions, but addresses neither error conditions nor actions which are not connected with state changes. In a later section, more detail is offered with respect to the reaction of the TCP to events.

NOTE BENE: this diagram is only a summary and must not be taken as the total specification.



TCP Connection State Diagram
Figure 6.

3.3. Sequence Numbers

A fundamental notion in the design is that every octet of data sent over a TCP connection has a sequence number. Since every octet is sequenced, each of them can be acknowledged. The acknowledgment mechanism employed is cumulative so that an acknowledgment of sequence number X indicates that all octets up to but not including X have been received. This mechanism allows for straight-forward duplicate detection in the presence of retransmission. Numbering of octets within a segment is that the first data octet immediately following the header is the lowest numbered, and the following octets are numbered consecutively.

It is essential to remember that the actual sequence number space is finite, though very large. This space ranges from 0 to 2**32 - 1. Since the space is finite, all arithmetic dealing with sequence numbers must be performed modulo 2**32. This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from 2**32 - 1 to 0 again. There are some subtleties to computer modulo arithmetic, so great care should be taken in programming the comparison of such values. The symbol "=<" means "less than or equal" (modulo 2**32).

The typical kinds of sequence number comparisons which the TCP must perform include:

(a) Determining that an acknowledgment refers to some sequence number sent but not yet acknowledged.

(b) Determining that all sequence numbers occupied by a segment have been acknowledged (e.g., to remove the segment from a retransmission queue).

(c) Determining that an incoming segment contains sequence numbers which are expected (i.e., that the segment "overlaps" the receive window).

In response to sending data the TCP will receive acknowledgments. The following comparisons are needed to process the acknowledgments.

SND.UNA = oldest unacknowledged sequence number

SND.NXT = next sequence number to be sent

SEG.ACK = acknowledgment from the receiving TCP (next sequence number expected by the receiving TCP)

SEG.SEQ = first sequence number of a segment

SEG.LEN = the number of octets occupied by the data in the segment (counting SYN and FIN)

SEG.SEQ+SEG.LEN-1 = last sequence number of a segment

A new acknowledgment (called an "acceptable ack"), is one for which the inequality below holds:

SND.UNA < SEG.ACK =< SND.NXT

A segment on the retransmission queue is fully acknowledged if the sum of its sequence number and length is less or equal than the acknowledgment value in the incoming segment.

When data is received the following comparisons are needed:

RCV.NXT = next sequence number expected on an incoming segments, and is the left or lower edge of the receive window

RCV.NXT+RCV.WND-1 = last sequence number expected on an incoming segment, and is the right or upper edge of the receive window

SEG.SEQ = first sequence number occupied by the incoming segment

SEG.SEQ+SEG.LEN-1 = last sequence number occupied by the incoming segment

A segment is judged to occupy a portion of valid receive sequence space if

RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

or

RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

The first part of this test checks to see if the beginning of the segment falls in the window, the second part of the test checks to see if the end of the segment falls in the window; if the segment passes either part of the test it contains data in the window.

Actually, it is a little more complicated than this. Due to zero windows and zero length segments, we have four cases for the acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

Note that when the receive window is zero no segments should be acceptable except ACK segments. Thus, it is be possible for a TCP to maintain a zero receive window while transmitting data and receiving ACKs. However, even when the receive window is zero, a TCP must process the RST and URG fields of all incoming segments.

We have taken advantage of the numbering scheme to protect certain control information as well. This is achieved by implicitly including some control flags in the sequence space so they can be retransmitted and acknowledged without confusion (i.e., one and only one copy of the control will be acted upon). Control information is not physically carried in the segment data space. Consequently, we must adopt rules for implicitly assigning sequence numbers to control. The SYN and FIN are the only controls requiring this protection, and these controls are used only at connection opening and closing. For sequence number purposes, the SYN is considered to occur before the first actual data octet of the segment in which it occurs, while the FIN is considered to occur after the last actual data octet in a segment in which it occurs. The segment length (SEG.LEN) includes both data and sequence space occupying controls. When a SYN is present then SEG.SEQ is the sequence number of the SYN.

Initial Sequence Number Selection

The protocol places no restriction on a particular connection being used over and over again. A connection is defined by a pair of sockets. New instances of a connection will be referred to as incarnations of the connection. The problem that arises from this is -- "how does the TCP identify duplicate segments from previous incarnations of the connection?" This problem becomes apparent if the connection is being opened and closed in quick succession, or if the connection breaks with loss of memory and is then reestablished.

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation. We want to assure this, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using. When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN. The generator is bound to a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4 microseconds. Thus, the ISN cycles approximately every 4.55 hours. Since we assume that segments will stay in the network no more than the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55 hours we can reasonably assume that ISN's will be unique.

For each connection there is a send sequence number and a receive sequence number. The initial send sequence number (ISS) is chosen by the data sending TCP, and the initial receive sequence number (IRS) is learned during the connection establishing procedure.

For a connection to be established or initialized, the two TCPs must synchronize on each other's initial sequence numbers. This is done in an exchange of connection establishing segments carrying a control bit called "SYN" (for synchronize) and the initial sequence numbers. As a shorthand, segments carrying the SYN bit are also called "SYNs". Hence, the solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISN's.

The synchronization requires each side to send it's own initial sequence number and to receive a confirmation of it in acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment.

1) A --> B SYN my sequence number is X
2) A <-- B ACK your sequence number is X
3) A <-- B SYN my sequence number is Y
4) A --> B ACK your sequence number is Y

Because steps 2 and 3 can be combined in a single message this is called the three way (or three message) handshake.

A three way handshake is necessary because sequence numbers are not tied to a global clock in the network, and TCPs may have different mechanisms for picking the ISN's. The receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not, unless it remembers the last sequence number used on the connection (which is not always possible), and so it must ask the sender to verify this SYN. The three way handshake and the advantages of a clock-driven scheme are discussed in [3].

Knowing When to Keep Quiet

To be sure that a TCP does not create a segment that carries a sequence number which may be duplicated by an old segment remaining in the network, the TCP must keep quiet for a maximum segment lifetime (MSL) before assigning any sequence numbers upon starting up or recovering from a crash in which memory of sequence numbers in use was lost. For this specification the MSL is taken to be 2 minutes. This is an engineering choice, and may be changed if experience indicates it is desirable to do so. Note that if a TCP is reinitialized in some sense, yet retains its memory of sequence numbers in use, then it need not wait at all; it must only be sure to use sequence numbers larger than those recently used.

The TCP Quiet Time Concept

This specification provides that hosts which "crash" without retaining any knowledge of the last sequence numbers transmitted on each active (i.e., not closed) connection shall delay emitting any TCP segments for at least the agreed Maximum Segment Lifetime (MSL) in the internet system of which the host is a part. In the paragraphs below, an explanation for this specification is given. TCP implementors may violate the "quiet time" restriction, but only at the risk of causing some old data to be accepted as new or new data rejected as old duplicated by some receivers in the internet system.

TCPs consume sequence number space each time a segment is formed and entered into the network output queue at a source host. The duplicate detection and sequencing algorithm in the TCP protocol relies on the unique binding of segment data to sequence space to the extent that sequence numbers will not cycle through all 2**32 values before the segment data bound to those sequence numbers has been delivered and acknowledged by the receiver and all duplicate copies of the segments have "drained" from the internet. Without such an assumption, two distinct TCP segments could conceivably be

assigned the same or overlapping sequence numbers, causing confusion at the receiver as to which data is new and which is old. Remember that each segment is bound to as many consecutive sequence numbers as there are octets of data in the segment.

Under normal conditions, TCPs keep track of the next sequence number to emit and the oldest awaiting acknowledgment so as to avoid mistakenly using a sequence number over before its first use has been acknowledged. This alone does not guarantee that old duplicate data is drained from the net, so the sequence space has been made very large to reduce the probability that a wandering duplicate will cause trouble upon arrival. At 2 megabits/sec. it takes 4.5 hours to use up 2**32 octets of sequence space. Since the maximum segment lifetime in the net is not likely to exceed a few tens of seconds, this is deemed ample protection for foreseeable nets, even if data rates escalate to 10's of megabits/sec. At 100 megabits/sec, the cycle time is 5.4 minutes which may be a little short, but still within reason.

The basic duplicate detection and sequencing algorithm in TCP can be defeated, however, if a source TCP does not have any memory of the sequence numbers it last used on a given connection. For example, if the TCP were to start all connections with sequence number 0, then upon crashing and restarting, a TCP might re-form an earlier connection (possibly after half-open connection resolution) and emit packets with sequence numbers identical to or overlapping with packets still in the network which were emitted on an earlier incarnation of the same connection. In the absence of knowledge about the sequence numbers used on a particular connection, the TCP specification recommends that the source delay for MSL seconds before emitting segments on the connection, to allow time for segments from the earlier connection incarnation to drain from the system.

Even hosts which can remember the time of day and used it to select initial sequence number values are not immune from this problem (i.e., even if time of day is used to select an initial sequence number for each new connection incarnation).

Suppose, for example, that a connection is opened starting with sequence number S. Suppose that this connection is not used much and that eventually the initial sequence number function (ISN(t)) takes on a value equal to the sequence number, say S1, of the last segment sent by this TCP on a particular connection. Now suppose, at this instant, the host crashes, recovers, and establishes a new incarnation of the connection. The initial sequence number chosen is S1 = ISN(t) -- last used sequence number on old incarnation of connection! If the recovery occurs quickly enough, any old

duplicates in the net bearing sequence numbers in the neighborhood of S1 may arrive and be treated as new packets by the receiver of the new incarnation of the connection.

The problem is that the recovering host may not know for how long it crashed nor does it know whether there are still old duplicates in the system from earlier connection incarnations.

One way to deal with this problem is to deliberately delay emitting segments for one MSL after recovery from a crash- this is the "quite time" specification. Hosts which prefer to avoid waiting are willing to risk possible confusion of old and new packets at a given destination may choose not to wait for the "quite time". Implementors may provide TCP users with the ability to select on a connection by connection basis whether to wait after a crash, or may informally implement the "quite time" for all connections. Obviously, even where a user selects to "wait," this is not necessary after the host has been "up" for at least MSL seconds.

To summarize: every segment emitted occupies one or more sequence numbers in the sequence space, the numbers occupied by a segment are "busy" or "in use" until MSL seconds have passed, upon crashing a block of space-time is occupied by the octets of the last emitted segment, if a new connection is started too soon and uses any of the sequence numbers in the space-time footprint of the last segment of the previous connection incarnation, there is a potential sequence number overlap area which could cause confusion at the receiver.

3.4. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the

implementation of a trade-off between memory and messages to provide information for this checking.

The simplest three-way handshake is shown in figure 7 below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Ellipsis (...) indicates a segment which is still in the network (delayed). An "XXX" indicates a segment which is lost or rejected. Comments appear in parentheses. TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

```
   TCP A                                                TCP B

1. CLOSED                                               LISTEN

2. SYN-SENT     --> <SEQ=100><CTL=SYN>               --> SYN-RECEIVED

3. ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED

4. ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>       --> ESTABLISHED

5. ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> --> ESTABLISHED
```

Basic 3-Way Handshake for Connection Synchronization

Figure 7.

In line 2 of figure 7, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!).

Simultaneous initiation is only slightly more complex, as is shown in figure 8. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED.

	TCP A			TCP B
1.	CLOSED			CLOSED
2.	SYN-SENT	-->	<SEQ=100><CTL=SYN>	...
3.	SYN-RECEIVED	<--	<SEQ=300><CTL=SYN>	<-- SYN-SENT
4.		...	<SEQ=100><CTL=SYN>	--> SYN-RECEIVED
5.	SYN-RECEIVED	-->	<SEQ=100><ACK=301><CTL=SYN,ACK>	...
6.	ESTABLISHED	<--	<SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
7.		...	<SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED

Simultaneous Connection Synchronization

Figure 8.

The principle reason for the three-way handshake is to prevent old duplicate connection initiations from causing confusion. To deal with this, a special control message, reset, has been devised. If the receiving TCP is in a non-synchronized state (i.e., SYN-SENT, SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset. If the TCP is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it aborts the connection and informs its user. We discuss this latter case under "half-open" connections below.

	TCP A			TCP B
1.	CLOSED			LISTEN
2.	SYN-SENT	--> <SEQ=100><CTL=SYN>	...	
3.	(duplicate)	... <SEQ=90><CTL=SYN>	-->	SYN-RECEIVED
4.	SYN-SENT	<-- <SEQ=300><ACK=91><CTL=SYN,ACK>	<--	SYN-RECEIVED
5.	SYN-SENT	--> <SEQ=91><CTL=RST>	-->	LISTEN
6.		... <SEQ=100><CTL=SYN>	-->	SYN-RECEIVED
7.	SYN-SENT	<-- <SEQ=400><ACK=101><CTL=SYN,ACK>	<--	SYN-RECEIVED
8.	ESTABLISHED	--> <SEQ=101><ACK=401><CTL=ACK>	-->	ESTABLISHED

Recovery from Old Duplicate SYN

Figure 9.

As a simple example of recovery from old duplicates, consider figure 9. At line 3, an old duplicate SYN arrives at TCP B. TCP B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP B, on receiving the RST, returns to the LISTEN state. When the original SYN (pun intended) finally arrives at line 6, the synchronization proceeds normally. If the SYN at line 6 had arrived before the RST, a more complex exchange might have occurred with RST's sent in both directions.

Half-Open Connections and Other Anomalies

An established connection is said to be "half-open" if one of the TCPs has closed or aborted the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized owing to a crash that resulted in loss of memory. Such connections will automatically become reset if an attempt is made to send data in either direction. However, half-open connections are expected to be unusual, and the recovery procedure is mildly involved.

If at site A the connection no longer exists, then an attempt by the

user at site B to send any data on it will result in the site B TCP receiving a reset control message. Such a message indicates to the site B TCP that something is wrong, and it is expected to abort the connection.

Assume that two user processes A and B are communicating with one another when a crash occurs causing loss of memory to A's TCP. Depending on the operating system supporting A's TCP, it is likely that some error recovery mechanism exists. When the TCP is up again, A is likely to start again from the beginning or from a recovery point. As a result, A will probably try to OPEN the connection again or try to SEND on the connection it believes open. In the latter case, it receives the error message "connection not open" from the local (A's) TCP. In an attempt to establish the connection, A's TCP will send a segment containing SYN. This scenario leads to the example shown in figure 10. After TCP A crashes, the user attempts to re-open the connection. TCP B, in the meantime, thinks the connection is open.

```
    TCP A                                                TCP B

1.  (CRASH)                                   (send 300,receive 100)

2.  CLOSED                                              ESTABLISHED

3.  SYN-SENT --> <SEQ=400><CTL=SYN>                --> (??)

4.  (!!)     <-- <SEQ=300><ACK=100><CTL=ACK>       <-- ESTABLISHED

5.  SYN-SENT --> <SEQ=100><CTL=RST>                --> (Abort!!)

6.  SYN-SENT                                            CLOSED

7.  SYN-SENT --> <SEQ=400><CTL=SYN>                -->
```

Half-Open Connection Discovery

Figure 10.

When the SYN arrives at line 3, TCP B, being in a synchronized state, and the incoming segment outside the window, responds with an acknowledgment indicating what sequence it next expects to hear (ACK 100). TCP A sees that this segment does not acknowledge anything it sent and, being unsynchronized, sends a reset (RST) because it has detected a half-open connection. TCP B aborts at line 5. TCP A will

continue to try to establish the connection; the problem is now reduced to the basic 3-way handshake of figure 7.

An interesting alternative case occurs when TCP A crashes and TCP B tries to send data on what it thinks is a synchronized connection. This is illustrated in figure 11. In this case, the data arriving at TCP A from TCP B (line 2) is unacceptable because no such connection exists, so TCP A sends a RST. The RST is acceptable so TCP B processes it and aborts the connection.

	TCP A		TCP B
1.	(CRASH)		(send 300, receive 100)
2.	(??)	<-- <SEQ=300><ACK=100><DATA=10><CTL=ACK>	<-- ESTABLISHED
3.		--> <SEQ=100><CTL=RST>	--> (ABORT!!)

Active Side Causes Half-Open Connection Discovery

Figure 11.

In figure 12, we find the two TCPs A and B with passive connections waiting for SYN. An old duplicate arriving at TCP B (line 2) stirs B into action. A SYN-ACK is returned (line 3) and causes TCP A to generate a RST (the ACK in line 3 is not acceptable). TCP B accepts the reset and returns to its passive LISTEN state.

	TCP A			TCP B
1.	LISTEN			LISTEN
2.		... <SEQ=Z><CTL=SYN>	-->	SYN-RECEIVED
3.	(??)	<-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK>	<--	SYN-RECEIVED
4.		--> <SEQ=Z+1><CTL=RST>	-->	(return to LISTEN!)
5.	LISTEN			LISTEN

Old Duplicate SYN Initiates a Reset on two Passive Sockets

Figure 12.

A variety of other cases are possible, all of which are accounted for by the following rules for RST generation and processing.

Reset Generation

As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case.

There are three groups of states:

1. If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means.

   If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the CLOSED state.

2. If the connection is in any non-synchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), and the incoming segment acknowledges something not yet sent (the segment carries an unacceptable ACK), or if an incoming segment has a security level or compartment which does not exactly match the level and compartment requested for the connection, a reset is sent.

   If our SYN has not been acknowledged and the precedence level of the incoming segment is higher than the precedence level requested then either raise the local precedence level (if allowed by the user and the system) or send a reset; or if the precedence level of the incoming segment is lower than the precedence level requested then continue as if the precedence matched exactly (if the remote TCP cannot raise the precedence level to match ours this will be detected in the next segment it sends, and the connection will be terminated then). If our SYN has been acknowledged (perhaps in this incoming segment) the precedence level of the incoming segment must match the local precedence level exactly, if it does not a reset must be sent.

   If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the same state.

3. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any unacceptable segment (out of window sequence number or unacceptible acknowledgment number) must elicit only an empty acknowledgment segment containing the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received, and the connection remains in the same state.

If an incoming segment has a security level, or compartment, or precedence which does not exactly match the level, and compartment, and precedence requested for the connection,a reset is sent and connection goes to the CLOSED state. The reset takes its sequence number from the ACK field of the incoming segment.

Reset Processing

In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields. A reset is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN.

The receiver of a RST first validates it, then changes state. If the receiver was in the LISTEN state, it ignores it. If the receiver was in SYN-RECEIVED state and had previously been in the LISTEN state, then the receiver returns to the LISTEN state, otherwise the receiver aborts the connection and goes to the CLOSED state. If the receiver was in any other state, it aborts the connection and advises the user and goes to the CLOSED state.

3.5. Closing a Connection

CLOSE is an operation meaning "I have no more data to send." The notion of closing a full-duplex connection is subject to ambiguous interpretation, of course, since it may not be obvious how to treat the receiving side of the connection. We have chosen to treat CLOSE in a simplex fashion. The user who CLOSEs may continue to RECEIVE until he is told that the other side has CLOSED also. Thus, a program could initiate several SENDs followed by a CLOSE, and then continue to RECEIVE until signaled that a RECEIVE failed because the other side has CLOSED. We assume that the TCP will signal a user, even if no RECEIVEs are outstanding, that the other side has closed, so the user can terminate his side gracefully. A TCP will reliably deliver all buffers SENT before the connection was CLOSED so a user who expects no data in return need only wait to hear the connection was CLOSED successfully to know that all his data was received at the destination TCP. Users must keep reading connections they close for sending until the TCP says no more data.

There are essentially three cases:

1) The user initiates by telling the TCP to CLOSE the connection

2) The remote TCP initiates by sending a FIN control signal

3) Both users CLOSE simultaneously

Case 1: Local user initiates the close

In this case, a FIN segment can be constructed and placed on the outgoing segment queue. No further SENDs from the user will be accepted by the TCP, and it enters the FIN-WAIT-1 state. RECEIVEs are allowed in this state. All segments preceding and including FIN will be retransmitted until acknowledged. When the other TCP has both acknowledged the FIN and sent a FIN of its own, the first TCP can ACK this FIN. Note that a TCP receiving a FIN will ACK but not send its own FIN until its user has CLOSED the connection also.

Case 2: TCP receives a FIN from the network

If an unsolicited FIN arrives from the network, the receiving TCP can ACK it and tell the user that the connection is closing. The user will respond with a CLOSE, upon which the TCP can send a FIN to the other TCP after sending any remaining data. The TCP then waits until its own FIN is acknowledged whereupon it deletes the connection. If an ACK is not forthcoming, after the user timeout the connection is aborted and the user is told.

Case 3: both users close simultaneously

A simultaneous CLOSE by users at both ends of a connection causes FIN segments to be exchanged. When all segments preceding the FINs have been processed and acknowledged, each TCP can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

```
    TCP A                                                    TCP B

1.  ESTABLISHED                                              ESTABLISHED

2.  (Close)
    FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>      --> CLOSE-WAIT

3.  FIN-WAIT-2  <-- <SEQ=300><ACK=101><CTL=ACK>          <-- CLOSE-WAIT

4.                                                           (Close)
    TIME-WAIT   <-- <SEQ=300><ACK=101><CTL=FIN,ACK>      <-- LAST-ACK

5.  TIME-WAIT   --> <SEQ=101><ACK=301><CTL=ACK>          --> CLOSED

6.  (2 MSL)
    CLOSED
```

Normal Close Sequence

Figure 13.

```
    TCP A                                                    TCP B

1.  ESTABLISHED                                              ESTABLISHED

2.  (Close)                                                  (Close)
    FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>      ... FIN-WAIT-1
                <-- <SEQ=300><ACK=100><CTL=FIN,ACK>      <--
                ... <SEQ=100><ACK=300><CTL=FIN,ACK>      -->

3.  CLOSING     --> <SEQ=101><ACK=301><CTL=ACK>          ... CLOSING
                <-- <SEQ=301><ACK=101><CTL=ACK>          <--
                ... <SEQ=101><ACK=301><CTL=ACK>          -->

4.  TIME-WAIT                                                TIME-WAIT
    (2 MSL)                                                  (2 MSL)
    CLOSED                                                   CLOSED
```

Simultaneous Close Sequence

Figure 14.

3.6. Precedence and Security

The intent is that connection be allowed only between ports operating with exactly the same security and compartment values and at the higher of the precedence level requested by the two ports.

The precedence and security parameters used in TCP are exactly those defined in the Internet Protocol (IP) [2]. Throughout this TCP specification the term "security/compartment" is intended to indicate the security parameters used in IP including security, compartment, user group, and handling restriction.

A connection attempt with mismatched security/compartment values or a lower precedence value must be rejected by sending a reset. Rejecting a connection due to too low a precedence only occurs after an acknowledgment of the SYN has been received.

Note that TCP modules which operate only at the default value of precedence will still have to check the precedence of incoming segments and possibly raise the precedence level they use on the connection.

The security paramaters may be used even in a non-secure environment (the values would indicate unclassified data), thus hosts in non-secure environments must be prepared to receive the security parameters, though they need not send them.

3.7. Data Communication

Once the connection is established data is communicated by the exchange of segments. Because segments may be lost due to errors (checksum test failure), or network congestion, TCP uses retransmission (after a timeout) to ensure delivery of every segment. Duplicate segments may arrive due to network or TCP retransmission. As discussed in the section on sequence numbers the TCP performs certain tests on the sequence and acknowledgment numbers in the segments to verify their acceptability.

The sender of data keeps track of the next sequence number to use in the variable SND.NXT. The receiver of data keeps track of the next sequence number to expect in the variable RCV.NXT. The sender of data keeps track of the oldest unacknowledged sequence number in the variable SND.UNA. If the data flow is momentarily idle and all data sent has been acknowledged then the three variables will be equal.

When the sender creates a segment and transmits it the sender advances SND.NXT. When the receiver accepts a segment it advances RCV.NXT and sends an acknowledgment. When the data sender receives an

acknowledgment it advances SND.UNA. The extent to which the values of these variables differ is a measure of the delay in the communication. The amount by which the variables are advanced is the length of the data in the segment. Note that once in the ESTABLISHED state all segments must carry current acknowledgment information.

The CLOSE user call implies a push function, as does the FIN control flag in an incoming segment.

Retransmission Timeout

Because of the variability of the networks that compose an internetwork system and the wide range of uses of TCP connections the retransmission timeout must be dynamically determined. One procedure for determining a retransmission time out is given here as an illustration.

An Example Retransmission Timeout Procedure

Measure the elapsed time between sending a data octet with a particular sequence number and receiving an acknowledgment that covers that sequence number (segments sent do not have to match segments received). This measured elapsed time is the Round Trip Time (RTT). Next compute a Smoothed Round Trip Time (SRTT) as:

SRTT = (ALPHA * SRTT) + ((1-ALPHA) * RTT)

and based on this, compute the retransmission timeout (RTO) as:

RTO = min[UBOUND,max[LBOUND,(BETA*SRTT)]]

where UBOUND is an upper bound on the timeout (e.g., 1 minute), LBOUND is a lower bound on the timeout (e.g., 1 second), ALPHA is a smoothing factor (e.g., .8 to .9), and BETA is a delay variance factor (e.g., 1.3 to 2.0).

The Communication of Urgent Information

The objective of the TCP urgent mechanism is to allow the sending user to stimulate the receiving user to accept some urgent data and to permit the receiving TCP to indicate to the receiving user when all the currently known urgent data has been received by the user.

This mechanism permits a point in the data stream to be designated as the end of urgent information. Whenever this point is in advance of the receive sequence number (RCV.NXT) at the receiving TCP, that TCP must tell the user to go into "urgent mode"; when the receive sequence number catches up to the urgent pointer, the TCP must tell user to go

into "normal mode". If the urgent pointer is updated while the user is in "urgent mode", the update will be invisible to the user.

The method employs a urgent field which is carried in all segments transmitted. The URG control flag indicates that the urgent field is meaningful and must be added to the segment sequence number to yield the urgent pointer. The absence of this flag indicates that there is no urgent data outstanding.

To send an urgent indication the user must also send at least one data octet. If the sending user also indicates a push, timely delivery of the urgent information to the destination process is enhanced.

Managing the Window

The window sent in each segment indicates the range of sequence numbers the sender of the window (the data receiver) is currently prepared to accept. There is an assumption that this is related to the currently available data buffer space available for this connection.

Indicating a large window encourages transmissions. If more data arrives than can be accepted, it will be discarded. This will result in excessive retransmissions, adding unnecessarily to the load on the network and the TCPs. Indicating a small window may restrict the transmission of data to the point of introducing a round trip delay between each new segment transmitted.

The mechanisms provided allow a TCP to advertise a large window and to subsequently advertise a much smaller window without having accepted that much data. This, so called "shrinking the window," is strongly discouraged. The robustness principle dictates that TCPs will not shrink the window themselves, but will be prepared for such behavior on the part of other TCPs.

The sending TCP must be prepared to accept from the user and send at least one octet of new data even if the send window is zero. The sending TCP must regularly retransmit to the receiving TCP even when the window is zero. Two minutes is recommended for the retransmission interval when the window is zero. This retransmission is essential to guarantee that when either TCP has a zero window the re-opening of the window will be reliably reported to the other.

When the receiving TCP has a zero window and a segment arrives it must still send an acknowledgment showing its next expected sequence number and current window (zero).

The sending TCP packages the data to be transmitted into segments

which fit the current window, and may repackage segments on the retransmission queue. Such repackaging is not required, but may be helpful.

In a connection with a one-way data flow, the window information will be carried in acknowledgment segments that all have the same sequence number so there will be no way to reorder them if they arrive out of order. This is not a serious problem, but it will allow the window information to be on occasion temporarily based on old reports from the data receiver. A refinement to avoid this problem is to act on the window information from segments that carry the highest acknowledgment number (that is segments with acknowledgment number equal or greater than the highest previously received).

The window management procedure has significant influence on the communication performance. The following comments are suggestions to implementers.

Window Management Suggestions

Allocating a very small window causes data to be transmitted in many small segments when better performance is achieved using fewer large segments.

One suggestion for avoiding small windows is for the receiver to defer updating a window until the additional allocation is at least X percent of the maximum allocation possible for the connection (where X might be 20 to 40).

Another suggestion is for the sender to avoid sending small segments by waiting until the window is large enough before sending data. If the the user signals a push function then the data must be sent even if it is a small segment.

Note that the acknowledgments should not be delayed or unnecessary retransmissions will result. One strategy would be to send an acknowledgment when a small segment arrives (with out updating the window information), and then to send another acknowledgment with new window information when the window is larger.

The segment sent to probe a zero window may also begin a break up of transmitted data into smaller and smaller segments. If a segment containing a single data octet sent to probe a zero window is accepted, it consumes one octet of the window now available. If the sending TCP simply sends as much as it can whenever the window is non zero, the transmitted data will be broken into alternating big and small segments. As time goes on, occasional pauses in the receiver making window allocation available will

result in breaking the big segments into a small and not quite so big pair. And after a while the data transmission will be in mostly small segments.

The suggestion here is that the TCP implementations need to actively attempt to combine small window allocations into larger windows, since the mechanisms for managing the window tend to lead to many small windows in the simplest minded implementations.

3.8. Interfaces

There are of course two interfaces of concern: the user/TCP interface and the TCP/lower-level interface. We have a fairly elaborate model of the user/TCP interface, but the interface to the lower level protocol module is left unspecified here, since it will be specified in detail by the specification of the lowel level protocol. For the case that the lower level is IP we note some of the parameter values that TCPs might use.

User/TCP Interface

The following functional description of user commands to the TCP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different TCP implementations may have different user interfaces. However, all TCPs must provide a certain minimum set of services to guarantee that all TCP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all TCP implementations.

TCP User Commands

The following sections functionally characterize a USER/TCP interface. The notation used is similar to most procedure or function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs).

The user commands described below specify the basic functions the TCP must perform to support interprocess communication. Individual implementations must define their own exact format, and may provide combinations or subsets of the basic functions in single calls. In particular, some implementations may wish to automatically OPEN a connection on the first SEND or RECEIVE issued by the user for a given connection.

In providing interprocess communication facilities, the TCP must not only accept commands, but must also return information to the processes it serves. The latter consists of:

(a) general information about a connection (e.g., interrupts, remote close, binding of unspecified foreign socket).

(b) replies to specific user commands indicating success or various types of failure.

Open

Format: OPEN (local port, foreign socket, active/passive [, timeout] [, precedence] [, security/compartment] [, options]) -> local connection name

We assume that the local TCP is aware of the identity of the processes it serves and will check the authority of the process to use the connection specified. Depending upon the implementation of the TCP, the local network and TCP identifiers for the source address will either be supplied by the TCP or the lower level protocol (e.g., IP). These considerations are the result of concern about security, to the extent that no TCP be able to masquerade as another one, and so on. Similarly, no process can masquerade as another without the collusion of the TCP.

If the active/passive flag is set to passive, then this is a call to LISTEN for an incoming connection. A passive open may have either a fully specified foreign socket to wait for a particular connection or an unspecified foreign socket to wait for any call. A fully specified passive call can be made active by the subsequent execution of a SEND.

A transmission control block (TCB) is created and partially filled in with data from the OPEN command parameters.

On an active OPEN command, the TCP will begin the procedure to synchronize (i.e., establish) the connection at once.

The timeout, if present, permits the caller to set up a timeout for all data submitted to TCP. If data is not successfully delivered to the destination within the timeout period, the TCP will abort the connection. The present global default is five minutes.

The TCP or some component of the operating system will verify the users authority to open a connection with the specified

precedence or security/compartment. The absence of precedence or security/compartment specification in the OPEN call indicates the default values must be used.

TCP will accept incoming requests as matching only if the security/compartment information is exactly the same and only if the precedence is equal to or higher than the precedence requested in the OPEN call.

The precedence for the connection is the higher of the values requested in the OPEN call and received from the incoming request, and fixed at that value for the life of the connection.Implementers may want to give the user control of this precedence negotiation. For example, the user might be allowed to specify that the precedence must be exactly matched, or that any attempt to raise the precedence be confirmed by the user.

A local connection name will be returned to the user by the TCP. The local connection name can then be used as a short hand term for the connection defined by the <local socket, foreign socket> pair.

Send

Format: SEND (local connection name, buffer address, byte count, PUSH flag, URGENT flag [,timeout])

This call causes the data contained in the indicated user buffer to be sent on the indicated connection. If the connection has not been opened, the SEND is considered an error. Some implementations may allow users to SEND first; in which case, an automatic OPEN would be done. If the calling process is not authorized to use this connection, an error is returned.

If the PUSH flag is set, the data must be transmitted promptly to the receiver, and the PUSH bit will be set in the last TCP segment created from the buffer. If the PUSH flag is not set, the data may be combined with data from subsequent SENDs for transmission efficiency.

If the URGENT flag is set, segments sent to the destination TCP will have the urgent pointer set. The receiving TCP will signal the urgent condition to the receiving process if the urgent pointer indicates that data preceding the urgent pointer has not been consumed by the receiving process. The purpose of urgent is to stimulate the receiver to process the urgent data and to indicate to the receiver when all the currently known urgent

data has been received. The number of times the sending user's TCP signals urgent will not necessarily be equal to the number of times the receiving user will be notified of the presence of urgent data.

If no foreign socket was specified in the OPEN, but the connection is established (e.g., because a LISTENing connection has become specific due to a foreign segment arriving for the local socket), then the designated buffer is sent to the implied foreign socket. Users who make use of OPEN with an unspecified foreign socket can make use of SEND without ever explicitly knowing the foreign socket address.

However, if a SEND is attempted before the foreign socket becomes specified, an error will be returned. Users can use the STATUS call to determine the status of the connection. In some implementations the TCP may notify the user when an unspecified socket is bound.

If a timeout is specified, the current user timeout for this connection is changed to the new one.

In the simplest implementation, SEND would not return control to the sending process until either the transmission was complete or the timeout had been exceeded. However, this simple method is both subject to deadlocks (for example, both sides of the connection might try to do SENDs before doing any RECEIVEs) and offers poor performance, so it is not recommended. A more sophisticated implementation would return immediately to allow the process to run concurrently with network I/O, and, furthermore, to allow multiple SENDs to be in progress. Multiple SENDs are served in first come, first served order, so the TCP will queue those it cannot service immediately.

We have implicitly assumed an asynchronous user interface in which a SEND later elicits some kind of SIGNAL or pseudo-interrupt from the serving TCP. An alternative is to return a response immediately. For instance, SENDs might return immediate local acknowledgment, even if the segment sent had not been acknowledged by the distant TCP. We could optimistically assume eventual success. If we are wrong, the connection will close anyway due to the timeout. In implementations of this kind (synchronous), there will still be some asynchronous signals, but these will deal with the connection itself, and not with specific segments or buffers.

In order for the process to distinguish among error or success indications for different SENDs, it might be appropriate for the

buffer address to be returned along with the coded response to the SEND request. TCP-to-user signals are discussed below, indicating the information which should be returned to the calling process.

Receive

Format: RECEIVE (local connection name, buffer address, byte count) -> byte count, urgent flag, push flag

This command allocates a receiving buffer associated with the specified connection. If no OPEN precedes this command or the calling process is not authorized to use this connection, an error is returned.

In the simplest implementation, control would not return to the calling program until either the buffer was filled, or some error occurred, but this scheme is highly subject to deadlocks. A more sophisticated implementation would permit several RECEIVEs to be outstanding at once. These would be filled as segments arrive. This strategy permits increased throughput at the cost of a more elaborate scheme (possibly asynchronous) to notify the calling program that a PUSH has been seen or a buffer filled.

If enough data arrive to fill the buffer before a PUSH is seen, the PUSH flag will not be set in the response to the RECEIVE. The buffer will be filled with as much data as it can hold. If a PUSH is seen before the buffer is filled the buffer will be returned partially filled and PUSH indicated.

If there is urgent data the user will have been informed as soon as it arrived via a TCP-to-user signal. The receiving user should thus be in "urgent mode". If the URGENT flag is on, additional urgent data remains. If the URGENT flag is off, this call to RECEIVE has returned all the urgent data, and the user may now leave "urgent mode". Note that data following the urgent pointer (non-urgent data) cannot be delivered to the user in the same buffer with preceeding urgent data unless the boundary is clearly marked for the user.

To distinguish among several outstanding RECEIVEs and to take care of the case that a buffer is not completely filled, the return code is accompanied by both a buffer pointer and a byte count indicating the actual length of the data received.

Alternative implementations of RECEIVE might have the TCP

allocate buffer storage, or the TCP might share a ring buffer with the user.

Close

Format: CLOSE (local connection name)

This command causes the connection specified to be closed. If the connection is not open or the calling process is not authorized to use this connection, an error is returned. Closing connections is intended to be a graceful operation in the sense that outstanding SENDs will be transmitted (and retransmitted), as flow control permits, until all have been serviced. Thus, it should be acceptable to make several SEND calls, followed by a CLOSE, and expect all the data to be sent to the destination. It should also be clear that users should continue to RECEIVE on CLOSING connections, since the other side may be trying to transmit the last of its data. Thus, CLOSE means "I have no more to send" but does not mean "I will not receive any more." It may happen (if the user level protocol is not well thought out) that the closing side is unable to get rid of all its data before timing out. In this event, CLOSE turns into ABORT, and the closing TCP gives up.

The user may CLOSE the connection at any time on his own initiative, or in response to various prompts from the TCP (e.g., remote close executed, transmission timeout exceeded, destination inaccessible).

Because closing a connection requires communication with the foreign TCP, connections may remain in the closing state for a short time. Attempts to reopen the connection before the TCP replies to the CLOSE command will result in error responses.

Close also implies push function.

Status

Format: STATUS (local connection name) -> status data

This is an implementation dependent user command and could be excluded without adverse effect. Information returned would typically come from the TCB associated with the connection.

This command returns a data block containing the following information:

local socket,

foreign socket,
local connection name,
receive window,
send window,
connection state,
number of buffers awaiting acknowledgment,
number of buffers pending receipt,
urgent state,
precedence,
security/compartment,
and transmission timeout.

Depending on the state of the connection, or on the implementation itself, some of this information may not be available or meaningful. If the calling process is not authorized to use this connection, an error is returned. This prevents unauthorized processes from gaining information about a connection.

Abort

Format: ABORT (local connection name)

This command causes all pending SENDs and RECEIVES to be aborted, the TCB to be removed, and a special RESET message to be sent to the TCP on the other side of the connection. Depending on the implementation, users may receive abort indications for each outstanding SEND or RECEIVE, or may simply receive an ABORT-acknowledgment.

TCP-to-User Messages

It is assumed that the operating system environment provides a means for the TCP to asynchronously signal the user program. When the TCP does signal a user program, certain information is passed to the user. Often in the specification the information will be an error message. In other cases there will be information relating to the completion of processing a SEND or RECEIVE or other user call.

The following information is provided:

Local Connection Name	Always
Response String	Always
Buffer Address	Send & Receive
Byte count (counts bytes received)	Receive
Push flag	Receive
Urgent flag	Receive

TCP/Lower-Level Interface

The TCP calls on a lower level protocol module to actually send and receive information over a network. One case is that of the ARPA internetwork system where the lower level module is the Internet Protocol (IP) [2].

If the lower level protocol is IP it provides arguments for a type of service and for a time to live. TCP uses the following settings for these parameters:

Type of Service = Precedence: routine, Delay: normal, Throughput: normal, Reliability: normal; or 00000000.

Time to Live = one minute, or 00111100.

Note that the assumed maximum segment lifetime is two minutes. Here we explicitly ask that a segment be destroyed if it cannot be delivered by the internet system within one minute.

If the lower level is IP (or other protocol that provides this feature) and source routing is used, the interface must allow the route information to be communicated. This is especially important so that the source and destination addresses used in the TCP checksum be the originating source and ultimate destination. It is also important to preserve the return route to answer connection requests.

Any lower level protocol will have to provide the source address, destination address, and protocol fields, and some way to determine the "TCP length", both to provide the functional equivlent service of IP and to be used in the TCP checksum.

3.9. Event Processing

The processing depicted in this section is an example of one possible implementation. Other implementations may have slightly different processing sequences, but they should differ from those in this section only in detail, not in substance.

The activity of the TCP can be characterized as responding to events. The events that occur can be cast into three categories: user calls, arriving segments, and timeouts. This section describes the processing the TCP does in response to each of the events. In many cases the processing required depends on the state of the connection.

Events that occur:

User Calls

OPEN
SEND
RECEIVE
CLOSE
ABORT
STATUS

Arriving Segments

SEGMENT ARRIVES

Timeouts

USER TIMEOUT
RETRANSMISSION TIMEOUT
TIME-WAIT TIMEOUT

The model of the TCP/user interface is that user commands receive an immediate return and possibly a delayed response via an event or pseudo interrupt. In the following descriptions, the term "signal" means cause a delayed response.

Error responses are given as character strings. For example, user commands referencing connections that do not exist receive "error: connection not open".

Please note in the following that all arithmetic on sequence numbers, acknowledgment numbers, windows, et cetera, is modulo 2**32 the size of the sequence number space. Also note that "=<" means less than or equal to (modulo 2**32).

A natural way to think about processing incoming segments is to imagine that they are first tested for proper sequence number (i.e., that their contents lie in the range of the expected "receive window" in the sequence number space) and then that they are generally queued and processed in sequence number order.

When a segment overlaps other already received segments we reconstruct the segment to contain just the new data, and adjust the header fields to be consistent.

Note that if no state change is mentioned the TCP stays in the same state.

OPEN Call

CLOSED STATE (i.e., TCB does not exist)

Create a new transmission control block (TCB) to hold connection state information. Fill in local socket identifier, foreign socket, precedence, security/compartment, and user timeout information. Note that some parts of the foreign socket may be unspecified in a passive OPEN and are to be filled in by the parameters of the incoming SYN segment. Verify the security and precedence requested are allowed for this user, if not return "error: precedence not allowed" or "error: security/compartment not allowed." If passive enter the LISTEN state and return. If active and the foreign socket is unspecified, return "error: foreign socket unspecified"; if active and the foreign socket is specified, issue a SYN segment. An initial send sequence number (ISS) is selected. A SYN segment of the form <SEQ=ISS><CTL=SYN> is sent. Set SND.UNA to ISS, SND.NXT to ISS+1, enter SYN-SENT state, and return.

If the caller does not have access to the local socket specified, return "error: connection illegal for this process". If there is no room to create a new connection, return "error: insufficient resources".

LISTEN STATE

If active and the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

OPEN Call

SYN-SENT STATE
SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection already exists".

SEND Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, then return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

If the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

SYN-SENT STATE
SYN-RECEIVED STATE

Queue the data for transmission after entering ESTABLISHED state. If no space to queue, respond with "error: insufficient resources".

ESTABLISHED STATE
CLOSE-WAIT STATE

Segmentize the buffer and send it with a piggybacked acknowledgment (acknowledgment value = RCV.NXT). If there is insufficient space to remember this buffer, simply return "error: insufficient resources".

If the urgent flag is set, then SND.UP <- SND.NXT-1 and set the urgent pointer in the outgoing segments.

SEND Call

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection closing" and do not service request.

RECEIVE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE
SYN-SENT STATE
SYN-RECEIVED STATE

Queue for processing after entering ESTABLISHED state. If there is no room to queue this request, respond with "error: insufficient resources".

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

If insufficient incoming segments are queued to satisfy the request, queue the request. If there is no queue space to remember the RECEIVE, respond with "error: insufficient resources".

Reassemble queued incoming segments into receive buffer and return to user. Mark "push seen" (PUSH) if this is the case.

If RCV.UP is in advance of the data currently being passed to the user notify the user of the presence of urgent data.

When the TCP takes responsibility for delivering data to the user that fact must be communicated to the sender via an acknowledgment. The formation of such an acknowledgment is described below in the discussion of processing an incoming segment.

CLOSE Call

CLOSE-WAIT STATE

Queue this request until all preceding SENDs have been segmentized; then send a FIN segment, enter CLOSING state.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Respond with "error: connection closing".

ABORT Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVEs should be returned with "error: connection reset" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

All queued SENDs and RECEIVEs should be given "connection reset" notification, delete the TCB, enter CLOSED state, and return.

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE

Send a reset segment:

<SEQ=SND.NXT><CTL=RST>

All queued SENDs and RECEIVEs should be given "connection reset" notification; all segments queued for transmission (except for the RST formed above) or retransmission should be flushed, delete the TCB, enter CLOSED state, and return.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Respond with "ok" and delete the TCB, enter CLOSED state, and return.

CLOSE-WAIT STATE

Since the remote side has already sent FIN, RECEIVEs must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection closing".

CLOSE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVEs are returned with "error: closing" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

Delete the TCB and return "error: closing" responses to any queued SENDs, or RECEIVEs.

SYN-RECEIVED STATE

If no SENDs have been issued and there is no pending data to send, then form a FIN segment and send it, and enter FIN-WAIT-1 state; otherwise queue for processing after entering ESTABLISHED state.

ESTABLISHED STATE

Queue this until all preceding SENDs have been segmentized, then form a FIN segment and send it. In any case, enter FIN-WAIT-1 state.

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

Strictly speaking, this is an error and should receive a "error: connection closing" response. An "ok" response would be acceptable, too, as long as a second FIN is not emitted (the first FIN may be retransmitted though).

STATUS Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Return "state = LISTEN", and the TCB pointer.

SYN-SENT STATE

Return "state = SYN-SENT", and the TCB pointer.

SYN-RECEIVED STATE

Return "state = SYN-RECEIVED", and the TCB pointer.

ESTABLISHED STATE

Return "state = ESTABLISHED", and the TCB pointer.

FIN-WAIT-1 STATE

Return "state = FIN-WAIT-1", and the TCB pointer.

FIN-WAIT-2 STATE

Return "state = FIN-WAIT-2", and the TCB pointer.

CLOSE-WAIT STATE

Return "state = CLOSE-WAIT", and the TCB pointer.

CLOSING STATE

Return "state = CLOSING", and the TCB pointer.

LAST-ACK STATE

Return "state = LAST-ACK", and the TCB pointer.

TIME-WAIT STATE

Return "state = TIME-WAIT", and the TCB pointer.

SEGMENT ARRIVES

If the state is CLOSED (i.e., TCB does not exist) then

all data in the incoming segment is discarded. An incoming segment containing a RST is discarded. An incoming segment not containing a RST causes a RST to be sent in response. The acknowledgment and sequence field values are selected to make the reset sequence acceptable to the TCP that sent the offending segment.

If the ACK bit is off, sequence number zero is used,

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the ACK bit is on,

<SEQ=SEG.ACK><CTL=RST>

Return.

If the state is LISTEN then

first check for an RST

An incoming RST should be ignored. Return.

second check for an ACK

Any acknowledgment is bad if it arrives on a connection still in the LISTEN state. An acceptable reset segment should be formed for any arriving ACK-bearing segment. The RST should be formatted as follows:

<SEQ=SEG.ACK><CTL=RST>

Return.

third check for a SYN

If the SYN bit is set, check the security. If the security/compartment on the incoming segment does not exactly match the security/compartment in the TCB then send a reset and return.

<SEQ=SEG.ACK><CTL=RST>

If the SEG.PRC is greater than the TCB.PRC then if allowed by the user and the system set TCB.PRC<-SEG.PRC, if not allowed send a reset and return.

<SEQ=SEG.ACK><CTL=RST>

If the SEG.PRC is less than the TCB.PRC then continue.

Set RCV.NXT to SEG.SEQ+1, IRS is set to SEG.SEQ and any other control or text should be queued for processing later. ISS should be selected and a SYN segment sent of the form:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

SND.NXT is set to ISS+1 and SND.UNA to ISS. The connection state should be changed to SYN-RECEIVED. Note that any other incoming control or data (combined with SYN) will be processed in the SYN-RECEIVED state, but processing of SYN and ACK should not be repeated. If the listen was not fully specified (i.e., the foreign socket was not fully specified), then the unspecified fields should be filled in now.

fourth other text or control

Any other control or text-bearing segment (not containing SYN) must have an ACK and thus would be discarded by the ACK processing. An incoming RST segment could not be valid, since it could not have been sent in response to anything sent by this incarnation of the connection. So you are unlikely to get here, but if you do, drop the segment, and return.

If the state is SYN-SENT then

first check the ACK bit

If the ACK bit is set

If SEG.ACK =< ISS, or SEG.ACK > SND.NXT, send a reset (unless the RST bit is set, if so drop the segment and return)

<SEQ=SEG.ACK><CTL=RST>

and discard the segment. Return.

If SND.UNA =< SEG.ACK =< SND.NXT then the ACK is acceptable.

second check the RST bit.

SEGMENT ARRIVES

If the RST bit is set

If the ACK was acceptable then signal the user "error: connection reset", drop the segment, enter CLOSED state, delete TCB, and return. Otherwise (no ACK) drop the segment and return.

third check the security and precedence

If the security/compartment in the segment does not exactly match the security/compartment in the TCB, send a reset

If there is an ACK

<SEQ=SEG.ACK><CTL=RST>

Otherwise

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If there is an ACK

The precedence in the segment must match the precedence in the TCB, if not, send a reset

<SEQ=SEG.ACK><CTL=RST>

If there is no ACK

If the precedence in the segment is higher than the precedence in the TCB then if allowed by the user and the system raise the precedence in the TCB to that in the segment, if not allowed to raise the prec then send a reset.

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the precedence in the segment is lower than the precedence in the TCB continue.

If a reset was sent, discard the segment and return.

fourth check the SYN bit

This step should be reached only if the ACK is ok, or there is no ACK, and it the segment did not contain a RST.

If the SYN bit is on and the security/compartment and precedence

are acceptable then, RCV.NXT is set to SEG.SEQ+1, IRS is set to SEG.SEQ. SND.UNA should be advanced to equal SEG.ACK (if there is an ACK), and any segments on the retransmission queue which are thereby acknowledged should be removed.

If SND.UNA > ISS (our SYN has been ACKed), change the connection state to ESTABLISHED, form an ACK segment

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

and send it. Data or controls which were queued for transmission may be included. If there are other controls or text in the segment then continue processing at the sixth step below where the URG bit is checked, otherwise return.

Otherwise enter SYN-RECEIVED, form a SYN,ACK segment

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

and send it. If there are other controls or text in the segment, queue them for processing after the ESTABLISHED state has been reached, return.

fifth, if neither of the SYN or RST bits is set then drop the segment and return.

SEGMENT ARRIVES

Otherwise,

first check sequence number

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

There are four cases for the acceptability test for an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the acknowledgment, drop the unacceptable segment and return.

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN), and only processing further if the segment then begins at RCV.NXT. Segments with higher begining sequence numbers may be held for later processing.

second check the RST bit,

SYN-RECEIVED STATE

If the RST bit is set

If this connection was initiated with a passive OPEN (i.e., came from the LISTEN state), then return this connection to LISTEN state and return. The user need not be informed. If this connection was initiated with an active OPEN (i.e., came from SYN-SENT state) then the connection was refused, signal the user "connection refused". In either case, all segments on the retransmission queue should be removed. And in the active OPEN case, enter the CLOSED state and delete the TCB, and return.

ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT

If the RST bit is set then, any outstanding RECEIVEs and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT

If the RST bit is set then, enter the CLOSED state, delete the TCB, and return.

SEGMENT ARRIVES

third check security and precedence

SYN-RECEIVED

If the security/compartment and precedence in the segment do not exactly match the security/compartment and precedence in the TCB then send a reset, and return.

ESTABLISHED STATE

If the security/compartment and precedence in the segment do not exactly match the security/compartment and precedence in the TCB then send a reset, any outstanding RECEIVEs and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

Note this check is placed following the sequence check to prevent a segment from an old connection between these ports with a different security or precedence from causing an abort of the current connection.

fourth, check the SYN bit,

SYN-RECEIVED
ESTABLISHED STATE
FIN-WAIT STATE-1
FIN-WAIT STATE-2
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

If the SYN is in the window it is an error, send a reset, any outstanding RECEIVEs and SEND should receive "reset" responses, all segment queues should be flushed, the user should also receive an unsolicited general "connection reset" signal, enter the CLOSED state, delete the TCB, and return.

If the SYN is not in the window this step would not be reached and an ack would have been sent in the first step (sequence number check).

fifth check the ACK field,

if the ACK bit is off drop the segment and return

if the ACK bit is on

SYN-RECEIVED STATE

If SND.UNA =< SEG.ACK =< SND.NXT then enter ESTABLISHED state and continue processing.

If the segment acknowledgment is not acceptable, form a reset segment,

<SEQ=SEG.ACK><CTL=RST>

and send it.

ESTABLISHED STATE

If SND.UNA < SEG.ACK =< SND.NXT then, set SND.UNA <- SEG.ACK. Any segments on the retransmission queue which are thereby entirely acknowledged are removed. Users should receive positive acknowledgments for buffers which have been SENT and fully acknowledged (i.e., SEND buffer should be returned with "ok" response). If the ACK is a duplicate (SEG.ACK < SND.UNA), it can be ignored. If the ACK acks something not yet sent (SEG.ACK > SND.NXT) then send an ACK, drop the segment, and return.

If SND.UNA < SEG.ACK =< SND.NXT, the send window should be updated. If (SND.WL1 < SEG.SEQ or (SND.WL1 = SEG.SEQ and SND.WL2 =< SEG.ACK)), set SND.WND <- SEG.WND, set SND.WL1 <- SEG.SEQ, and set SND.WL2 <- SEG.ACK.

Note that SND.WND is an offset from SND.UNA, that SND.WL1 records the sequence number of the last segment used to update SND.WND, and that SND.WL2 records the acknowledgment number of the last segment used to update SND.WND. The check here prevents using old segments to update the window.

SEGMENT ARRIVES

FIN-WAIT-1 STATE

In addition to the processing for the ESTABLISHED state, if our FIN is now acknowledged then enter FIN-WAIT-2 and continue processing in that state.

FIN-WAIT-2 STATE

In addition to the processing for the ESTABLISHED state, if the retransmission queue is empty, the user's CLOSE can be acknowledged ("ok") but do not delete the TCB.

CLOSE-WAIT STATE

Do the same processing as for the ESTABLISHED state.

CLOSING STATE

In addition to the processing for the ESTABLISHED state, if the ACK acknowledges our FIN then enter the TIME-WAIT state, otherwise ignore the segment.

LAST-ACK STATE

The only thing that can arrive in this state is an acknowledgment of our FIN. If our FIN is now acknowledged, delete the TCB, enter the CLOSED state, and return.

TIME-WAIT STATE

The only thing that can arrive in this state is a retransmission of the remote FIN. Acknowledge it, and restart the 2 MSL timeout.

sixth, check the URG bit,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

If the URG bit is set, RCV.UP <- max(RCV.UP,SEG.UP), and signal the user that the remote side has urgent data if the urgent pointer (RCV.UP) is in advance of the data consumed. If the user has already been signaled (or is still in the "urgent mode") for this continuous sequence of urgent data, do not signal the user again.

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT

This should not occur, since a FIN has been received from the remote side. Ignore the URG.

seventh, process the segment text,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

Once in the ESTABLISHED state, it is possible to deliver segment text to user RECEIVE buffers. Text from segments can be moved into buffers until either the buffer is full or the segment is empty. If the segment empties and carries an PUSH flag, then the user is informed, when the buffer is returned, that a PUSH has been received.

When the TCP takes responsibility for delivering the data to the user it must also acknowledge the receipt of the data.

Once the TCP takes responsibility for the data it advances RCV.NXT over the data accepted, and adjusts RCV.WND as apporopriate to the current buffer availability. The total of RCV.NXT and RCV.WND should not be reduced.

Please note the window management suggestions in section 3.7.

Send an acknowledgment of the form:

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

This acknowledgment should be piggybacked on a segment being transmitted if possible without incurring undue delay.

SEGMENT ARRIVES

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

This should not occur, since a FIN has been received from the remote side. Ignore the segment text.

eighth, check the FIN bit,

Do not process the FIN if the state is CLOSED, LISTEN or SYN-SENT since the SEG.SEQ cannot be validated; drop the segment and return.

If the FIN bit is set, signal the user "connection closing" and return any pending RECEIVEs with same message, advance RCV.NXT over the FIN, and send an acknowledgment for the FIN. Note that FIN implies PUSH for any segment text not yet delivered to the user.

SYN-RECEIVED STATE
ESTABLISHED STATE

Enter the CLOSE-WAIT state.

FIN-WAIT-1 STATE

If our FIN has been ACKed (perhaps in this segment), then enter TIME-WAIT, start the time-wait timer, turn off the other timers; otherwise enter the CLOSING state.

FIN-WAIT-2 STATE

Enter the TIME-WAIT state. Start the time-wait timer, turn off the other timers.

CLOSE-WAIT STATE

Remain in the CLOSE-WAIT state.

CLOSING STATE

Remain in the CLOSING state.

LAST-ACK STATE

Remain in the LAST-ACK state.

TIME-WAIT STATE

Remain in the TIME-WAIT state. Restart the 2 MSL time-wait timeout.

and return.

USER TIMEOUT

For any state if the user timeout expires, flush all queues, signal the user "error: connection aborted due to user timeout" in general and for any outstanding calls, delete the TCB, enter the CLOSED state and return.

RETRANSMISSION TIMEOUT

For any state if the retransmission timeout expires on a segment in the retransmission queue, send the segment at the front of the retransmission queue again, reinitialize the retransmission timer, and return.

TIME-WAIT TIMEOUT

If the time-wait timeout expires on a connection delete the TCB, enter the CLOSED state and return.

September 1981

Transmission Control Protocol

[Page 78]

GLOSSARY

1822
: BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ACK
: A control bit (acknowledge) occupying no sequence space, which indicates that the acknowledgment field of this segment specifies the next sequence number the sender of this segment is expecting to receive, hence acknowledging receipt of all previous sequence numbers.

ARPANET message
: The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet
: A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

connection
: A logical communication path identified by a pair of sockets.

datagram
: A message sent in a packet switched computer communications network.

Destination Address
: The destination address, usually the network and host identifiers.

FIN
: A control bit (finis) occupying one sequence number, which indicates that the sender will send no more data or control occupying sequence space.

fragment
: A portion of a logical unit of data, in particular an internet fragment is a portion of an internet datagram.

FTP
: A file transfer protocol.

September 1981

Transmission Control Protocol
Glossary

header
Control information at the beginning of a message, segment, fragment, packet or block of data.

host
A computer. In particular a source or destination of messages from the point of view of the communication network.

Identification
An Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.

IMP
The Interface Message Processor, the packet switch of the ARPANET.

internet address
A source or destination address specific to the host level.

internet datagram
The unit of data exchanged between an internet module and the higher level protocol together with the internet header.

internet fragment
A portion of the data of an internet datagram with an internet header.

IP
Internet Protocol.

IRS
The Initial Receive Sequence number. The first sequence number used by the sender on a connection.

ISN
The Initial Sequence Number. The first sequence number used on a connection, (either ISS or IRS). Selected on a clock based procedure.

ISS
The Initial Send Sequence number. The first sequence number used by the sender on a connection.

leader
Control information at the beginning of a message or block of data. In particular, in the ARPANET, the control information on an ARPANET message at the host-IMP interface.

[Page 80]

left sequence
: This is the next sequence number to be acknowledged by the data receiving TCP (or the lowest currently unacknowledged sequence number) and is sometimes referred to as the left edge of the send window.

local packet
: The unit of transmission within a local network.

module
: An implementation, usually in software, of a protocol or other procedure.

MSL
: Maximum Segment Lifetime, the time a TCP segment can exist in the internetwork system. Arbitrarily defined to be 2 minutes.

octet
: An eight bit byte.

Options
: An Option field may contain several options, and each option may be several octets in length. The options are used primarily in testing situations; for example, to carry timestamps. Both the Internet Protocol and TCP provide for options fields.

packet
: A package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.

port
: The portion of a socket that specifies which logical input or output channel of a process is associated with the data.

process
: A program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.

PUSH
: A control bit occupying no sequence space, indicating that this segment contains data that must be pushed through to the receiving user.

RCV.NXT
: receive next sequence number

RCV.UP
: receive urgent pointer

RCV.WND
: receive window

receive next sequence number
: This is the next sequence number the local TCP is expecting to receive.

receive window
: This represents the sequence numbers the local (receiving) TCP is willing to receive. Thus, the local TCP considers that segments overlapping the range RCV.NXT to RCV.NXT + RCV.WND - 1 carry acceptable data or control. Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.

RST
: A control bit (reset), occupying no sequence space, indicating that the receiver should delete the connection without further interaction. The receiver can determine, based on the sequence number and acknowledgment fields of the incoming segment, whether it should honor the reset command or ignore it. In no case does receipt of a segment containing RST give rise to a RST in response.

RTP
: Real Time Protocol: A host-to-host protocol for communication of time critical information.

SEG.ACK
: segment acknowledgment

SEG.LEN
: segment length

SEG.PRC
: segment precedence value

SEG.SEQ
: segment sequence

SEG.UP
: segment urgent pointer field

SEG.WND
: segment window field

segment
: A logical unit of data, in particular a TCP segment is the unit of data transfered between a pair of TCP modules.

segment acknowledgment
: The sequence number in the acknowledgment field of the arriving segment.

segment length
: The amount of sequence number space occupied by a segment, including any controls which occupy sequence space.

segment sequence
: The number in the sequence field of the arriving segment.

send sequence
: This is the next sequence number the local (sending) TCP will use on the connection. It is initially selected from an initial sequence number curve (ISN) and is incremented for each octet of data or sequenced control transmitted.

send window
: This represents the sequence numbers which the remote (receiving) TCP is willing to receive. It is the value of the window field specified in segments from the remote (data receiving) TCP. The range of new sequence numbers which may be emitted by a TCP lies between SND.NXT and SND.UNA + SND.WND - 1. (Retransmissions of sequence numbers between SND.UNA and SND.NXT are expected, of course.)

SND.NXT
: send sequence

SND.UNA
: left sequence

SND.UP
: send urgent pointer

SND.WL1
: segment sequence number at last window update

SND.WL2
: segment acknowledgment number at last window update

SND.WND
send window

socket
An address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port.

Source Address
The source address, usually the network and host identifiers.

SYN
A control bit in the incoming segment, occupying one sequence number, used at the initiation of a connection, to indicate where the sequence numbering will start.

TCB
Transmission control block, the data structure that records the state of a connection.

TCB.PRC
The precedence of the connection.

TCP
Transmission Control Protocol: A host-to-host protocol for reliable communication in internetwork environments.

TOS
Type of Service, an Internet Protocol field.

Type of Service
An Internet Protocol field which indicates the type of service for this internet fragment.

URG
A control bit (urgent), occupying no sequence space, used to indicate that the receiving user should be notified to do urgent processing as long as there is data to be consumed with sequence numbers less than the value indicated in the urgent pointer.

urgent pointer
A control field meaningful only when the URG bit is on. This field communicates the value of the urgent pointer which indicates the data octet associated with the sending user's urgent call.

REFERENCES

[1] Cerf, V., and R. Kahn, "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, May 1974.

[2] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.

[3] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks, Vol. 2, No. 6, pp. 454-473, December 1978.

[4] Postel, J., "Assigned Numbers", RFC 790, USC/Information Sciences Institute, September 1981.

RFC - 869

A Host Monitoring Protocol

Robert M. Hinden

BBN Communications Corporation

December 1983

Table of Contents

RFC-869
Replaces IEN-197

December 1983

A Host Monitoring Protocol

1 Introduction

The Host Monitoring Protocol (HMP) is used to collect information from hosts in various networks. A host is defined as an addressable Internet entity that can send and receive messages; this includes hosts such as server hosts, personal work stations, terminal concentrators, packet switches, and gateways. At present the Host Monitoring Protocol is being used to collect information from Internet Gateways and TACs, and implementations are being designed for other hosts. It is designed to monitor hosts spread over the internet as well as hosts in a single network.

This document is organized into three parts. Section 2 and 3 contains a general description of the Host Monitoring protocol and its relationship to other protocols. Section 4 describes how it operates. Section 5 and 6 contain the descriptions and formats of the HMP messages. These are followed by appendices containing the formats of messages sent by some of the hosts that use the HMP to collect their monitoring information. These appendicies included as examples only and are not part of the HMP protocol.

RFC-869 December 1983

This document replaces the previous HMP document "IEN-197, A Host Monitoring Protocol."

2 General Description

The Host Monitoring Protocol is a transaction-oriented (i.e., connection-less) transport protocol. It was designed to facilitate certain simple interactions between two internet entities, one of which may be considered to be "monitoring" the other. (In discussing the protocol we will sometimes speak of a "monitoring host" and a "monitored entity".) HMP was intended to be a useful transport protocol for applications that involve any or all of the following three different kinds of interactions:

- The monitored entity sometimes needs to send unsolicited datagrams to the monitoring host. The monitoring host should be able to tell when messages from the monitored entity have been lost in transit, and it should be able to determine the order in which the messages were sent, but the application does not require that all messages be received or that they be received strictly in the same sequence in which they were sent.

- The monitoring host needs to gather data from the monitored entity by using a query-response protocol at the application level. It is important to be able to determine which query is being answered by a particular response, and to determine whether successive responses are duplicates of previous ones.

- The monitoring host must be able to initiate certain control functions in the monitored entity, possibly including the setting of parameters in the monitored entity. The monitoring host needs to know if the control function has been carried out.

In addition, we assume that a given monitoring host may be monitoring several different types of entities simultaneously, and may be gathering several different types of data from a given

type of monitored entity. Several different monitoring hosts may be monitoring a given entity, and several processes on the same host may even be monitoring the same entity.

Messages from the monitoring host to the monitored entity are called "polls". They need to contain enough information to allow the monitored entity to make the following determinations:

- The monitored entity must be able to determine that this message is in fact a poll from a monitoring host. The "system type," "message type," and "password" fields in the HMP header have been defined to meet this need.
- The monitored entity may need to be able to identify the particular process on the monitoring host that sent this poll, so it can send its response back to the right process. The "port number" field in the HMP header has been defined to meet this need.
- The monitored entity must be able to indicate to the monitoring host, in its response, precisely which query is being answered by a particular response. The "sequence number field" has been defined to meet this need.
- The monitored entity must be able to determine just what kind of action the monitoring host is requesting. That is, the HMP transport protocol must provide some way of multiplexing and demultiplexing the various higher-level applications which use it. The "R-message type" and "R-subtype" fields of the polling message have been defined to meet this need.

Messages from the monitored entity to the monitoring host need to contain enough information to enable the monitoring host to make the following determination:

- The monitoring host must be able to route this message to the correct process. The "port number" field meets this need.

- The monitoring host must be able to match up received messages with the polls, if any, that elicited them. The "returned sequence number" field in the HMP header has been defined to meet this need.

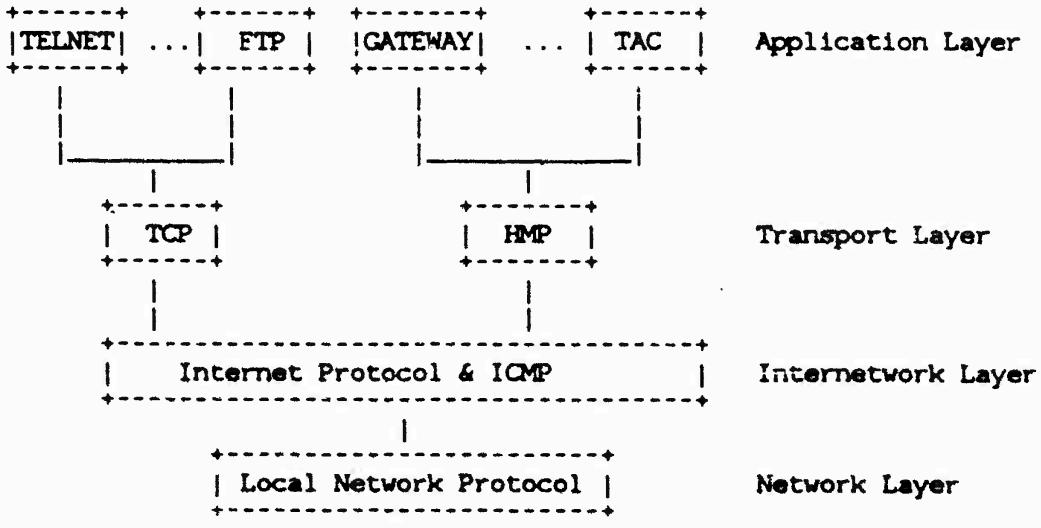
- The monitoring host must be able to determine which higher level application should receive a particular message. The "system type" and "message type" fields are used for this purpose.

- The monitoring host must be able to determine whether some messages of a given type were lost in transit, and whether messages have arrived out of sequence. Although this function, strictly speaking, belongs to the application and not to the transport layer, the HMP header contains a "sequence number" for this purpose.

In addition, a simple one's complement checksum is provided in the HMP header to detect data corruption during transmission.

3 Relationship to Other Protocols

The Host Monitoring Protocol is a transport protocol designed to fit into the layered internet protocol environment. It operates on top of the Internet/ICMP protocol and under applications that require its services. This relationship is illustrated in the following diagram:



If internetwork services are not required it should be possible to run the HMP without an Internetwork layer. As long as HMPs' service requirments (addressing, protocol demultiplexing, and occasional delivery) are met it should run over a variety of protocols.

4 Protocol Operation

The HMP is built around the idea that most of the intelligence needed to monitor a host should reside in a monitoring center, not in the host. The host should be required only to collect data and send it to the monitoring center, either spontaneously or on request from the monitoring center. The host is not responsible for insuring that the data arrives reliably (except that it checksums the data); instead, the monitoring center is responsible for ensuring that the data it requests is received correctly.

Consequently, the HMP is based on polling hosts for messages. When the monitoring center requires a particular type of data (e.g., throughput data), it sends a poll to the host requesting that type of report. The host, upon receiving the poll, responds with its latest set of collected data. If the host finds that the poll is incorrect (e.g., if the poll was for throughput data and the host is not collecting throughput data), it responds with an error message. The monitoring center waits a reasonable length of time for the host to answer its poll. If no response is received, it sends another poll for the same data. In this way, if either a poll or the response is lost, the correct data is still collected.

The HMP is used to collect three different classes of data:

- o Spontaneous Events (or Traps)
- o Current status
- o Statistical data collected over time

These classes of data allow a host to send data in a manner best suited to the data. For instance, the host may quickly inform the monitoring center that a particular event has happened by sending a trap message, while the monitoring center is reliably collecting the host's throughput and accounting data.

Traps report spontaneous events, as they occur, to the monitoring center. In order to insure their prompt delivery, the traps are sent as datagrams with no reliability mechanisms (except checksums) such as acknowledgments and retransmissions. Trap messages usually contain an identifier to indicate which event is being reported, the local time in the host that the event occured, and data pertinent to the event. The data portion is intended to be host and event specific.

Status information, the second type of data collected by the Host Monitoring Protocol describes the current state of the host. Status information is useful at one point, but it does not have to be collected cumulatively over a certain period of time. Only the latest status is of interest; old status provides no useful information. The monitoring center collects status information

by sending a poll for status to a host. Upon receiving the poll, the host responds with its latest status information, always creating a new status message. If the monitoring center does not receive a response to its poll, it sends another poll. The monitoring center can decide if the host is up or down based on whether the host responds to its polls.

The third type of data collected by the HMP is statistical data. These are measurements taken over time, such as the number of packets sent or received by a host and the count of packets dropped for a particular reason. It is important that none of this type of data be lost. Statistical data is collected in a host over a time interval. When the collection time interval expires, the current data is copied to another area, and the counters are cleared. The copied data is sent to the monitoring center when the host receives a poll requesting statistical information. If another poll is received before the collection time interval has expired, the data in the buffer is sent again. The monitoring center can detect duplicate messages by using the sequence number in the header of the message, since each type of statistical data has its own sequence number counter.

The collection frequency for statistics messages from a particular host must be relatively long compared to the average round trip message time between the monitoring center and that host inorder to allow the monitoring center to re-poll if it does

not receive an answer. With this restriction, it should be possible to avoid missing any statistics messages. Each statistics message contains a field giving the local time when the data was collected and the time at which the message was sent. This information allows the monitoring center to schedule when it sends a poll so that the poll arrives near the beginning of each collection period. This ensures that if a message is lost, the monitoring center will have sufficient time to poll again for the statistics message for that period.

The HMP also includes a provision to send data to and read parameters in hosts. The data may be used to set switches or interval timers used to control measurements in a host, or to control the host itself (e.g. a restart switch). The format of the data and parameters is host specific.

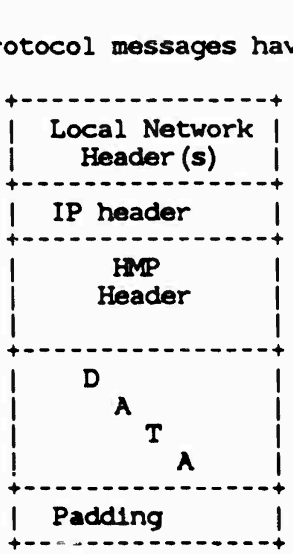
To send data to a host, the monitoring center sends the host a poll for a control-acknowledgment message. This poll message includes the type of the data and the data being sent. When the host receives this poll, it processes the data and responds with a control-acknowledgment message.

To read parameters in a host, the monitoring center will send a poll for parameters to the host. This poll includes the type of the parameters being read. When the host receives this poll, it will send the parameters of the requested type to the

monitoring center in a parameters message.

5 Header Formats

Host Monitor Protocol messages have the following format:



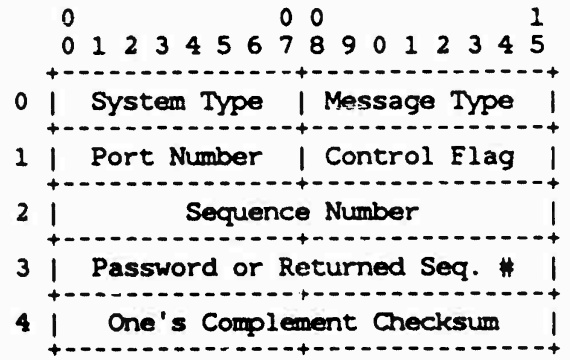
5.1 IP Headers

HMP messages are sent using the version 4 IP header as described in RFC-791 "Internet Protocol." The HMP protocol number is 20 (decimal). The time to live field should be set to a reasonable value for the hosts being monitored.

All other fields should be set as specified in RFC-791.

5.2 HMP Header

The HMP header format is:



HMP FIELDS:

System Type
Message Type

The combination of system type and message type determines the format of the data in the monitoring message.

The system types which have been defined are:

System Type	Meaning
1	Monitoring Host
2	IMP
3	TAC
4	Gateway
5	SIMP
6	BBN VAX/C70 TCP
7	PAD
8	Reserved
9	TIU
10	FEP
11	Cronus Host
12	Cronus MCS

Message types are defined and used for each system type according to the needs of that system. The message types currently defined are:

Type	Description
1	Trap
2	Status
3	Thruput
4	HTM - Host Traffic Matrix
5	Parameters
6	Routing
7	Call Accounting
100	Poll
101	Error
102	Control Acknowledgment

Port Number

This field can be used to multiplex similar messages to/from different processes in one host. It is currently unused.

Control Flag

This field is used to pass control information. Currently Bit 15 is defined as the "More bit" which is used in a message in responce to a poll to indicate that there is more data to poll for.

Sequence Number

Every message contains a sequence number. The sequence number is incremented when each new message of that type is sent.

Password or Returned Sequence Number

The Password field of a polling message from an monitoring center contains a password to verify that the monitoring center is allowed to gather information. Responses to polling messages copy the Sequence Number from the polling message and return it in this field for

identification and round-trip time calculations.

Checksum

The Checksum field is the one's complement of the one's complement sum of all the 16-bit words in the header and data area.

6 HMP Monitoring Center Message Formats

6.1 Message Type 100: Polling Message

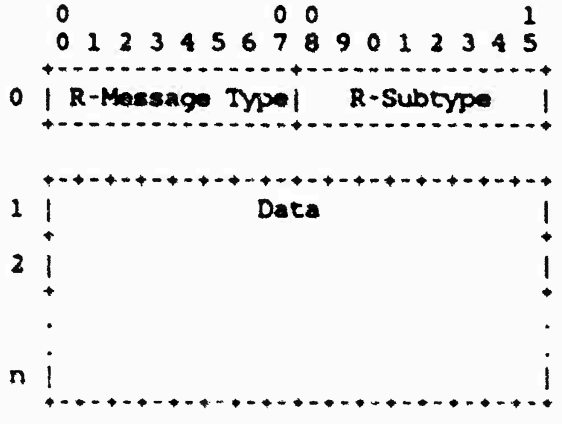
Description

The monitoring center will send polls to the hosts it is monitoring to collect their monitoring data. When the host receives a poll it will return a message of the type requested. It will only answer a poll with the correct system type and password and will return an error message (Message Type 101) if it receives a poll for the wrong system type or an unsupported message type.

The Poll message includes a facility to send data to a monitored host. The poll message to send data consists of a poll for a Control Acknowledgment message (type 102) followed by the data. The R-Subtype specifies the type of the data that is being sent. When the monitored host receives a Poll for a Control acknowledgment, it processes the data, and then responds with an Control acknowledgment message. If the monitored host can not process the data, it should respond with an error message.

A poll to read parameters consists a poll for a Parameters message. The R-Subtype specifies the type of parameters being read. When the monitored host receives a poll for a Parameters message, it responds with a Parameters message containing the requested information.

A polling message has the following form:



HMP FIELDS

System Type

The type of machine being polled.

Message Type

Polling Message = 100

Port Number

Unused

Control Flag

Unused

Sequence Number

The sequence number identifies the polling request. The Monitoring Center will maintain separate sequence numbers for each host it monitors. This sequence number is returned in the response to a poll and the monitoring center will use this information to associate polls with their responses and to determine round trip times.

Password

The monitoring password.

POLL FIELDS

R-Message Type

The message type requested.

R-Subtype

This field is used when sending data and reading parameters and it specifies the type of the data being sent or parameters being read.

Data

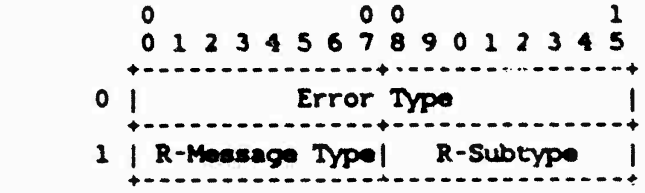
When the poll is requesting a Control Acknowledgment message, data is included in the poll message. A poll for any other type of message does not include any data . The contents of the data is host specific.

6.2 Message Type 101: Error in Poll

Description

This message is sent in response to a faulty poll and specifies the nature of the error.

An error message has the following form:



HMP FIELDS

System Type

The type of machine sending message.

Message Type

Error Message = 101

Port Number

Unused

Control Flag

Unused

Sequence Number

A 16 bit number incremented each time an error message is sent.

Returned Sequence Number

The Sequence Number of the polling message which caused the error.

ERROR MESSAGE FIELDS

Error Type

This field specifies the nature of the error in the poll. The following error types have been defined.

1 = Reason unspecified.
2 = Bad R-Message Type.
3 = Bad R-Subtype.
4 = Unknown parameter
5 = Invalid parameter value
6 = Invalid parameter/value format
7 = Machine in Loader

R-Message Type
R-Subtype

These fields identify the poll request in error.

6.3 Message Type 102: Control acknowledgment

Description

This message is sent in response to a poll for this type of message. It is used to acknowledge poll messages that are used to set parameters in the monitored host.

The Control acknowledgment has no fields other than the HMP header.

HMP FIELDS

System Type

The type of the system sending the message.

Message Type

Control acknowledgment = 102

Port Number

Unused

Control Flag

Unused

Sequence Number

A 16 bit number incremented each time a Control acknowledgment message is sent.

Returned Sequence Number

The Sequence Number of the polling message which requested this message.

A Appendix A - IMP Monitoring

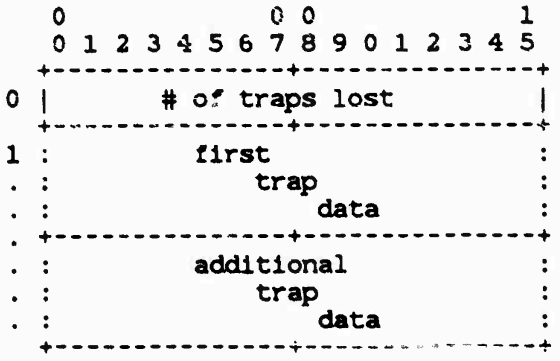
A.1 Message Type 1: IMP Trap

Description

When a trap occurs, it is buffered in the IMP and sent as soon as possible. Trap messages are unsolicited. If traps happen in close sequence, several traps may be sent in one message.

Through the use of sequence numbers, it will be possible to determine how many traps are being lost. If it is discovered that many are lost, a polling scheme might be implemented for traps.

A IMP trap message has the following form:



HMP Fields

System Type

IMP = 2

Message Type

IMP Trap Message = 1

Port Number

Unused

Control Flag

Unused

Password

Unused

Sequence Number

A 16 bit number incremented each time a trap message is sent so that the HM can order the received trap messages and detect missed messages.

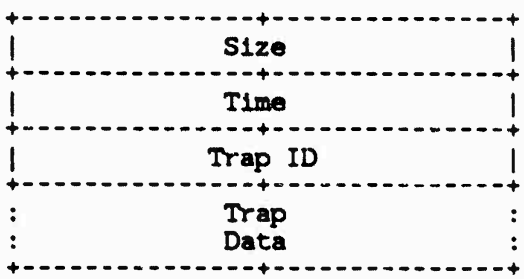
IMP TRAP FIELDS

# of traps lost

Under certain conditions, an IMP may overflow its internal trap buffers and be unable to save traps to send. This counter keeps track of such occurrences.

Trap Reports

There can be several blocks of trap data in each message. The format for each such block is below.



Size

Size is the number of 16 bit words in the trap, not counting the size field.

Time

The time (in 640 ms. units) at which the trap occurred.

This field is used to sequence the traps in a message and associate groups of traps.

Trap ID

This is usually the program counter at the trap. The ID identifies the trap, and does not have to be a program counter, provided it uniquely identifies the trap.

Trap Data

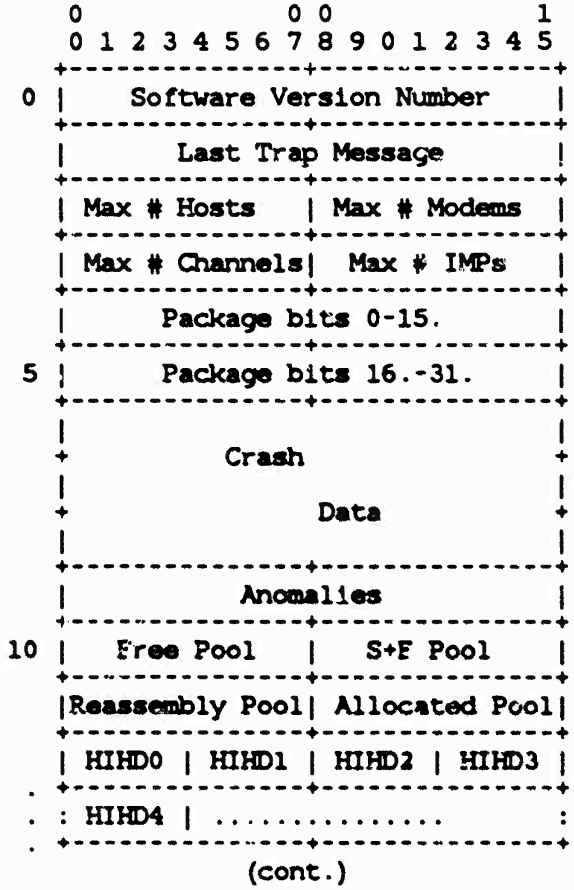
The IMP returns data giving more information about the trap. There are usually two entries: the values in the accumulator and the index register at the occurrence of the trap.

A.2 Message Type 2: IMP status

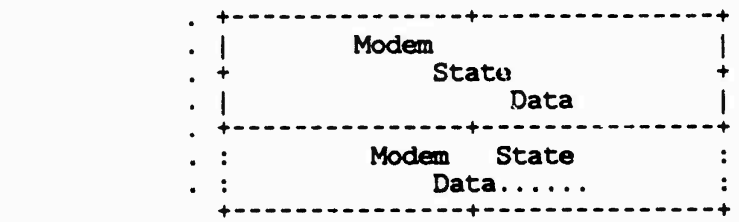
Description

The status message gives a quick summary of the state of the IMP. Status of the most important features of the IMP are reported as well as the current configuration of the machine.

The format of the status message is as follows:



Imp Status (cont.)



HMP FIELDS

System Type

IMP = 2

Message Type

IMP status message = 2

Port Number

Unused

Control Flag

Unused

Sequence Number

A 16 bit number incremented each time a status message is sent.

Password

The password contains the sequence number of the polling message to which this message responds.

IMP STATUS FIELDS

Software Version Number

The IMP version number.

Last Trap Message

Contains the sequence number of the last trap message sent to the HM. This will allow the HM to detect how many trap messages are being lost.

Hosts

The number of configured hosts in this system.

Modems

The number of configured modems in this system.

Channels

The maximum possible number of IMP-IMP channels in this system.

IMPs

The maximum possible number of IMPs in this system.

Package Bits

This is a bit encoded word that reports the set of packages currently loaded in the system. The table below defines the bits.

Bit (octal)	Package
(1st Word)	
1	VDH
2	Logical address tables
4	Mezmode
10	Cumulative Statistics
20	Trace
40	TTY
100	DDT
200	HDLC
400	HDH
1000	Cassette Writer
2000	Propagation Delay Measurement
4000	X25
10000	Profile Measurements
20000	Self Authenticating Password
40000	Host traffic Matrix
100000	Experimental/Special
(2nd Word)	
1	End-to-end Statistics
2	Store and Forward statistics

Crash Data

Crash data reports the circumstances surrounding an unexpected crash. The first word reports the location of the crash and the following two are the contents of the accumulator and index registers.

Anomalies

Anomalies is a collection of bit flags that indicate the state of various switches or processes in the IMP. These are very machine dependent and only a representative sampling of bits is listed below.

Bit (octal)	Meaning
20	Override ON
200	Trace ON
1000	Statistics ON
2000	Message Generator ON
4000	Packet Trace ON
10000	Host Data Checksum is BAD
20000	Reload Location SET

Buffer Pool Counts

These are four bytes of counters indicating the current usage of buffers in the IMP. The four counters are: free buffers, store-and-forward buffers, reassembly buffers and allocated buffers.

HIHD0 - HIHDn

Each four bit HIHD field gives the state of the corresponding host.

Value	Meaning
0	UP
1	ready line down
2	tardy
3	non-existent

Modem State Data

Modem state data contains six fields of data distributed over four words. The first field (4 bits) indicates the line speed; the second field (4 bits) is the number of the modem that is used by the neighboring IMP on this line; the third field (8 bits) is the number of line protocol ticks covered by this report; the fourth (1 bit) indicates line down(1) or up(0); the fifth (7 bits) is the IMP number of neighbor IMP on the line; and the sixth (8 bits) is a count of missed protocol packets over the interval specified in the third field.

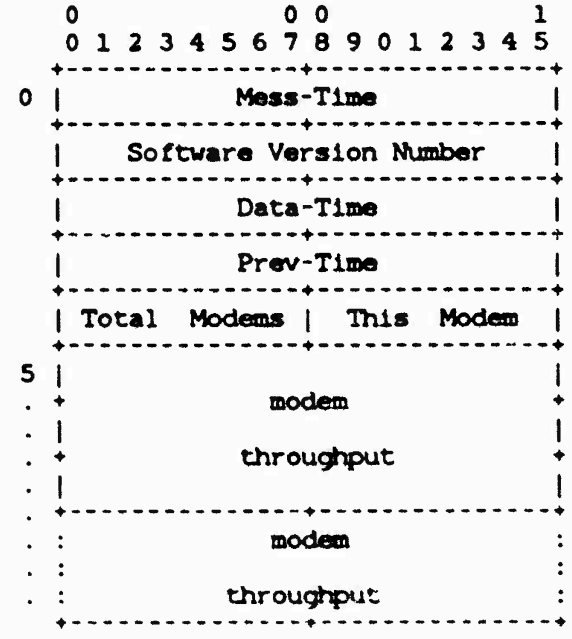
A.3 Message Type 3: IMP Modem Throughput

Description

The modem throughput message reports traffic statistics for each modem in the system. The IMP will collect these data at regular intervals and save them awaiting a poll from the HM. If a period is missed by the HM, the new results simply overwrite the old. Two time stamps bracket the collection interval (data-time and prev-time) and are an indicator of missed reports. In addition, mess-time indicates the time at which the message was sent.

The modem throughput message will accommodate up to fourteen modems in one packet. A provision is made to split this into multiple packets by including a modem number for the first entry in the packet. This field is not immediately useful, but if machine sizes grow beyond fourteen modems or if modem statistics become more detailed and use more than three words per modem, this can be used to keep the message within a single ARPANET packet.

The format of the modem throughput message is as follows:



HMP FIELDS

System Type

IMP = 2

Message Type

IMP Modem Throughput message = 3

Port Number

Unused

Control Flag

Unused

Sequence Number

A 16 bit number incremented at each collection interval (i.e. when a new throughput message is assembled). The HM will be able to detect lost or duplicate messages by checking the sequence numbers.

Password

The password contains the sequence number of the polling message to which this message responds.

IMP MODEM THROUGHPUT FIELDS

Mess-time

The time (in 640ms. units) at which the message was sent to the HM.

Software Version Number

The IMP version number.

Data-Time

Data-time is the time (in 640ms. units) when this set of data was collected. (See Description.)

Prev-Time

Prev-time is the time (in 640 ms. units) of the previous collection of data (and therefore, is the time when the data in this message began accumulating.)

Total Modems

This is the number of modems in the system.

This Modem

This Modem is the number of the first modem reported in this message. Large systems that are unable to fit all their modem reports into a single packet may use this field to separate their message into smaller chunks to take advantage of single packet message efficiencies.

Modem Throughput

Modem throughput consists of three words of data reporting packets and words output on each modem. The first word counts packets output and the following two count word throughput. The double precision words are arranged high order first. (Note also that messages from Honeywell type machines (316s, 516s and C30s) use a fifteen bit low order word.) The first block reports output on the modem specified by "This Modem". The following blocks report on consecutive modems.

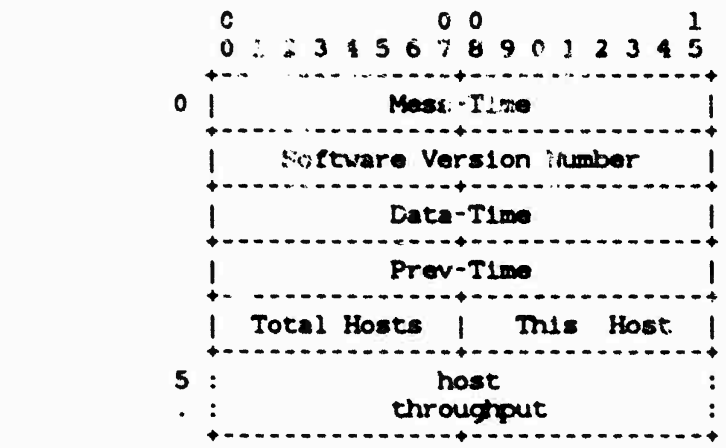
A.4 Message Type 4: IMP Host Throughput

Description

The host throughput message reports traffic statistics for each host in the system. The IMP will collect these data at regular intervals and save them awaiting a poll from the HM. If a period is missed by the HM, the new results simply overwrite the old. Two time stamps bracket the collection interval (data-time and prev-time) and are an indicator of missed reports. In addition, mess-time indicates the time at which the message was sent.

The host throughput format will hold only three hosts if packet boundaries are to be respected. A provision is made to split this into multiple packets by including a host number for the first entry in the packet.

The format of the host throughput message is as follows:



IMP FIELDS

System Type

IMP = 2

Message Type

IMP host Throughput message = 4

Port Number

Unused

Control Flag

Unused

Sequence Number

A 16 bit number incremented at each collection interval (i.e. when a new throughput message is assembled). The HM will be able to detect lost or duplicate messages by checking the sequence numbers.

Password

The password contains the sequence number of the polling message to which this message responds.

IMP HOST THROUGHPUT FIELDS

Mess-time

The time (in 640ms. units) at which the message was sent to the HM.

Software Version Number

The IMP version number.

Data-Time

Data-time is the time (in 640ms. units) when this set of data was collected. (See Description.)

Prev-Time

Prev-time is the time (in 640 ms. units) of the previous collection of data (and therefore, is the time when the data in this message began accumulating.)

Total Hosts

The total number of hosts in this system.

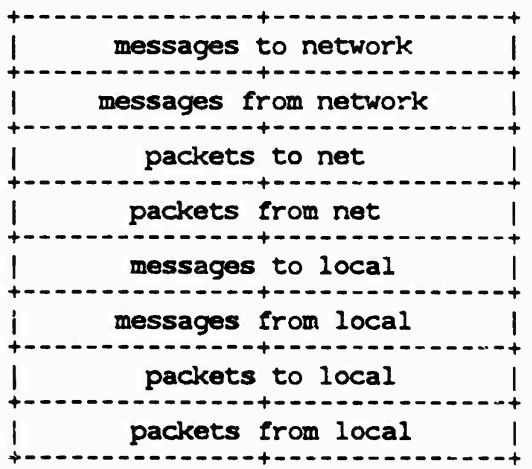
This Host

This host is the number of the first host reported in this

message. Large systems that are unable to fit all their host reports into a single packet may use this field to separate their message into smaller chunks to take advantage of single packet message efficiencies.

Host Throughput

Each host throughput block consists of eight words in the following format:



Each host throughput message will contain several blocks of data. The first block will contain data for the host specified in First Host Number. Following blocks will contain data for consecutive hosts. All counters are single precision.

B Appendix B - TAC Monitoring

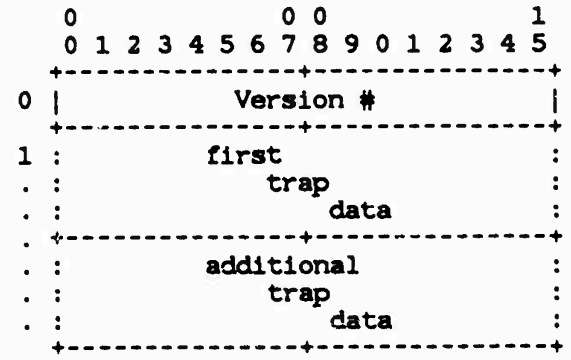
B.1 Message Type 1: TAC Trap Message

Description

When a trap occurs, it is buffered in the TAC and sent as soon as possible. Trap messages are unsolicited. If traps happen in close sequence, several traps may be sent in one message.

Through the use of sequence numbers, it will be possible to determine how many traps are being lost. If it is discovered that many are lost, a polling scheme might be implemented for traps.

A TAC trap message has the following form:



HMP FIELDS

System Type

TAC = 3

Message Type

TAC Trap Message = 1

Port Number

Unused

Control Flag

Unused

Password or Returned Sequence Number

Unused

Sequence Number

A 16 bit number incremented each time a trap message is sent so that the HM can order the received trap messages and detect missed messages.

TAC TRAP FIELDS

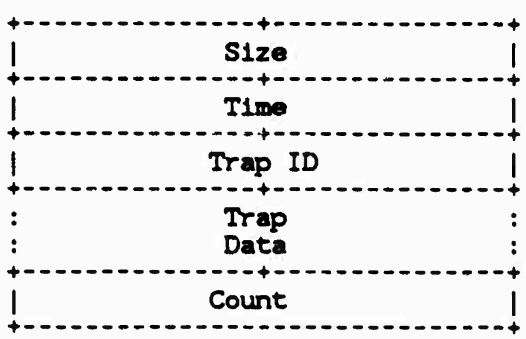
Version #

The version # of the TAC Software.

Trap Reports

There can be several blocks of trap data in each message.

The format of the trap data is as follows:



Size

Size is the number of 16 bit words in the trap, not counting the size field.

Time

The time (in 640ms. units) at which the trap occurred. This field is used to sequence the traps in a message and

associate groups of traps.

Trap ID

This is (usually) the program counter at the trap. The ID identifies the trap, and does not have to be a program counter, provided that it uniquely identifies the trap.

Trap Data

The TAC returns data giving more information about the trap. There are usually two entries: the values in the accumulator and the index register at the occurrence of the trap.

Count

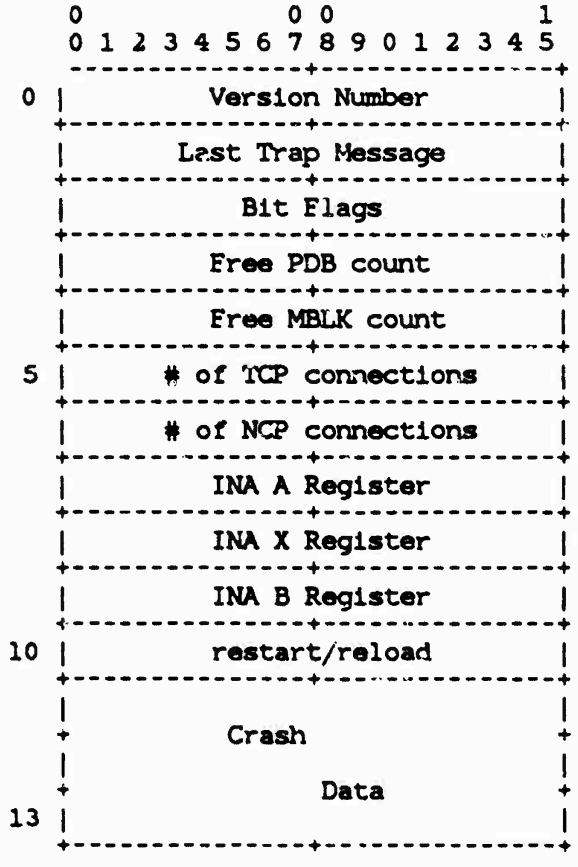
The TAC Counts repetitions of the same trap ID and reports this count here.

B.2 Message Type 2: TAC Status

Description

The status message gives a quick summary of the state of the TAC. Status of the most important features of the TAC are reported as well as the current configuration of the machine.

A TAC status message has the following form:



HMP FIELDS

System Type

TAC = 3

Message Type

TAC Status Message = 2

Port Number

Unused

Control Flag

Unused

Sequence Number

A 16 bit number incremented each time a status message is sent.

Returned Sequence Number

Contains the sequence number from the polling message requesting this report.

TAC STATUS FIELDS

Version Number

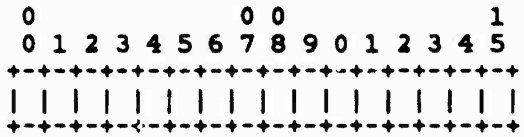
The TAC's software version number.

Last Trap Message

Contains the sequence number of the last trap message sent to the HM. This will allow the HM to detect how many trap messages are being lost.

Bit Flags

There are sixteen bit flags available for reporting the state of various switches (hardware and software) in the TAC. The bits are numbered as follows for purposes of the discussion below.



The bit flags report the status of the following:

Bit	Meaning
15	0 => DDT override off; 1 => override on.
11-14	0 => Sense Switch n is off; 1 => SSn on.
10	0 => Traps to remote monitor; 1 => Traps to console.
9	1 => Message generator on.
0-8	unused

Free PDB count

The number of PDBs on the free queue.

Free MBLK count

The number of MBLKs on the free queue.

# of TCP connections
# of NCP connections

The number of open connections for each protocol.

INA Report

These three fields report the values retained by an INA 1011 instruction in a C/30. This instruction returns micro-machine status and errors. In a #316, the fields are meaningless.

Restart/Reload

This word reports a restart or reload of the TAC

Value	Meaning
1	restarted
2	reloaded

Crash Data

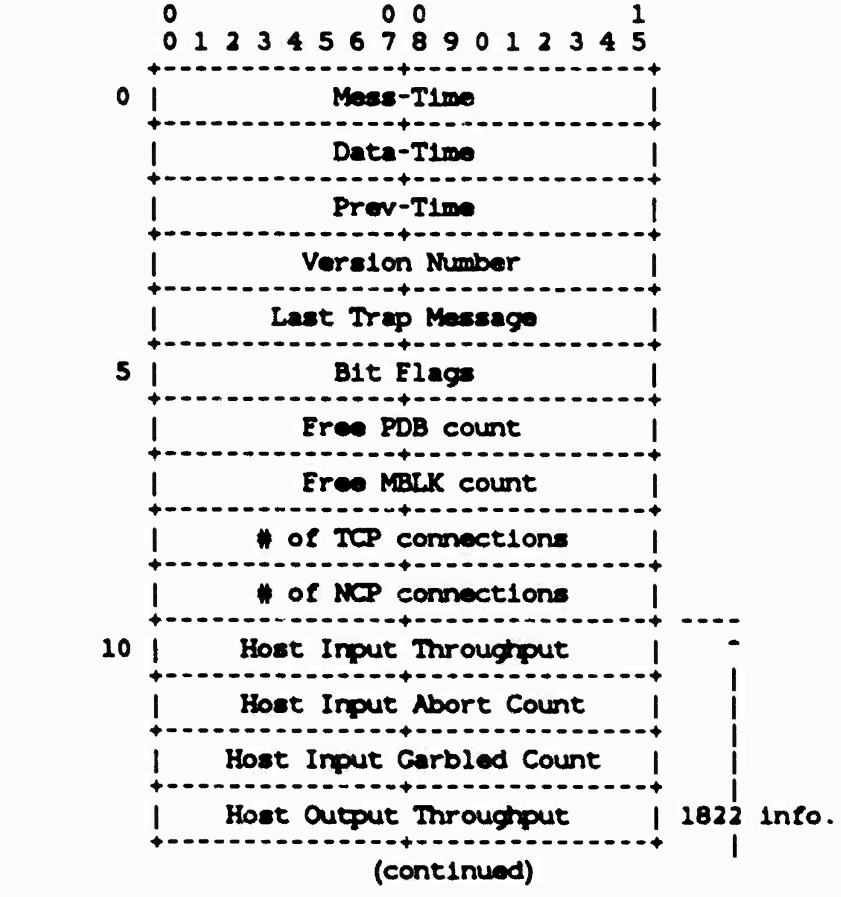
Crash data reports the circumstances surrounding an unexpected crash. The first word reports the location of the crash and the following two are the contents of the accumulator and index registers.

B.3 Message Type 3: TAC Throughput

Description

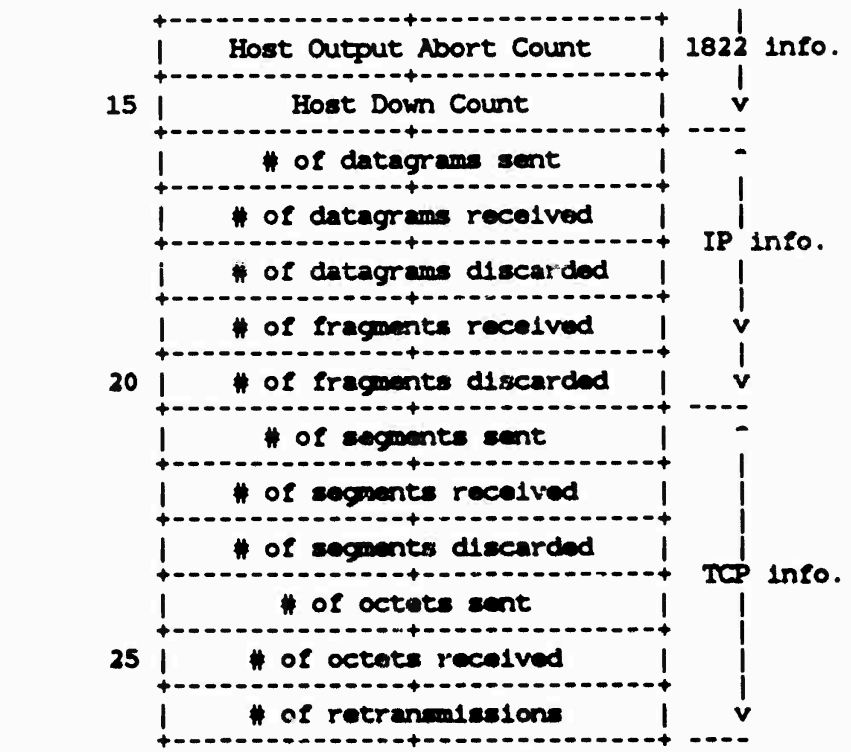
The TAC throughput message reports statistics for the various modules of the TAC. The TAC will collect these data at regular intervals and save them awaiting a poll from the HM. If a period is missed by the HM, the new results simply overwrite the old. Two time stamps bracket the collection interval (data-time and prev-time) and are an indicator of missed reports. In addition, mess-time indicates the time at which the message was sent.

A TAC throughput message has the following form:



(continued)

TAC throughput (cont.)



HMP FIELDS

System Type

TAC = 3

Message Type

TAC Throughput Message = 3

Port Number

Unused

Control Flag

Unused

Sequence Number

A 16 bit number incremented at each collection interval (i.e. when a new throughput message is assembled). The HM will be able to detect lost or duplicate messages by checking the sequence numbers.

Returned Sequence Number

Contains the sequence number from the polling message requesting this report.

TAC THROUGHPUT FIELDS

Mess-time

The time (in 640ms. units) at which the message was sent to the HM.

Data-Time

Data-time is the time (in 640ms. units) when this set of data was collected. (See Description.)

Prev-Time

Prev-time is the time (in 640 ms. units) of the previous collection of data (and therefore, is the time when the data in this message began accumulating.)

Version Number

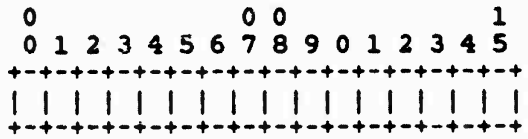
The TAC's software version number.

Last Trap Message

Contains the sequence number of the last trap message sent to the HM. This will allow the HM to detect how many trap messages are being lost.

Bit Flags

There are sixteen bit flags available for reporting the state of various switches (hardware and software) in the TAC. The bits are numbered as follows for purposes of the discussion below.



The bit flags report the status of the following:

Bit	Meaning
15	0 => DDT override off; 1 => override on.
11-14	0 => Sense Switch n is off; 1 => SSn on.
10	0 => Traps to remote monitor; 1 => Traps to console.
9	1 => Message generator on.
0-8	unused

Free PDB count

The number of PDBs on the free queue.

Free MBLK count

The number of MBLKs on the free queue.

# of TCP connections
# of NCP connections

The number of open connections for each protocol.

1822 info.

These six fields report statistics which concern the operation of the 1822 protocol module, i.e. the interface between the TAC and its IMP.

IP info.

These five fields report statistics which concern Internet Protocol in the TAC.

TCP info.

These six fields report statistics which concern TCP protocol in the TAC.

C Appendix C - Gateway Monitoring

C.1 Gateway Parameters

The gateway supports parameters to set Throughput and Host traffic matrix measurements. The type of parameters and the parameter and data pairs are as follows:

Throughput - Type = 3

Parm.	Description	Control Data Word
1	Start/Stop	0=Stop,1=Start
2	Collection Interval	Time in 1 minute ticks

Host Traffic Matrix - Type = 4

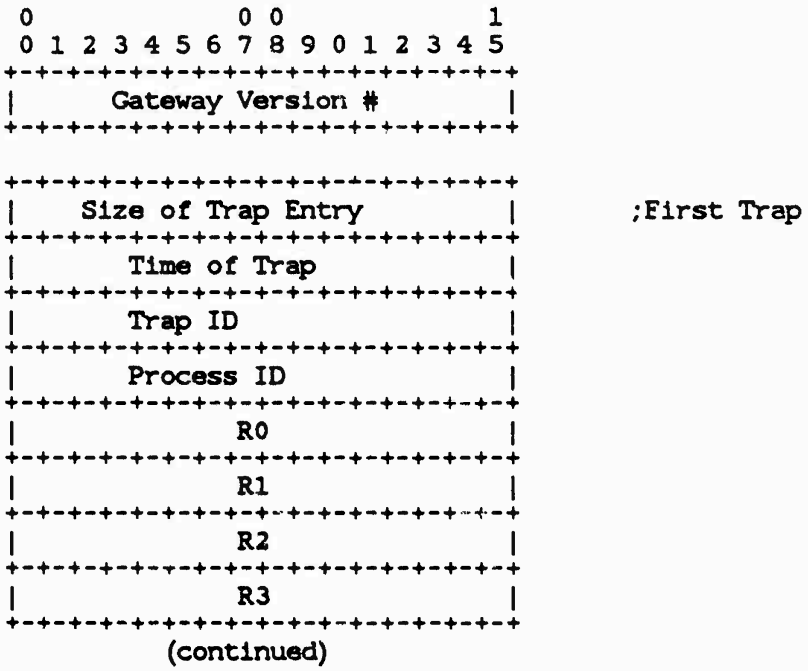
Parm.	Description	Control Data Word
1	Start/Stop	0=Stop,1=Start
2	Collection Interval	Time in 1 minute ticks
3	HTM Switch Control	Include Control Protocols

C.2 Message Type 1: Gateway Trap

Description

When traps occur in the gateway they are buffered. At a fixed time interval (currently 10 seconds) the gateway will send any traps that are in the buffer to the monitoring center. The traps are sent as unsolicited messages.

A Gateway trap message has the following format:



(continued)

GATEWAY TRAP FIELDS

Gateway Version #

The software version number of the gateway sending the trap.

Trap Reports

The remainder of the trap message consists of the trap reports. Each consists of the following fields:

Size of Trap Entry

The size in 16-bit words of the trap entry, not including the size field.

Time of Trap

The time in (in 1/60 sec. ticks) at which the trap occurred.

Trap ID

The number of the trap which is used to identify the trap.

Process ID

The identifier of the process that executed the trap.

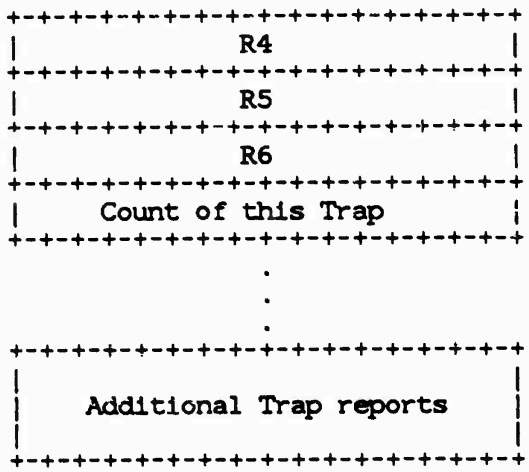
R0-R6

The registers of the machine at the occurrence of the trap.

Count of this Trap

The number of times that this trap occurred.

Gateway Trap Message (cont'd.)



HMP FIELDS

System Type

Gateway = 4

Message Type

Gateway Trap Message = 1

Port Number

Unused

Control Flag

Unused

Password or Returned Sequence Number

Unused

Sequence Number

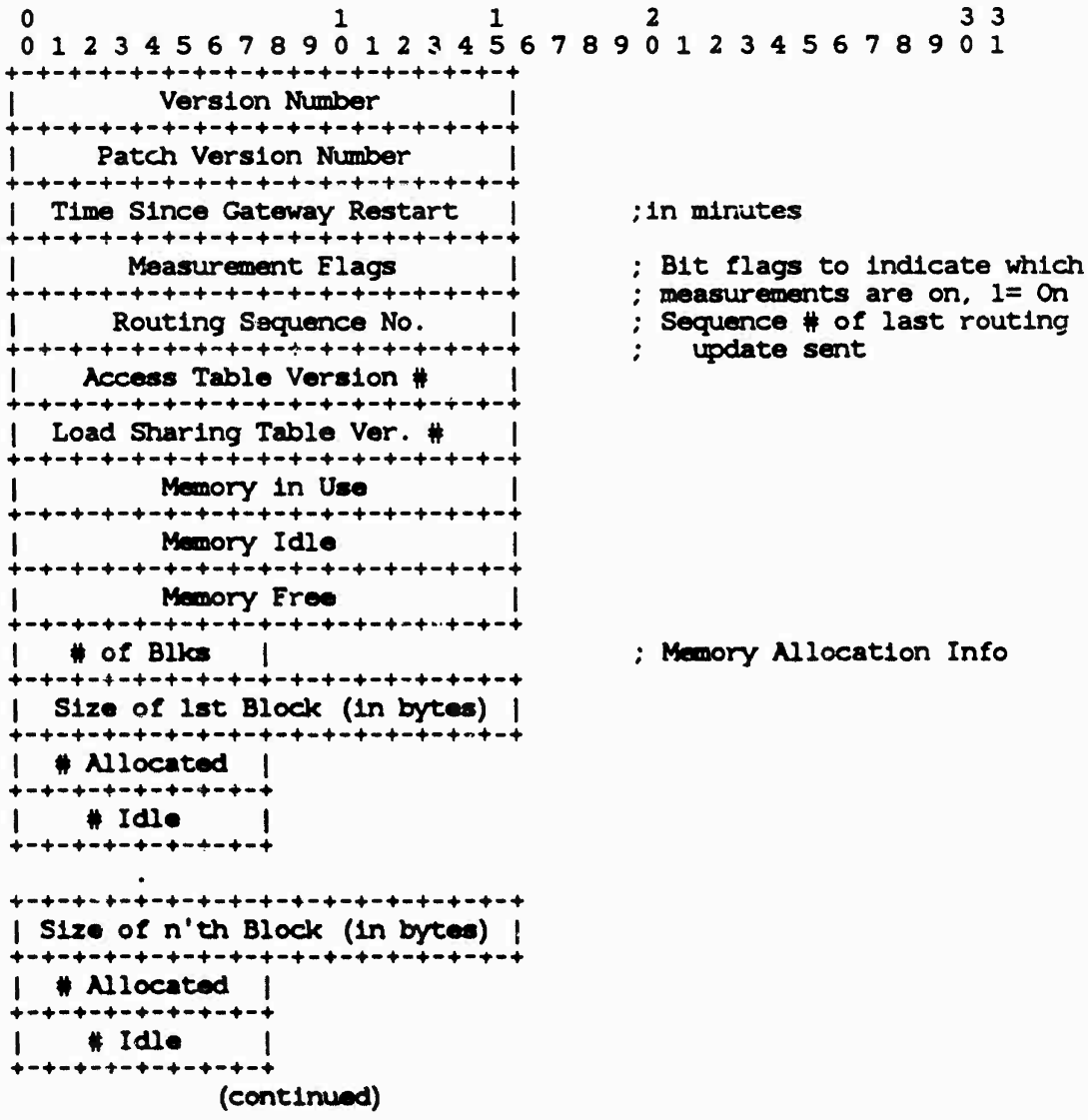
A 16 bit number incremented each time a trap message is sent so that the monitoring center can order the received trap messages and detect missed messages.

C.3 Message Type 2: Gateway Status

Description

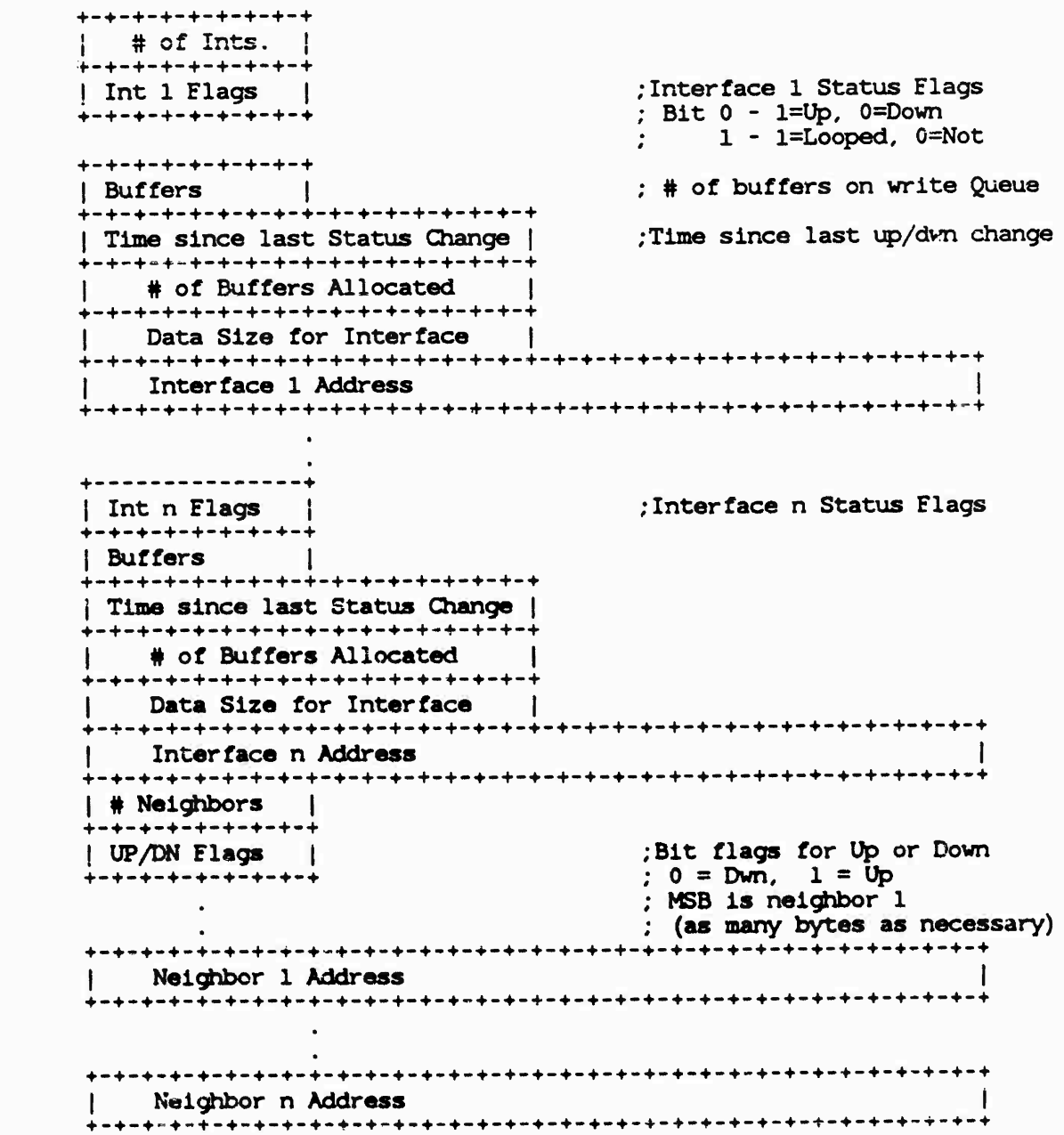
The gateway status message gives a summary of the status of the gateway. It reports information such as version number of the gateway, buffer memory usage, interface status and neighbor gateway status.

A Gateway Status message has the following format:


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Version Number
Patch Version Number
Time Since Gateway Restart
;in minutes
Measurement Flags
; Bit flags to indicate which
; measurements are on, 1= On
Routing Sequence No.
; Sequence # of last routing
; update sent
Access Table Version #
Load Sharing Table Ver. #
Memory in Use
Memory Idle
Memory Free
# of Blks
; Memory Allocation Info
Size of 1st Block (in bytes)
Allocated
Idle
Size of n'th Block (in bytes)
Allocated
Idle

(continued)

Gateway Status Message (cont'd.)



HMP FIELDS

System Type

Gateway = 4

Message Type

Gateway Status Message = 2

Port Number

Unused

Control Flag

Unused

Password or Returned Sequence Number

Unused

Sequence Number

A 16 bit number incremented each time a trap message is sent so that the monitoring center can order the received trap messages and detect missed messages.

GATEWAY STATUS FIELDS

Version Number

The version number of the gateway sending the Status message.

Patch Version Number

The patch version number of the gateway.

Time Since Gateway Restart

The time in minutes since the gateway was last restarted or reloaded.

Measurement Flags

Flags that, if set, indicate which measurements are turned on. Current values are:

```
Bit 0   =   Message Generator
    1   =   Throughput
    2   =   Host Traffic Matrix
    3   =   Access Control 1
    4   =   Access Control 2
    5   =   Load Sharing
    6   =   EGP in Gateway
```

Routing Sequence Number

The sequence number of the last routing update sent by this gateway.

Access Control Table Version #

The version number of the access control table.

Load Sharing Table Version #

The version number of the load sharing table.

Memory In Use

The number of bytes of buffer memory that are currently in use.

Memory Idle

The number of bytes of buffer memory that have been allocated but are currently idle.

Memory Free

The number of bytes of buffer memory that has not been allocated.

MEMORY ALLOCATION INFORMATION

The next part of the status message contains information on the buffer pools in the gateway. The fields are:

# of Blocks

The number of different buffer pools.

Size of Block

The size of this block in bytes.

Allocated

The number of blocks of this size that have been allocated.

Idle

The number of blocks of this size that are idle.

GATEWAY INTERFACE FIELDS

The next part of the status message are fields that provide information about the gateway's interfaces. The fields are:

# of Interfaces

The number of network interfaces that the gateway has.

Interface Flags

Flags that indicate the status of this interface. The current values are:

```
Bit 0  -  1=Up/0=Down
    1  -  1=Looped/0=Not Looped
```

Buffers

The numbers on this interfaces write queue.

Time Since Last Status Change

The time in minutes since this interface changed status (Up/Down).

# of Buffers Allocated

The number of buffers allocated for this interface.

Data Size for Interface

The buffer size required for this interface.

Interface Address

The Internet address of this interface.

NEIGHBOR GATEWAY FIELDS

The final part of the status message consists of information about this gateway's neighbor gateways. The fields are:

# of Neighbors

The number of gateways that are neighbor gateways to this gateway.

UP/DN Flags

Bit flags to indicate if the neighbor is up or down.

Neighbor Address

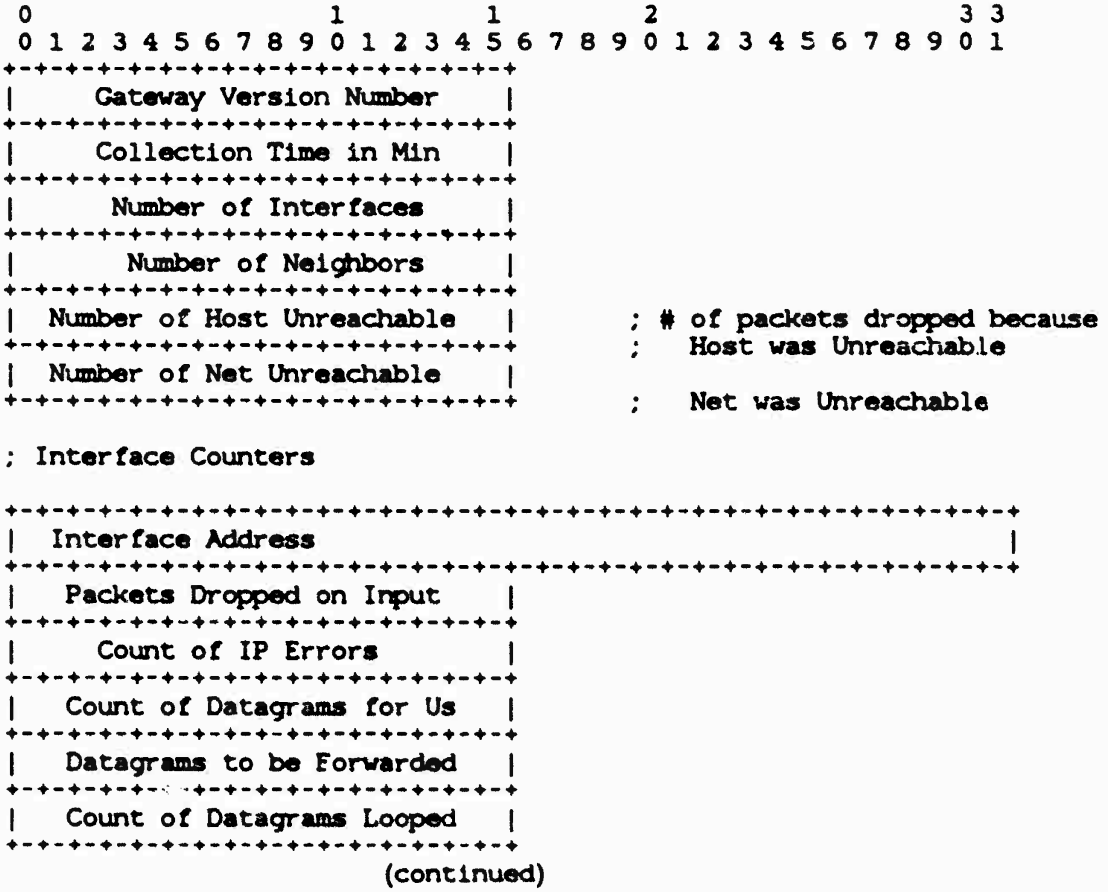
The Internet address of the neighbor gateway.

C.4 Message Type 3: Gateway Throughput

Description

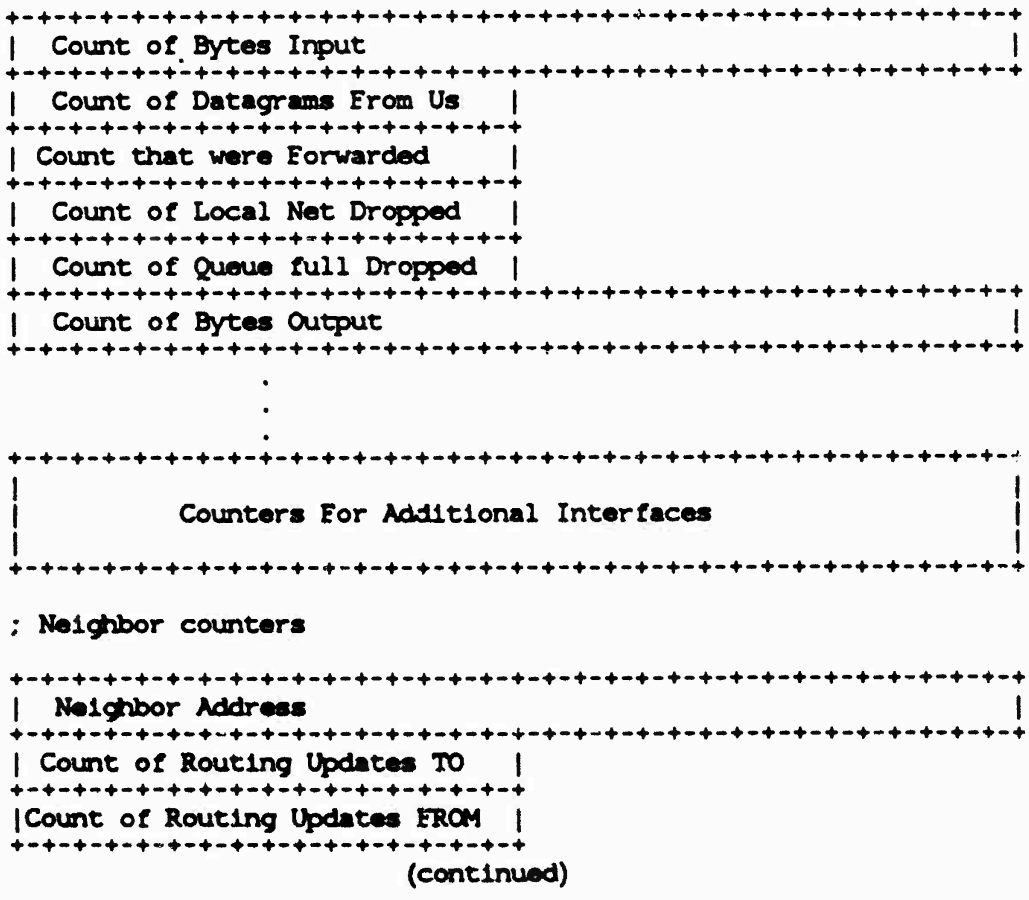
The gateway collects throughput statistics for the gateway, its interfaces, and its neighbor gateways. It collects them for regular intervals and will save them for collection via a Poll message from the Monitoring host. If they are not collected by the end of the next interval, they will be lost because another copy will be put into the saved area.

A Gateway Throughput message has the following format:



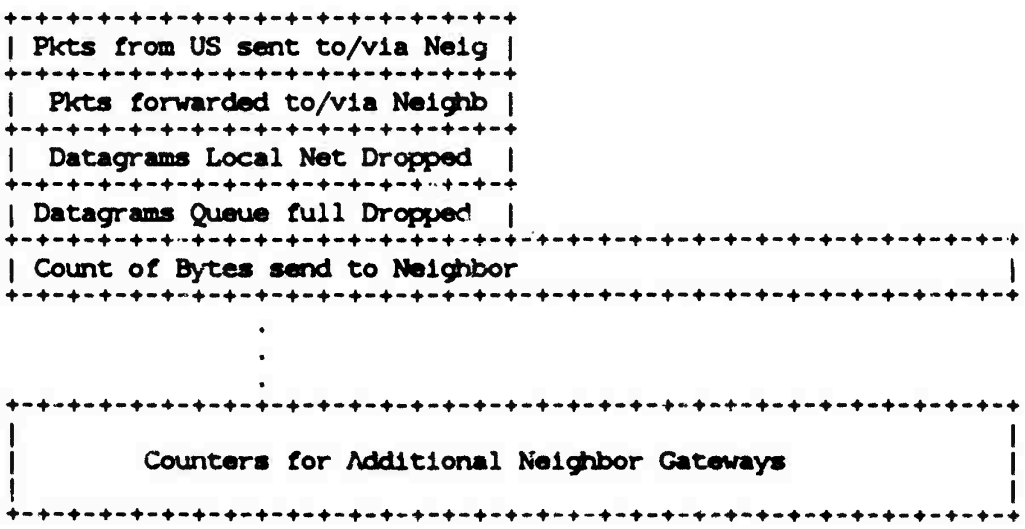
(continued)

Gateway Throughput Message (cont'd.)



(continued)

Gateway Throughput Message (cont'd.)



HMP FIELDS

System Type

Gateway = 4

Message Type

Gateway Throughput Message = 3

Port Number

Unused

Control Flag

Unused

Password or Returned Sequence Number

Unused

Sequence Number

A 16 bit number incremented each time a trap message is sent so that the HM can order the received trap messages and

detect missed messages.

GATEWAY THROUGHPUT FIELDS

Gateway Version Number

The software version number of the gateway sending the trap.

Collection Time in Min.

The time period in minutes in which the throughput data is to be collected.

Number of Interfaces

The number of interfaces this gateway has.

Number of Neighbors

The number of neighbor gateways this gateway has.

Number of Host Unreachable

The number of packets dropped because the Host was unreachable.

Number of Net Unreachable

The number of packets dropped because the Network was unreachable.

INTERFACE COUNTERS

The next part of the Throughput message contains counters for the gateways interfaces. Each interface has the following fields:

Interface Address

The Internet address of this interface.

Packets Dropped on Input

The number of packets on input to this interface because there were not enough buffers.

Count of IP Errors

The number of packets received with bad IP headers.

Count of Datagrams for Us

The number of datagrams received addressed to this gateway.

Datagrams to be Forwarded

The number of datagrams were not for this gateway and should be sent out another interface.

Count of Datagrams Looped

The number of datagrams that were received on and sent out of this interface.

Count of Bytes Input

The number of bytes received on this interface.

Count of Datagrams From Us

The number of datagrams that originated at this gateway.

Count that were Forwarded

The number of datagrams that were forwarded to another gateway.

Count of Local Net Dropped

The number of packets that were dropped because of local network flow control restrictions.

Count of Queue full Dropped

The number of packets that were dropped because the output queue was full.

Count of Bytes Output

The number of bytes sent out on this interface.

NEIGHBOR COUNTERS

The last part of the Throughput message are counts for each neighbor gateway. The fields are:

Neighbor Address

The Internet address of this neighbor gateway.

Count of Routing Updates TO

The number of routing updates sent to this neighbor gateway.

Count of Routing Updates FROM

The number of routing updates received from this neighbor gateway.

Pkts from US sent to/via Neig

The number of packets from this gateway sent to or via this neighbor gateway.

Pkts forwarded to/via Neighb

The number of packets forwarded to or via this neighbor gateway.

Datagrams Local Net Dropped

The number of datagrams dropped to this neighbor gateway because of local network flow control restrictions.

Datagrams Queue full Dropped

The number of datagrams dropped to this neighbor because the output queue was full.

Count of Bytes send to Neighbor

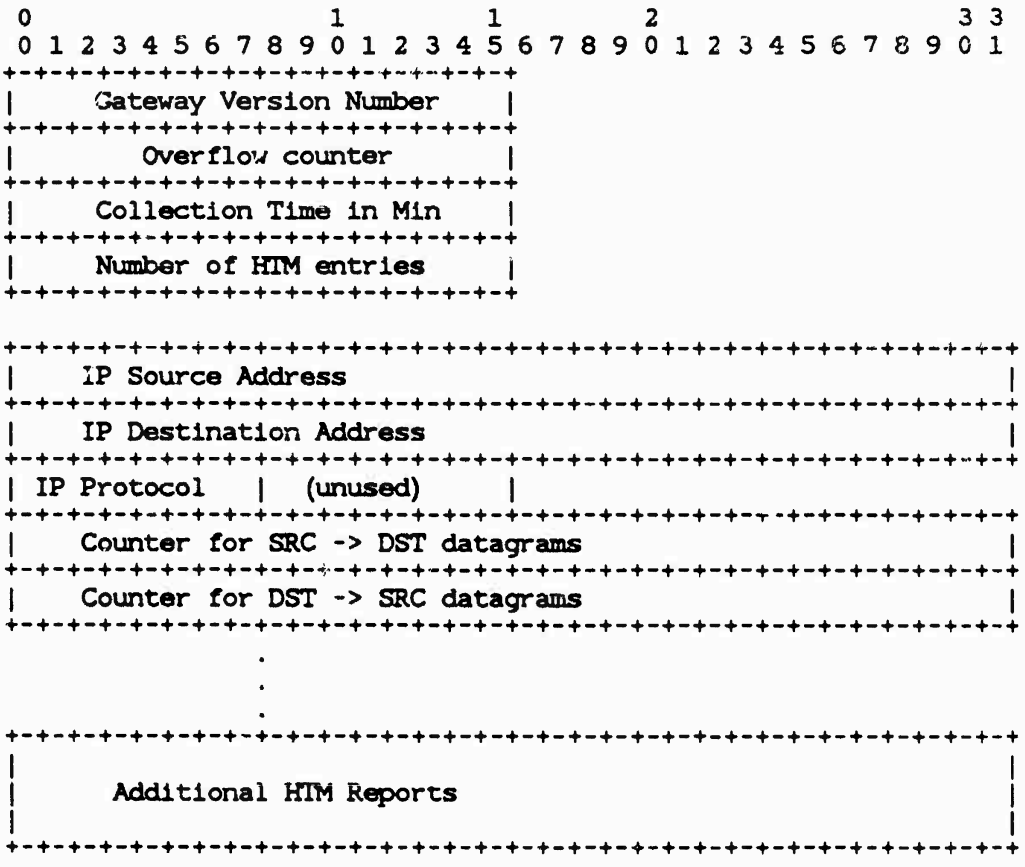
The number of bytes sent to this neighbor gateway.

C.5 Message Type 4: Gateway Host Traffic Matrix

Description

The Host Traffic Matrix (HTM) message contains information about the traffic that flows through the gateway. Each entry consists of the number of datagrams sent and received for a particular source/destination pair.

A Gateway HTM message has the following format:



HMP FIELDS

System Type

Gateway = 4

Message Type

Gateway HTM Message = 4

Port Number

Unused

Control Flag

Unused

Password or Returned Sequence Number

Unused

Sequence Number

A 16 bit number incremented each time a trap message is sent so that the HM can order the received trap messages and detect missed messages.

GATEWAY HTM FIELDS

Gateway Version Number

The software version number of this gateway.

Overflow counter

The number of HTM entries lost because the HTM buffer was full.

Collection Time in Min

The time period in minutes in which the HTM data is being collected.

Number of HTM entries

The number of HTM reports included in this message.

HTM ENTRIES

The remainder of the HTM message consists of the actual HTM entries. Each entry consists of the following fields:

IP Source Address

The source Internet address of the datagrams being counted.

IP Destination Address

The destination Internet address of the datagrams being counted.

IP Protocol

The protocol number of the datagrams.

Counter for SRC -> DST datagrams

The number of datagrams sent in the Source to Destination address direction.

Counter for DST -> SRC datagrams

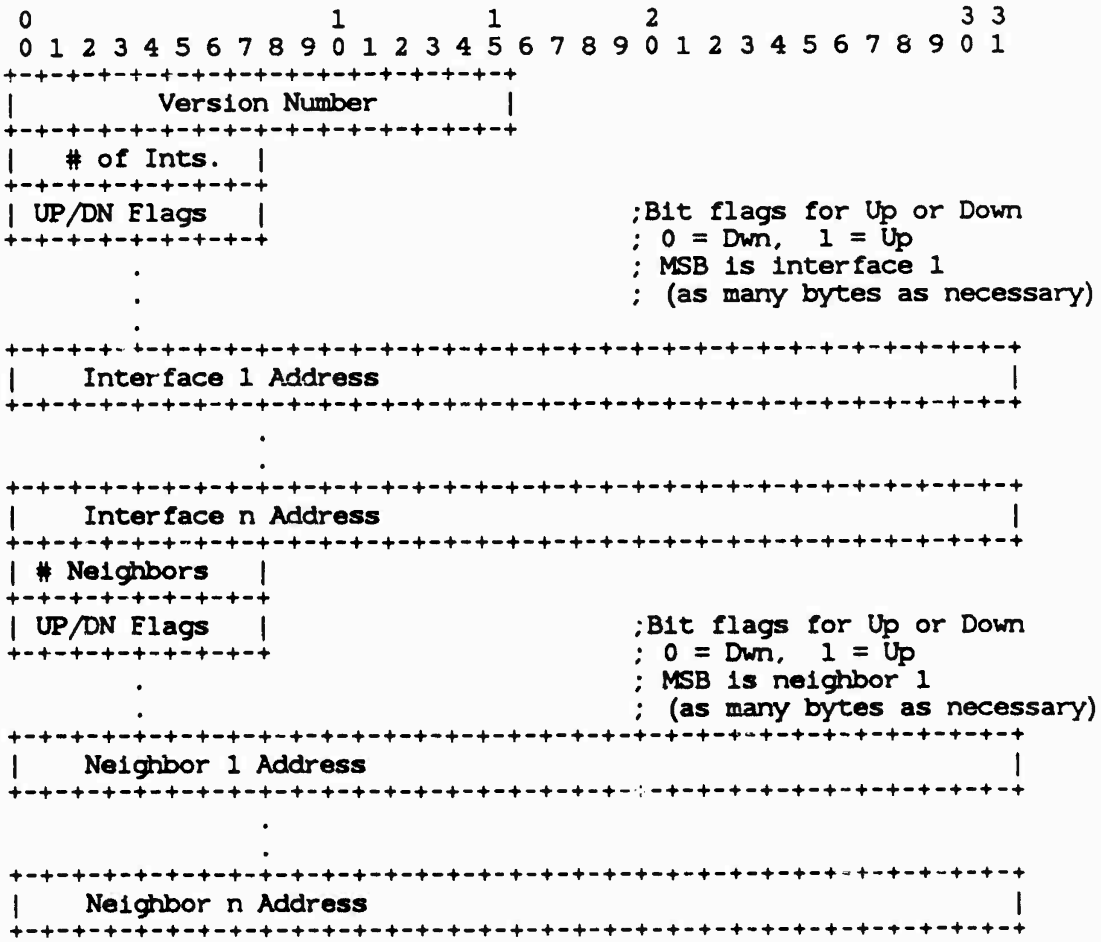
The number of datagrams sent in the Destination to Source address direction.

C.6 Message Type 6: Gateway Routing

Description

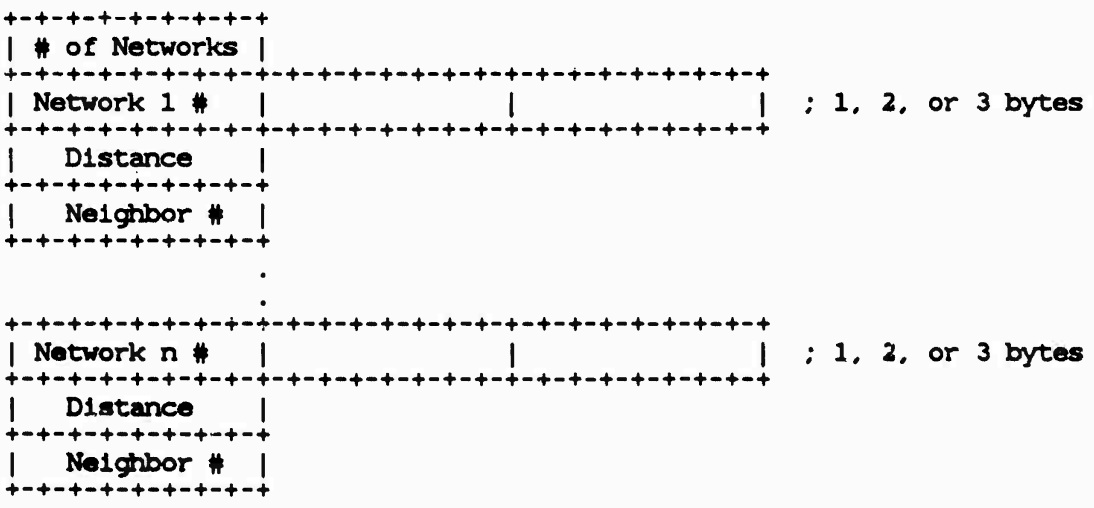
The Routing message contains information about routes the gateway has to the networks that make up the Internet. It includes information about its interfaces and its neighbor gateways.

A Gateway Routing message has the following format:



(continued)

Gateway Routing Message (cont'd.)



HMP FIELDS

System Type

Gateway = 4

Message Type

Gateway Trap Message = 6

Port Number

Unused

Control Flag

Unused

Password or Returned Sequence Number

Unused

Sequence Number

A 16 bit number incremented each time a trap message is sent so that the HM can order the received trap messages and detect missed messages.

GATEWAY ROUTING FIELDS

Gateway Version #

The software version number of the gateway sending the trap.

INTERFACE FIELDS

The first part of the routing message contains information about the gateway's interfaces. There is data for each interface. The fields are:

# of Interfaces

The number of interfaces that this gateway has.

UP/DN Flags

Bit flags to indicate if the Interface is up or down.

Interface Address

The Internet address of the Interface.

NEIGHBOR FIELDS

The next part of the routing message contains information about this gateway's neighbor gateways. The fields are:

# of Neighbors

The number of gateways that are neighbor gateways to this gateway.

UP/DN Flags

Bit flags to indicate if the neighbor is up or down.

Neighbor Address

The Internet address of the neighbor gateway.

NETWORK ROUTING FIELDS

The last part of the routing message contains information about this gateway's routes to other networks. This includes the distance to each network and which neighbor gateway is the route to the network. The fields are:

# of Networks

The number of networks that are reachable from this gateway.

Network #

The network number of this network. This is the network part of the Internet address and may be one, two, or three bytes in length depending on whether it is a Class A, B, or C address.

Distance

The distance in hops to this network. Zero hops means that the network is directly connected to this gateway. A negative number means that the network is currently unreachable.

Neighbor #

The neighbor gateway that is the next hop to reach this network. This is an index into the previous information on this gateway's neighbor gateways. This field is only valid if the Distance is greater than zero.

IEN 158

Jack Haverty
Bolt Beranek and Newman
1 October 1980

XNET Formats for Internet Protocol Version 4

Jack Haverty
Bolt Beranek and Newman Inc.
October 1,1980

This IEN is intended to capture in print the formats used currently in the version 4 XNET protocol; most of the data is courtesy of Ray Tomlinson.

Version 4 XNET is identical with version 2.5 XNET with the exceptions listed below. The version 2.5 format is described in RFC 643. It should be noted that the manner in which the protocol is used by a user program (such as the PDP10 XNET program), and by the various target-machine XNET servers, is not defined herein. In particular there are several problems and heuristics in dealing with the operation of the protocol in the internet environment, where individual packets may be duplicated, lost, and reordered.

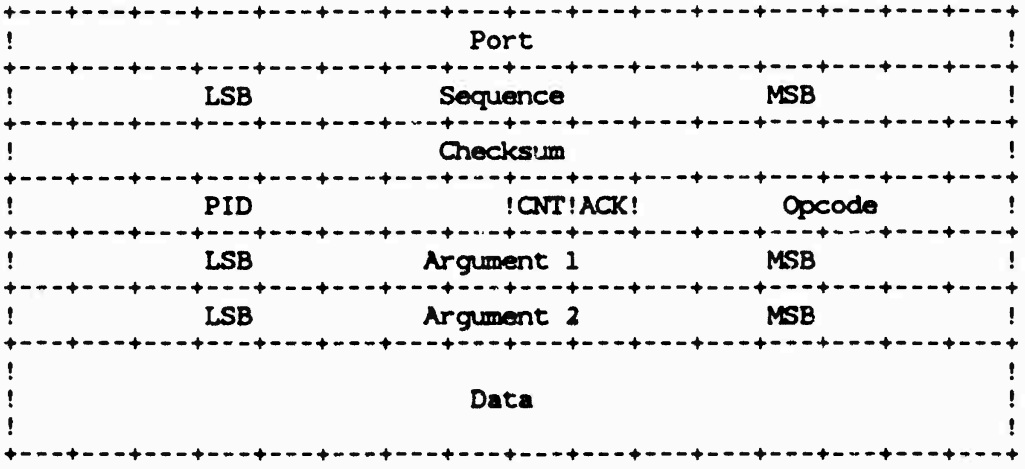
Changes from the version 2 formats include the following:

1) XNET header and data is embedded in a IN V4 packet instead of a V2 5 packet.
2) Packet format changed to add Port, Sequence number, and Checksum fields.
3) Change in asynchronous reply codes.
4) Addition of ACK bit to opcode field.
5) Positive acknowledgement of all messages.

IEN 158

Jack Haverty
Bolt Beranek and Newman
1 October 1980

Packet Format



The IN protocol is set to the XNET protocol number (17 octal).

Host to target opcodes

NOP	0	No operation.
DEBUG	1	Start debugging a process or address space.
ENDBUG	2	End debugging a process or address space.
HALT	3	Halt the process.
DPOSIT	4	Deposit in memory.
RESUME	5	Resume execution of a process.
EXAM	6	Examine memory.
DSV	7	Deposit state vector (r0-r5,sp,pc,ps).
SETBPT	10	Set breakpoint.
REMBPT	11	Remove breakpoint.
ONESTP	12	Single step process (using trace trap).
PROCD	13	Proceed from breakpoint.
CREAP	14	Create a new process (or address space).
DSTROY	15	Destroy (delete) a process or address space.
XIOREP	16	Reply to XIO output (not used anymore).
XINREP	17	Reply to XIO input.
DEFALL	20	Define and allocate memory to an address space.
SAP	21	Start all processes.
SAVDSK	22	Save on disk.
GETDSK	23	Get from disk.
ENTRST	24	Enter address space into restart table.

IEN 158

Jack Haverty
Bolt Beranek and Newman
1 October 1980

Opcodes from target to host machine.

HALTED	77	Process halted (FREEP with arguments of 0).
TRAPPED	76	Process trapped due to error.
TTRAP	75	Trace trap.
BPT	74	Breakpoint hit.
XIOIN	73	XIO input request.
XIOOUT	72	XIO output request.

Checksum

The checksum is the same as that for the IN header; ones complement of ones complement sum of words in the packet from Port field to last data word inclusive. In case of an odd number of data bytes, an additional byte of zeroes is assumed for checksum purposes.

Port number

The port number is a unique number relative to the host machine which appears in every packet for a particular debugging session. It is suggested that this number be derived from the time of day so that each session will be unique over a long period of time.

Sequence number

The first packet of a session (first use of a particular port number) is numbered 0. Subsequent packets increment by 1 modulo 2**16. Packets initiated by the target machine (opcodes

Jack Haverty
Bolt Beranek and Newman
1 October 1980

72-77) are also numbered starting from 0. The target machine is allowed to execute packets out of order but must never execute a packet twice unless the effect is harmless. For example, an examine packet should be re-executed so that the data may be returned to the sender. Deposit or resume should not be re-executed. The host machine is responsible for correct ordering of critical functions. For example, it must not send a RESUME command until all prior deposits have been acknowledged.

Acknowledgements

Each packet must be acknowledged by the receiver. An acknowledgement consists of the original header plus any requested data (e.g. EXAM) with the ACK bit set. Acknowledgements are not cumulative; an acknowledgement acknowledges only the one packet with the matching sequence number. If the target debugger is incapable of performing the requested function, it should set the CNT (can't) bit instead of the ACK bit. Both bits may be set meaning that the function is available but the data required is no longer available. This might be the result of a duplicate packet.

2 May 1979

IEN 90

Multiplexing Protocol

Danny Cohen

Jon Postel

2 May 1979

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

(213) 822-1511

IEN-90

D. Cohen
J. Postel
ISI
2 May 1979

Multiplexing Protocol

Introduction

This Multiplexing Protocol is defined to allow the combining of transmission units of different higher level protocols in one transmission unit of a lower level protocol. Only messages with the same Internet Protocol (IN) [1] header, with the possible exception of the protocol field may be combined. For example, the msg (H1, B1) and the message (H2, B2), where Hi and Bi are the headers and the bodies of the messages, respectively, may be combined (multiplexed) only if H=H1=H2. The combined messages are either (H, B1, B2) or (H, B2, B1).

Since (H,D1)+(H,D2)=(H,D1+D2) resembles the notion of factoring, we sometime refer to this process as "factoring".

The receiver of this combined message should treat it as if the two original messages, (H,D1), and (H,D2), arrived separately, in either order.

The multiplexing is achieved by combining the individual messages, (H,B1) through (H,Bn), into a single message. This single message has an IN header which is equal to H, but having in the PROTOCOL field the value 18 which is the protocol number of the multiplexing protocol. This IN header is followed by all the message bodies, B1 through Bn. Each message body, Bi, is preceeded by a 4 octet multiplexing link. This link contain the number of the protocol to which this body is addressed. It also contain the total length of this portion (message body), including this multiplexing link. Since this link is not otherwise protected by a checksum, it also includes a checksum field which covers this multiplexing link.

If an error is discovered in a checksum of some multiplexing header, the rest of the message, starting there, is ignored.

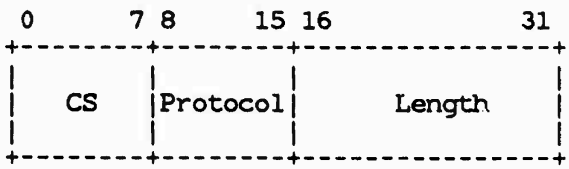
If an unknown PROTOCOL field is discovered in any multiplexing header, this section, and only this one, is ignored.

Multiplexing Protocol
Introduction

The demultiplexing routine should be able to handle recursively multiplexed messages. This is to allow higher level protocol to demultiplex their own messages if they can be combined. Since such a multiplexed message may be multiplexed again by the IN level, a multi-level multiplexing results.

This protocol assumes that the Internet Protocol is used as the underlying protocol.

Format



Multiplexing Header Format

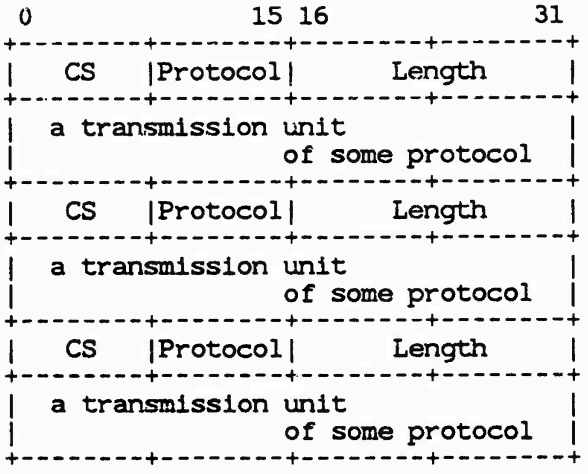
Fields

CS is a checksum covering only this 32 bit multiplexing header. Until further notice, it is the exclusive OR of the other three octets in this header.

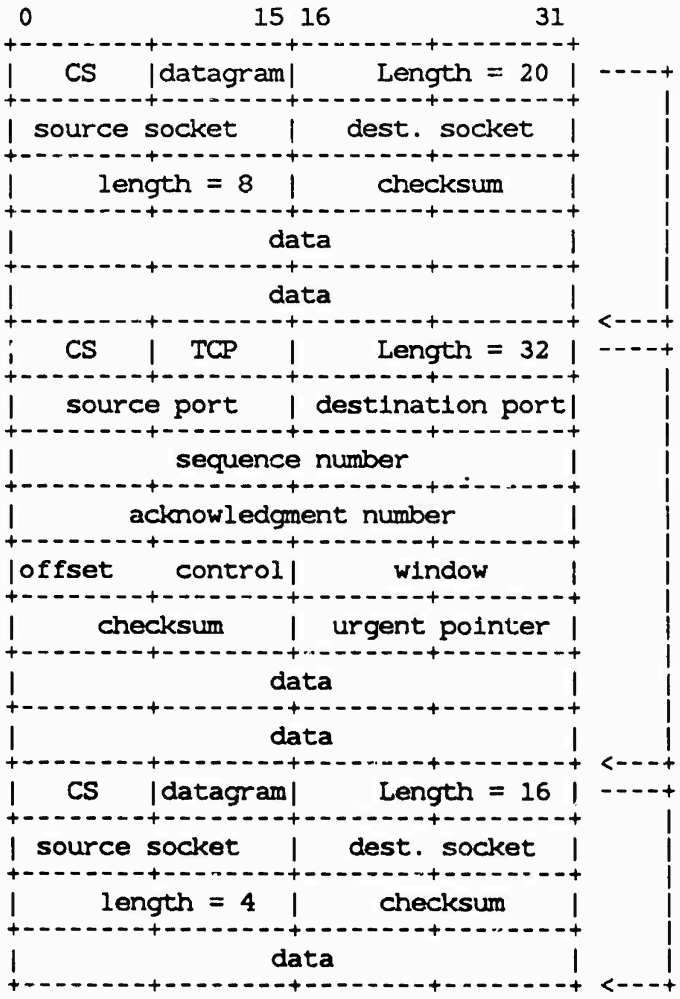
Protocol is the number of the following protocol.

Length is the length in octets of this header and the following protocol block. Hence, it must be at least 4.

Example



Multiplexing Protocol Concept



Multiplexing Protocol Example

Protocol Application

The major use of this protocol is to allow several transmission units from differing (or the same) higher level protocols to be combined into one transmission unit of a lower level protocol.

Protocol Number

This is protocol 18 (22 octal) when used in the Internet Protocol. Other protocol numbers are listed in [2].

Notes

- If so desired, one has the option of applying this multiplexing protocol recursively.

- The receiving process should never be able to tell if its messages were multiplexed or not. The multiplexing is totally transparent to the higher lever protocols.

- Information from the external header (e.g., the IN header) is available to each protocol in the multiplexed message.

References
---

[1] Postel, J., "Internet Datagram Protocol -- Version 4," IEN-80, USC-Information Sciences Institute, February 1979.

[2] Postel, J., "Assigned Numbers," USC-Information Sciences Institute, RFC-755, IEN 93, May 1979.

IEN 119

ST - A Proposed Internet Stream Protocol

by

James W. Forgie

M. I. T. Lincoln Laboratory

7 September 1979

IEN 119 ST.DOC 7 September 1979

1.0 INTRODUCTION

The internet stream protocol (ST) described in this document has been developed to support efficient delivery of streams of packets to either single or multiple destinations in applications requiring guaranteed data rates and controlled delay characteristics. The principal applications with these requirements are point-to-point speech communication and voice conferencing. While ST has been developed as a part of the ARPA Internet Program and has been formulated as an extension to the presently defined Internet Protocol (IP), it is not likely to find useful application in the current ARPA internet environment where the networks and gateways lack the capacity to handle significant speech communication. Instead, ST is aimed at application in wideband networks, in particular those intended to carry a large fraction of packet voice in their traffic mixes. Work is currently underway on such networks both for local access and long haul use. These networks will serve as vehicles for research on techniques for flow and traffic control and as testbeds for evaluating the potential of packet technology for providing economical speech communication. The design of the ST protocol represents a compromise among the sometimes conflicting requirements of compatibility with the existing IP and the gateways which handle it, the need for flexibility in supporting flow and traffic control research, and transmission efficiency.

The concepts in this protocol originated in the deliberations of a working group consisting of Danny Cohen, Estil Hoversten, and the author. They have been influenced by interactions with many other people. In order to examine the cost and feasibility of the protocol, the author has fleshed out some aspects of the protocol in detail. The other working group participants have not had an opportunity to approve or modify these detailed aspects of the protocol, and consequently all responsibility for them lies with the author.

The state of the protocol is such that, while there are still details to be worked out, implementation could begin if the protocol were acceptable to those interested.

2.0 MOTIVATION AND GENERAL DESCRIPTION

It is reasonable to ask why a new protocol is required to handle applications such as point-to-point speech and voice conferencing. This section addresses that question and begins with a brief statement of the requirements for packet speech communication. They are:

1. The network must be able to keep up with the data rate requirements of the speech terminal. Because no bits need be transmitted during silent intervals, the average data rate for conversational speech can be expected to be between 40 and 50% of the peak data rate for commonly used constant-rate encoding techniques. Experimental variable-rate encoding techniques have exhibited higher peak-to-average ratios. The network must be able to sustain the peak rate for the duration of talkspurt that can be as long as 20 seconds.

2. The stream of packets containing a talkspurt (a continuous segment of speech between silent intervals) must be delivered with a delay dispersion whose spread does not exceed some value that can be estimated with a high probability of success prior to the start of the talkspurt. Since the individual packets of the spurt will experience different delays as they pass through the net, delay must be added at the receiver to allow continuous speech to be played out for the listener. It is necessary to predict the value of this smoothing delay before starting to play out the talkspurt. Packets that are delayed more than the predicted worst-case value will arrive too late to be used, and gaps will occur in the output speech.

3. Overall delay should be kept low. If the overall round-trip delay is less than about 1/4 second, conversations are carried out in a "normal" fashion with considerable feedback from "listener" to "talker" taking place. When greater delay is experienced, people switch to a more formal mode in which feedback utterances are mostly suppressed, and the listener generally waits until the talker indicates that he has finished before saying anything. User satisfaction declines with increasing delay, but systems remain usable for delays as long as several seconds.

4. The amount of speech for any one talker contained in a packet (the basic unit subject to transmission loss) should be kept small. The loss of small (50 msec or less) chunks of speech produces a degradation of quality, but sentence intelligibility tends to be

preserved up to fairly high percentage losses. Larger chunks of speech represent whole syllables or words, and their loss can change the meaning of sentences.

5. Listeners will tolerate some packet loss without downgrading the acceptability of the system, but the probability of loss due to either failed or late delivery must be kept in the order of 1% or less to be considered acceptable for everyday (non-crisis) use.

6. As a network approaches its load limit it should reject (or hold off) new offered load on a call basis not on an individual packet basis. Continuing to accept new calls beyond capacity can result in unsatisfactory communication for many users.

7. If packet-switched speech transmission is to become economically competitive with circuit-switched transmission, a further requirement must be met. The product of packet efficiency and average link utilization must equal or exceed the efficiency of circuit switching. That efficiency is defined as one minus the fraction of the time that silence occurs in conversational situations. Estimates of this fraction for real-world conversations give values for efficiency between 40 and 50%. We will use 45% as a convenient figure for comparative purposes. Packet switching can easily take advantage of the silent intervals in a conversation by not transmitting packets, but that advantage may be lost through the combination of overhead bits in packet headers (packet efficiency) and the difficulty of operating communication links at high average utilization while keeping queueing delays within reasonable bounds.

Conventional datagram networks are unsatisfactory for speech communication except under conditions of light overall load or where speech constitutes a small fraction of the overall load and can be given priority service. The difficulty with datagram nets comes from their inability to provide the controlled delay and guaranteed data rate required for speech. Delay increases with offered load, slowly at light load, but dramatically as average load approaches capacity. Flow control strategies tend to be aimed at buffer management and fairness goals, both of which will operate to restrict the effective data rate available to an individual user as load increases. Traffic control strategies are mainly concerned with congestion control and are primarily defensive, resulting in offered datagrams being held off or refused when difficulties are detected. Unfortunately for the speech user, by the time congestion is detected, it is already too late. For satisfactory speech

service, congestion due to overload must be prevented. Since a datagram net has no knowledge of the a priori requirements of users, it cannot develop traffic control strategies to meet these requirements.

Another disadvantage of datagrams for speech is their packet efficiency. The speech content of an individual user packet can be anything from 50 or so bits up to 1200 or 1300 bits depending upon the speech digitization technique in use. The need to carry full source and destination addresses as well as other packet-handling information in each packet penalizes datagrams relative to other packet and circuit switching techniques. In the internet case the penalty is worse since two sets of header information have to be carried.

For example, IP datagrams on SATNET carrying 40-msec chunks of 16-kbps speech (a reasonaable chunk size and popular data rate) would have a packet efficiency of about 56% and would require utilization factors of about 80% to break even with respect to circuit switching. It is unlikely that delay characteristics would be satisfactory at this level of load.

The goal of the ST design effort has been to attempt to overcome both of the difficulties associated with datagrams. ST uses abbreviated internet headers and also allows speech from many talkers to be aggregated into single packets for transmission on wide-band networks where such aggregation is possible. For the case of ST messages on a wide-band SATNET each carrying ten 40-msec chunks of 16-kbps speech for ten different talkers, packet efficiency would be about 86% allowing break-even link utilization to occur at 52%, a much more comfortable level for assuring desirable delay characteristics.

Overcoming the inability of datagram nets to maintain data rates and delay characteristics as offered load increases is more difficult to achieve than improving packet efficiency. Circuit switching solves the problem by dedicating communication capacity to individual streams. The goal of ST is to support traffic control policies that match stated user requirements to available resources taking into account the statistical properties of these requirements rather than the peak requirements used in circuit switching. ST does not itself specify the traffic control algorithms to be used. The development of such algorithms is an area of research that the protocol is intended to support. Some algorithms may need only rough statistical knowledge of user requirements and network behavior. Others may want more detailed knowledge and need to monitor the behavior of individual streams. The protocol is intended to be general enough to support both extremes. A successful traffic control algorithm would retain much of the statistical multiplexing advantages of datagram nets while at the

same time gaining much of the guaranteed data rate and controlled delay capabilities of circuit switched nets. A packet net using ST also has the ability of a circuit switched network to deny access to, or negotiate lower rates with, users whose demands would exceed capability.

The ST protocol requires users to state the data rate and delay requirements for a data stream before accepting any stream data. These requirements are used by ST agents (host processes and gateways) to determine whether or not resources are available in the catenet to support the offered stream load. The determination is based on knowledge of the stated requirements of other users, negotiation with networks such as SATNET which have built-in resource allocation mechanisms, and statistical load estimates of traffic on networks that lack such mechanisms. In order to accept the offered stream load, the cooperating agents must find a route through networks with sufficient uncommitted capacity to handle the new stream. In the process of routing the stream, intermediate agents retain information about the stream. The existence of this information allows the use of abbreviated headers on stream data packets and the efficient distribution of the multi-addressed packets required for conferencing.

The process used by ST agents in finding a route with sufficient capacity between source and destination is assumed to use a distributed routing algorithm to take advantage of the robustness and flexibility characteristic of distributed packet routing techniques. In the simplest case, the result would be a fixed-path internet route (a fixed set of intermediate agents (gateways)) for the stream packets. In the event of gateway or network failure, rerouting would be required. This can be undertaken automatically, but success is not guaranteed, since loss of the failed element or elements is likely to result in inadequate capacity to carry the original load. Fixed-path routing is not required by the protocol. If desired, dynamic alternate routing of stream packets can be used at the discretion of individual agents, but gateway implementation and the routing process will be more complex if that option is chosen. The protocol described in this document assumes fixed-path routing.

The goal toward which the cooperating ST agents in a catenet work is the maintenance of a controlled delay, guaranteed data rate environment in which packet speech communications can take place in a satisfactory fashion. Obviously, other non-cooperating users of the networks involved in the catenet can make it impossible to achieve that goal. Some independence of other users can be obtained in some networks such as SATNET by the use of dedicated resources. Gateways can be programmed to throttle non-cooperating internet traffic. To some extent, networks with poor delay characteristics can be avoided in the routing process. Priority service can be used in some nets to

allow small quantities (proportionately) of ST traffic to be handled satisfactorily in spite of the activities of other users. However, the success of ST or any other approach to handling packet speech will require either the cooperation of all network users or the involvement of the networks themselves in providing the required controlled delay, guaranteed data rate services.

3.0 RELATIONSHIP TO IP

ST is intended to operate as an extension of the presently defined internet protocol (IP). ST provides a different kind of service than the datagram service offered by IP. ST must operate on the same level as IP in order to access local net resources such as SATNET streams and to be able to take advantage of any available local net multi-address delivery capabilities to support conferencing applications. If an ST agent shares a local net port with an IP datagram handler, the two must cooperate in the use of the port to regulate traffic flow through the port.

In order to get the advantage of abbreviated headers on stream packets, ST uses a different header format than that used for IP datagrams. Packets with this format (see Section 5.0 for details) are called ST packets in this document. They pass from one ST agent to another, and the abbreviated header information changes on a hop-to-hop basis. However, ST packets cannot be transmitted until a route for the stream has been found and intermediate agents have built routing tables to translate the abbreviated headers. Since end-to-end negotiation between ST users is often desirable before stream routing takes place, for example to agree on vocoder type and data rate, and it is convenient for a user to interface to only one protocol handler, ST provides a second type of service. This service uses IP datagrams with an "ST" value in the IP Protocol Field. These packets are called "IP.ST" packets. They pass through datagram handlers in gateways and reach ST agents only at their destination hosts.

A third type of packet is allowed by the protocol. This type is realized by embedding an ST packet in an IP.ST packet. This method of sending an ST packet allows it to pass through gateways that do not support the ST protocol but do support IP datagrams. Of course, the packet efficiency and traffic control benefits of ST are lost in such a case, but the use of this artifice could be justified on the grounds that any communication is better than none.

4.0 CONCEPTS

The key concept in ST is that of a connection. Connections are supported by entities called agents which are made aware of the connection during a setup process that precedes use of the connection for data transfer.

4.1 Agents

There are four types of agents that may be involved in supporting ST connections. They are:

4.1.1 ST Hosts

The users of connections are processes that run in host computers and communicate over connections through other processes or software modules that adhere to the ST protocol. Hosts having these processes or modules are called "ST hosts" (or "hosts," when the context permits). ST hosts perform the functions of gateway halves in interacting with gateways for internet traffic. ST hosts share the management of local net ST resources with the other agents on the local net and are capable of routing connections to other agents as may be required. In networks with local multi-addressing capability, ST hosts make use of this capability in routing conference connections. In networks lacking such capability, ST hosts may need to replicate messages for conference connections unless a special agent called a "replicator" is available in the local net. In some local nets it may be desirable for hosts to forward traffic for conference connections. The protocol allows but does not require the latter capability.

4.1.2 ST Gateways

ST gateways perform routing and forwarding functions very similar to those performed by IP gateways. Unlike IP gateways, they store information about the connections they support and share the management of resources in the nets to which they are connected with the other agents in those nets. Like hosts, ST gateways may have to replicate packets for conference connections.

4.1.3 Replicators

In networks that lack multi-addressing or broadcast capability it may be desirable to provide special server hosts to handle the replication required for conferences. Replicators are needed in situations where the load caused by replication would produce congestion at a gateway port. Use of a replicator adds delay and is probably not warranted unless the number of copies needed in a particular net exceeds some threshold that depends

upon network port capacity. In worst-case situations a daisy-chain type of replication might be required because the peak rate could not be sustained at any network site. The existence of a replicator does not eliminate the need for replication in hosts and gateways. For example, a host in a conference with some participants on the same net but others on other nets may need to send packets to one or more gateways for speedy internet delivery as well as to a replicator for automatic distribution to other local net participants.

4.1.4 Access Controllers

The management of ST conference connections involves the services of an access controller. The functions of an access controller are to control conference participation and provide a central source for information about the data rate requirements of a conference connection. Ideally, access control services would be provided by a set of hosts distributed throughout the catenet that shared information about the connections being controlled. The addresses of these public access controllers would be known to all other agents, and a query to any one controller would provide information about any connection. In the absence of public access controllers, the protocol allows any host to serve as a private access controller. It is proposed to use a bit in the conference connection name to allow agents to determine whether a public or private access controller is responsible for a particular conference. The name identifies the "owner" of the conference. The owner is also the access controller in the private case.

4.2 Connections

Most applications for ST connections require full-duplex (bi-directional) communication between the parties in a point-to-point connection and omni-directional communication among the participants in a conference connection. In the design of the protocol two different approaches to realizing the desired capability have been considered. The first, called the simplex approach, uses a combination of simplex (one-way) connections. For example, in the simplex approach the caller requests a simplex connection to the called party, who, after accepting the connection, requests another simplex connection for the return path to the caller. In the second, called the full-duplex approach, the caller requests a full-duplex connection at the outset, and as soon as the called party has accepted the connection, data can flow in both directions.

For conference connections, the simplex approach requires each participant to request a simplex connection to all the others. The full-duplex approach requires that a participant request connection only to those that have not already requested

connection to him.

Both approaches can provide workable bases for the required capabilities. The pros and cons for both may be summarized as follows:

1. Simplex connections can take maximum advantage of available resources by using different routes for the forward and return paths. The routing of a full-duplex connection is more likely to fail since a path with the desired capacity in both directions must be found. This advantage for simplex connections is most pronounced in networks where load is assymetrical, a situation to be expected in nets carrying relatively heavy data loads.

2. Full-duplex connections can, except perhaps under conditions of heavy load, be set up more rapidly and with less control message traffic. The difference is most pronounced for conference connections. With full-duplex components of a conference connection, m-1 connection requests are required for an m-participant conference, since each new participant must connect to all those already in the conference. In the case of simplex components each new participant must also connect to all those already in the conference; but, in addition, those already in must connect to each newcomer. This activity adds sigma (m-1) connection requests (and responses) to the setup procedure.

3. Simplex connections have an advantage in situations in which two parties attempt to call each other at the same time. The two simplex connections can easily be combined into the required full-duplex connection. If the two parties start out with full-duplex connections, one of them must be refused or disconnected, a somewhat more complex task for the higher level protocol requesting the connection.

This document proposes a full-duplex basis for ST connections because the author believes that the advantage of relative simplicity and efficiency in setting up conference connections outweighs the advantages of the simplex basis. To allow connections with assymetrical flow requirements, the protocol allows users to specify different data rates in the two directions.

Even though traffic can flow in both directions on an ST connection, the connection has an orientation, and packets are said to move in either the "forward" or "backward" direction depending on whether they are moving away from or toward the

originator of the connection.

ST provides two types of connections: Point-to-Point (PTP) and Conference (CONF). PTP connections use different packet header formats and setup procedures to reduce overhead and allow faster setup for that more frequently used type.

4.2.1 Point-to-Point (PTP) Connections

PTP connections are set up in response to a CONNECT command from an originating process to an ST agent. The CONNECT specifies the following:

1. The NAME of the connection. The NAME is obtained by concatenating the ST address of the originating process (ORIGIN) with an arbitrary number. The ST address is the internet host address (ala IP) concatenated with an "extension" field (32 bits) to specify a process in the host (a telephone for NVP applications). It is the responsibility of the originating process to provide arbitrary numbers that keep the names of all outstanding connections unique.

2. The internet address of the process to which the connection is desired. This address is called the "TARGET." The terms "ORIGIN" and "TARGET" are used instead of "SOURCE" and "DESTINATION" because the latter terms will be used to refer to the senders and receivers of packets travelling on the connection. Thus the ORIGIN process can be both SOURCE and a DESTINATION for packets on the full-duplex connection.

3. A flow specification (FLOW-SPEC) that tells ST agents about the desired characteristics of the connection. In addition to information about the data rate requirements for both directions of the full-duplex connection, the FLOW-SPEC has a PRECEDENCE value that agents can use as a basis for the preemption of this or other connections as part of the traffic control strategy. The FLOW-SPEC is discussed in more detail in Section 4.5.

4. An arbitrary 16-bit number that the agent is to use to identify all ST packets that it will send to the originator on the connection (the backward direction). This identifier is called the "CID.B." If the connection request is accepted, the originator will be given a CID.F to be used to identify all packets it sends in the forward direction on the connection. These CID's allow abbreviated headers to be used on ST

packets and provide a means for agents to rapidly locate the stored forwarding table involved in handling a received packet. CID's are assigned by the agents receiving packets and need be only locally unique since they are reassigned on a hop-by-hop basis. The CID to be used on the next hop is stored in the agent's forwarding table.

During the setup procedure the CONNECT command propagates from agent to agent until it reaches the TARGET process. This propagation differs from ordinary packet forwarding in that the intermediate agents inspect the command, take appropriate action, and retain information about the requested connection. If the TARGET process agrees to the connection, it sends an ACCEPT command that is propagated back through the same intermediate agents that handled the CONNECT. The agents take appropriate action as they process the ACCEPT. If the TARGET process is not willing to accept the connection, it issues a REFUSE command which propagates back in the same fashion as the ACCEPT. REFUSE's are generated by intermediate agents if they find themselves unable to support a requested connection. An agent receiving such a REFUSE tries alternate routes and passes the REFUSE back another hop only when it has exhausted its routing alternatives. Appropriate REASON codes are included in the REFUSE commands.

After a connection has become established (an ACCEPT has reached the ORIGIN), changes to the FLOW-SPEC can be accomplished by the ORIGIN issuing a new CONNECT or the TARGET issuing a new ACCEPT command. (Actually, the TARGET can issue a new ACCEPT at any time after issuing the first ACCEPT, and it can also at that time begin sending packets on the connection although there is some hazard in doing so since they may pass the ACCEPT enroute and be discarded.) For the case where the FLOW-SPEC calls for a connection whose rate can be varied at the discretion of the catenet, intermediate agents issue CONNECT's and ACCEPT's to inform other agents and the end users about rate changes. These commands are marked to distinguish them from end user commands.

The ACCEPT command contains the same kinds of information as the CONNECT except that the backward connection identifier (CID.B) is replaced by a forward identifier (CID.F). In addition, the FLOW-SPEC will generally be different and will indicate the data rates and delay characteristics accepted by the agents. The CONNECT that arrives at the TARGET will be similarly modified from the CONNECT that was issued by the ORIGIN and will match the ACCEPT received by the ORIGIN. See Section 4.5 for a discussion of the changes that can occur to the FLOW-SPEC.

4.2.2 Conference (CONF) Connections

The type of connection required for voice conferencing is one in which any participant can send messages to all the others. Connections of this type have been called "omniplex" connections. ST realizes a conference connection by means of a superposition of tree-like components that start from an origin process (the root) and extend to a set of targets (the leaves). The set of participants in a conference is represented by a bit map. Each participant has a location in the conference bit map that is assigned by the access controller (AC). When a conference CONNECT command is given, a TARGET-BIT-MAP (TBM) is used to specify the set of targets to which connection is requested. The TBM is supplied by the AC when a participant joins a conference. The tree-like components all have the same NAME, and intermediate agents combine branches from the components whenever possible to minimize resources committed to the conference. Because of this combining, an ORIGIN-BIT-MAP (OBM) is needed to represent the set of originators that have requested connection to a particular participant.

The list of participating processes in a CONF connection is not carried in the CONNECT request but is is maintained by the AC and provided to agents and participants when needed. Another function of the AC is to provide the FLOW-SPEC for the connection to any agent on request. The reason for assigning these tasks to an access controller is to prevent unauthorized connection to a conference and to assure that all components of the connection use the same FLOW-SPEC.

The first step in establishing a conference is to install a list of participants and a FLOW-SPEC in an AC. The list of participants may be fixed at the outset or be allowed to grow during the course of the conference. A participant may depart from a conference, but his position in the list and the bit maps is not reused. The method by which the list of participants is made known to the AC is not of concern to ST itself and is not specified in this document. Higher level protocols such as a network voice protocol (NVP) engage in communications between participant processes and the AC in the process of setting up a conference. For example, an NVP issues a JOIN command to request access to a conference. If the NVP process is on the participant list or is otherwise acceptable, the AC responds with a WELCOME command that among other things tells the participating NVP its location in the CONF bit map. The NVP then sends TELL-ME messages to the AC to obtain the participant list and FLOW-SPEC for the CONF connection. This information is provided in INFO messages from the AC. Several of these messages may be required to transmit all the information about a large conference. The messages exchanged between participants and the AC are IP.ST datagrams. They cannot be ST packets because no ST connection

exists between the participants and the AC.

Once a participant has received a WELCOME message from the AC, it can issue a CONNECT.CONF command to its ST host agent. It uses a TARGET-BIT-MAP (TBM) that it received as part of the data in the WELCOME message. This TBM has bits set for all the previous joiners of the conference. The CONNECT.CONF will thus attempt to establish a full-duplex path to each of the previous joiners. These paths will make use of common links where possible and will result in a connection resembling a tree rooted at the site of the process originating the connection. When the CONNECT.CONF is issued by the originator it contains an ORIGIN-BIT-MAP (OBM) with a single bit set corresponding to the originating participant. If the CONNECT.CONF is successful (i.e., some subset of the targets are reached), an ACCEPT.CONF will be returned with bits set in the TBM indicating the participants to which connection has been achieved. In a CONF connection attempt, success may not be achieved with the entire set of targets specified by TBM. Some may be unreachable for any of a number of reasons. REFUSE.CONF messages will be returned for all such failures with bits in the TBM identifying the unreachable participants. If the failures in a particular attempt are due to more than one REASON, at least one REFUSE.CONF will be returned for each reason.

The technique for setting up conference connections proposed for ST results in each participant actively connecting to some subset of the others while accepting connections from the rest. The first participant does not issue a CONNECT and accepts all the others. The last connects to all the others and accepts none. Each participant can maintain up-to-date information about participation in the conference by utilizing the information in the CONNECT and ACCEPT messages it receives.

The CONNECT.CONF messages received by agents during the setup procedure do not contain information about the identity of the participants. In order to route the connection, the agents must acquire this information, and they do so by sending TELL-ME messages to the AC and getting INFO messages in response. They need to retain this information only during the routing phase of connection setup. Once the connection is established, bit map information in forwarding tables combined with a FORWARDING-BIT-MAP (FBM) in the ST packet is sufficient to handle the forwarding of packets on the connection. The FBM is used to specify the set of destinations for the packet. Thus a packet can be sent to all or any subset of the connection participants. The source of the packet is identified by a number representing the position of the source participant in the conference bit map.

In the case of a voice conference, no useful purpose is accomplished when many people speak at the same time. It is expected that a higher level protocol (part of NVP) would regulate the activity of the conference and would normally allow one or perhaps two persons to transmit speech at the same time. ST is not involved in this aspect of conference control except to the extent that if there are too many simultaneous talkers, the traffic-handling capability of the connection could be exceeded, and ST might discard some of the packets. The higher level control protocol should set the FLOW-SPEC for the connection to accommodate the expected traffic flow. Thus, for a simple one-at-a-time conference, ST would be asked for a data rate corresponding to a single speech stream.

The above discussion has described a connection arrangement suitable for supporting voice conferences in which any participant can transmit and be heard by all others. ST also provides another kind of multi-address message delivery capability. If only one participant issues a CONNECT.CONF command with a TBM specifying connection to all the others, a tree-like connection will be set up that allows the ORIGIN to send packets to all the others and receive from any of the others, but packets sent by the others will be received only by the ORIGIN.

4.2.3 Taking Connections Down

The process of taking a connection down is initiated either by an ORIGIN issuing a DISCONNECT message or a TARGET issuing a REFUSE. These messages propagate from agent to agent along the connection path so that intermediate agents can take appropriate action to clean up their stored information about the connection.

Connections can also be taken down as a result of intermediate agents detecting a faulty link or gateway or deciding to preempt the connection. In this case the agent or agents involved issue a DISCONNECT/ REFUSE pair that propagate in the appropriate directions. A REASON code in the messages informs the users as to the cause of the disconnection.

In the case of conference connections, bit maps allow selective disconnection and refusal.

4.3 Types of Service

ST offers two types of service for packets travelling on connections. Neither type has any delivery guarantees, i.e., there are no acknowledgements or retransmissions on either a hop-by-hop or an end-to-end basis. Neither type guarantees packet integrity; i.e., if local nets offer a type of service

that can deliver packets with bits in error, ST may use that type of service. The headers of ST packets are sum-checked by ST agents, but the data portions are not.

The two types of service differ in whether or not they use the channel capacity nominally allocated to the connection and also in the strategy used by intermediate agents in buffering them. The two types are:

1. Stream Packets (called ST.ST packets). These packets use the allocated resources and are buffered for a short time only, since they are intended for applications such as speech communication where a late packet is not worth delivering. They are discarded by intermediate agents if queue conditions indicate that they cannot be delivered in a timely fashion.

2. Datagrams (called ST.DG packets). These packets have the same form as ST.ST packets except for a flag bit in the header and travel over the same connection path. They use allocated resources only when spare capacity exists, e.g., when the ST.ST flow drops below the allocated value. Otherwise they share local net resources with other IP datagram traffic. They are buffered with a queueing strategy appropriate for datagram traffic and are discarded only when agent buffer resources approach exhaustion. They are intended for use by higher level protocols such as NVP in applications such as dynamic control of the "floor" in a conference. They are also used by ST itself for connection management.

4.4 Packet Aggregation

ST allows any ST packets, stream or datagram, to be aggregated together that have the same next-agent local-net destination. "Aggregation" is a form of multiplexing, but is given a different name to distinguish it from the multiplexing done in the IP Multiplexing protocol that allows multiplexing only for packets with the same end-to-end source and destination. The term "envelope" is used to refer to any ST message sent from one agent to another. An envelope may contain one or more ST packets and is limited in size by the maximum size of packet that the local net can carry. The envelope has a short header in addition to the header of the individual aggregated packets. See Section 5.0 for a description of header formats.

The ST aggregation technique requires agents to look inside of received envelopes and handle the packets as individual entities. This procedure adds to the computing load of gateways, but can achieve significant communication savings in networks

with high per-packet overhead such as SATNET, particularly when many short packets must be handled.

4.5 Flow Specifications

The FLOW-SPEC that is carried by CONNECT and ACCEPT messages contains several fields. Some are specified by the originator of the CONNECT. Others are produced either during the process of setting up the connection or changing its allowed flow characteristics. Some apply in common to both directions of the full-duplex connection. Others apply individually to allow different flows in the two directions and appear in pairs in the control messages.

Data rate, the basic quantity used in traffic control computations, is specified by means of three parameters; a stream interval (SI), a packet length (PL), and a duty factor (DF). The average expected data rate can be computed by taking the product of PL, DF, and the reciprocol of SI. The FLOW SPEC allows for one value each for SI and DF for each direction. However, as many as four values of PL can be provided as options, allowing the ST agents flexibility in allocating resources for some types of traffic flow.

The flow type (TYPE) parameter is intended to allow ST to take into account a variety of different user load characteristics. The set of possible types can be expected to grow with experience, but a relatively few types seem to be adequate to deal with presently contemplated voice encoding techniques. These are:

1. Fixed Rate. The data rate is held fixed for the life of the connection. A simple speech encoder that can run at only one rate would use this type value with all four PL's set to the same value. A somewhat more complex encoder that could run at more than one rate but could not change rates on the fly would use the fixed-rate type but could offer a choice of up to four values for PL. A variable-rate vocoder such as the LPC2 vocoder used in the ARPANET that has a rate that varies depending on the short time behavior of the speech signal would also use the fixed-rate type but would set the duty factor to a lower value than the 0.5 or so used by a simple encoder.

2. Multiple Rate. The data rate allowed can be of any of the four specified by the four PL's and the agents are free to change rates at any time to accommodate to network load changes. Whenever an agent changes the rate, it sends appropriate CONNECT and ACCEPT messages to tell other agents and the users about the change.

Since such rate changes require extra communication and processing in the catenet, agents would have to avoid frequent changes. This flow type would be used by enoders that run at a variety of rates and can switch rates rapidly but need to do so explicitly either because packet formats must change with rate changes or because some parameter such as sampling rate must be changed at sender and receiver. This flow type could also be useful for sending data rather than voice over ST connections.

3. Prioritized Variable Rate. This flow type is intended for use by certain advanced encoders of a kind called "embedded" where subsets of the coded bit stream can be stripped en route without loss of intelligibility. There is, of course, some loss of quality and/or ability to withstand acoustical background noise when stripping occurs. For this flow type each of the four PL's corresponds to one of the four packet priorities that can be attached to ST.ST packets. The encoder would place the bits needed for its lowest rate in the highest priority packet, the next lowest in the second highest, etc. When pressed for channel capacity, agents would be free to discard the lower priority packets for this flow type. The overall precedence of the connection would also affect the probability of packet discard. It is not anticipated that agents would send explicit messages to announce that discarding was taking place.

Another set of parameters in the FLOW-SPEC is concerned with transmission delay. ST does not allow the user to specify a delay requirement, but it does allow some control over the tradeoff between delay and data rate options during the routing process. A ROUTING-STRATEGY parameter is provided for this purpose. Currently, two strategy options for PTP connections are envisioned, but others could be added if desired. One gives preference to minimizing delay at the expense of data rate. The other gives preference to data rate over delay. The ROUTING-STRATEGY options are meaningful only when data rate options are available. Otherwise data rate is as absolute requirement in routing.

While a user cannot specify a delay requirement to ST, ST does provide the user with an estimate of both minimum delay and delay dispersion in fields of the FLOW-SPEC. The estimates are based on a priori statistics relating delays to average network loads. When an agent propagates a CONNECT packet, it adds values from tables indexed on the current load estimate to the MIN-DELAY and DISPERSION fields of the FLOW-SPEC for the forward direction. It performs the same function for the backward direction as it

propagates the ACCEPT. The MIN-DELAY is the simple sum of the hop-to-hop contributions, but the DISPERSION is a sum of squares. The receiver can compute an estimate of overall delay by adding the MIN-DELAY to the square root of the DISPERSION. The DISPERSION estimate by itself can be useful in setting the reconstitution delay value needed to play out satisfactory speech for listeners. The proper value can vary over a wide range depending on the path through a catenet of networks with very different delay characteristics.

Another parameter set by agents during the routing process is the ACCEPTED-RATE field. This field informs the users as to which of the four possible data rate options (PL's) have been accepted for each of the two directions of the connection. Of course, if none were acceptable, a REFUSE would be returned with a REASON code indicating unavailability of resources at the requested precedence level. Another flow-related reason for refusal could be an inability of the networks to handle a too-short stream interval.

All FLOW-SPEC parameters except PRECEDENCE and ROUTING-STRATEGY can be independently specified or are reported separately for each of the two directions of the full-duplex connection. The exceptions are required to apply to the entire connection to simplify the task of gateways in handling connections.

The ROUTING-STRATEGY field has other control functions in addition to weighting the tradeoff between data rate and delay. For CONF connections it indicates whether or not data rate options must match in both directions (a requirement for voice conferencing) or can be negotiated independently. If ST agents support split routing, (a capability to divide the traffic on a connection among two or more paths) the ROUTING-STRATEGY field will indicate whether or not this technique is to be applied to the connection. Split routing also requires additional fields to indicate the fraction of the nominal traffic that has been accepted or is requested to be handled. This document does not propose the implementations of split routing in the first version of ST.

5.0 PACKET FORMATS

The messages sent between ST agents on connections are envelopes containing one or more ST packets. The envelope consists of an envelope header (EH) followed by one or more packet headers (PH's) followed by the data portions of the packets in the same order. The envelope thus has the form:

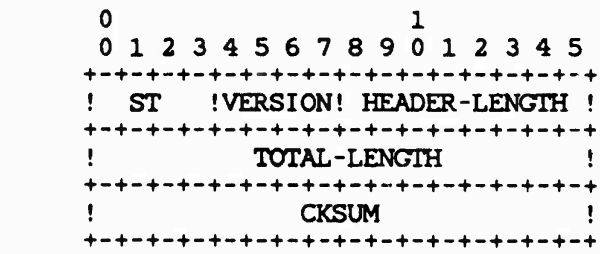
EH, PH1, PH2, . . .PHn, DATA1, DATA2, . . . DATAn

The reason for aggregating the headers separately from the data is that doing so allows the header region to be checksummed easily as a unit before attempting to parse the envelope. It is expected that ST will be used in networks that can deliver messages with bits in error and that some non-negligible fraction of the messages will have such errors. To require the entire envelope to be error-free in order to use any of it would result in an excessive rate of lost packets.

Since ST operates as an extension of IP, the envelope arrives at the same network port that IP uses to receive IP datagrams. It is proposed to use a unique code in the first field of the message to identify it as an ST envelope. The first four bits of an IP datagram are defined to be the Version Number field. It is therefore proposed to use one of the 16 possible IP versions to distinguish ST envelopes from IP datagrams. With this convention an envelope header will have the following format:

IEN 119 ST.DOC 7 September 1979

ENVELOPE HEADER



ST is the particular IP Version Number assigned to identify ST envelopes.

VERSION is the ST version number. This document is a proposal for VERSION 1.

HEADER-LENGTH* is the length in words of the envelope header (3) plus the sum of the header lengths of the aggregated packets.

TOTAL-LENGTH is the length of the entire envelope. It does not include any local net headers or trailers.

CKSUM covers the envelope header and all packet headers.

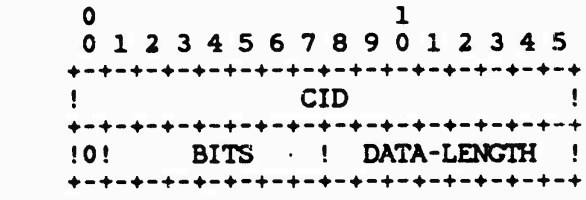
++++++++++++++

*All ST communications use the 16-bit word as a basic unit. All lengths are in word units.

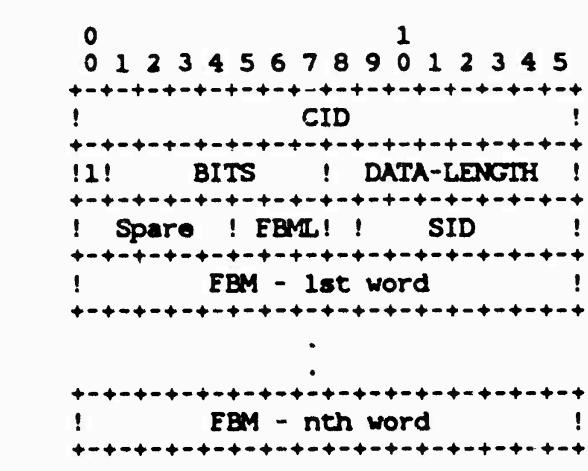
++++++++++++++

The individual packet headers have one of two formats depending on whether they are for PTP or CONF connections. These formats are:

PTP PACKET HEADER



CONF PACKET HEADER



CID is an arbitrary identifier assigned by the agent receiving the packet for the purpose of identifying the connection on which the packet is travelling. Since the CID is unique only to the agent that assigned it, it will generally have a different value on each hop of the connection path.

BITS are defined as follows:

Bit 1 distinguishes stream packets (ST.ST) from datagrams (ST.DG) (1 = DG).

Bits 2 and 3 define the packet priority (00 = highest priority).

Bits 4 and 5 are spares.

Bits 6 and 7 are unused (may be used by higher level protocols if desired).

DATA-LENGTH is the length of the data field in words.

FBML (3 bits) is one less than the length of the Forwarding Bit Map (FBM) in words.

SID (7 bits) identifies the source of the packet on a CONF connection (the source is implicit for a PTP connection packet). The value of SID corresponds to the bit position of the source in the conference bit map. Bit numbers start with zero, and positions start with the left-most (most significant) bit of the first word of the bit map.

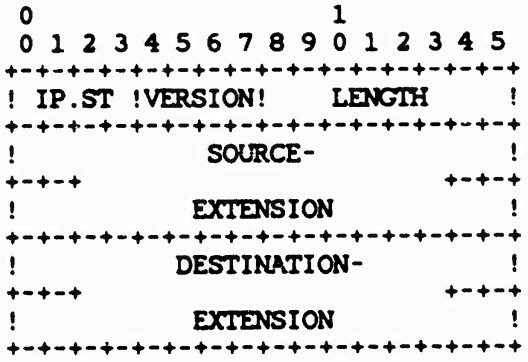
FBM is the Forwarding Bit Map. It can be at most 128 bits (8 words) long, and thus it limits conferences to 128 participants (a generous number). Ones in the FBM indicate that the packet is to be delivered to the corresponding participants. The FBM is allowed to increase in one word increments to allow new participants to enter during the course of a conference, but it does not shrink when participants leave, and bit positions are not reused.

As pointed out in Section 3.0, ST supports a second type of communication called IP.ST datagrams. These are ordinary IP datagrams with an "ST" value in the protocol field. They are used to allow higher level protocols to communicate prior to the setting up of an ST connection, and they are also used for communication between access controllers and other ST agents during the setup of CONF connections. They are strictly point-to-point communications since they are IP datagrams. According to the conventions for IP datagrams, these messages would have the form:

IP Header, IP.ST Header, Data

The IP.ST Header has the following form:

IP.ST PACKET HEADER



IP.ST is a value chosen to be different from the "ST" value used in the first four bits of the ST envelope. This field allows IP.ST datagrams to be distinguished from ST envelopes embedded in IP.ST packets, a technique that can be used to get ST envelopes through gateways that do not support ST.

VERSION is the ST version number.

LENGTH is the total length of the IP.ST packet excluding IP and local net headers, etc.

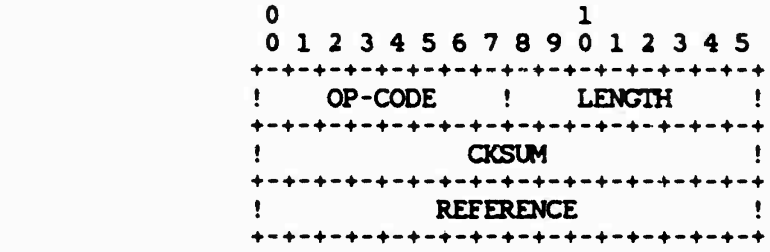
SOURCE- and DESTINATION-EXTENSION's are 32-bit fields used to identify the source and destination processes. Like ARPANET NCP process identifiers, they are not specified by the protocol. The source and destination host addresses are carried in the IP header.

6.0 CONTROL MESSAGES

With the exception of communications with access controllers, ST control messages are sent from agent to agent as ST.DG packets with the CID set to zero. This convention is similar to the ARPANET NCP use of Link 0 for control. Communication with AC's uses IP.ST packets. The form is otherwise the same. The control protocol follows a request-response model with all requests expecting responses and all responses expecting acknowledgements. Retransmission after timeout is used to allow for lost or ignored messages. A packet may contain more than one control message. Control messages do not extend across packet boundaries.

Control message headers have the following format:

CONTROL MESSAGE FORMAT



OP-CODE specifies the request or response.

LENGTH is the length of the control message in words.

CKSUM is the checksum of the control message. Because the control messages travel in envelopes that may be delivered with bits in error, each control message must be checked before it is acted upon.

REFERENCE is an arbitrary reference number used to associate requests with responses and acknowledgements.

The header is followed by parameters as required for the particular OP-CODE. Each parameter is identified with a P-CODE byte that is followed by a P-LENGTH byte indicating the length of the parameter (including the P-CODE, P-LENGTH word) in words. Parameters can be sent in any order. The format of individual parameters is specified in the following sections in connection with the OP-CODE's with which they are used.

Control messages fall into two categories according to whether they deal with PTP or CONF connections. There are four messages that are independent of connection type. These are:

6.0.1 [ACK]

ACK (OP-CODE = 1) has no parameters. The REFERENCE in the header is the REFERENCE number of the message being acknowledged. ACK's are used to acknowledge responses to requests and in some cases constitute responses or partial responses themselves.

6.0.2 [HELLO]

HELLO (OP-CODE = 2) is used to determine whether or not another agent is alive and well. It has no parameters and expects an ACK in response.

6.0.3 [ERROR-IN-REQUEST] <REF> <ERROR-TYPE>

ERROR-IN-REQUEST (OP-CODE = 3) is sent in response to a request in which an error is detected. An ACK is expected. No action is taken on the erroneous request.

REF (P-CODE = 7, P-LENGTH = 2) is the REFERENCE number of the erroneous request.

ERROR-TYPE is not yet specified.

6.0.4 [ERROR-IN-RESPONSE] <REF> <ERROR-TYPE>

This message (OP-CODE = 4) is sent in lieu of an ACK for a response in which an error is detected. No ACK is expected. Action taken by the requester and responder will vary with the nature of the request.

REF identifies the erroneous response.

ERROR-TYPE is not yet specified.

6.1 Control Messages for PTP Connections

PTP connections are set up and taken down with the following messages:

6.1.1 [CONNECT.PTP] <NAME> <TARGET> <FLOW-SPEC> <CID.B>

CONNECT.PTP (OP-CODE = 5) requests the set up (routing) of a PTP connection or asks for a change in the flow specification of a connection already routed. Its parameters are:

NAME (P-CODE = 1, P-LENGTH = 6) is the ST address of the process that originated the CONNECT.PTP (the ORIGIN) concatenated with a 16-bit number chosen to make the name unique. An ST address is a 32-bit IP host address concatenated with a 32-bit EXTENSION identifier chosen to identify a particular process in the host. The EXTENSION is provided by some higher-level protocol and is assumed by ST to be unique to the host. For NVP use the EXTENSION identifies a particular telephone and is presumably a well-known

quantity.

TARGET (P-CODE = 2, P-LENGTH = 5) is the ST address of the target process.

FLOW-SPEC (P-CODE = 3, P-LENGTH = 18) is a complex parameter that both specifies and reports on the flow requirements and expected delay characteristics of the full-duplex connection. See Section 4.5 for further information.

CID.B (P-CODE = 4, P-LENGTH = 2) is the connection identifier to be used on packets moving in the backward direction on the connection.

CONNECT.PTP expects a response. There are four response possibilities: ACCEPT.PTP, REFUSE.PTP, ACK, and ERROR-IN-REQUEST. Receipt of an ACK means that the agent receiving the request is working on it, and the requester should wait for a future ACCEPT or REFUSE. ERROR-IN-REQUEST will be returned only when a format error is detected in the CONNECT.PTP. Other errors, if detected, will elicit REFUSE messages.

The processing of CONNECT messages requires care to avoid routing loops that could result from delays in propagating routing information among gateways. The example in Section 7.0 describes in some detail the actions of agents in handling CONNECT requests while routing a connection.

6.1.2 [ACCEPT.PTP] <NAME> <TARGET> <FLOW-SPEC> <CID.F)

ACCEPT.PTP (OP-CODE = 6) is returned to indicate that the requirements of a CONNECT.PTP have been met or that a change in flow specifications has occured. Parameters are the same as for CONNECT.PTP except that a CID.F (P-CODE = 5, P-LENGTH = 2) is returned for use on packets travelling in the forward direction. The FLOW-SPEC will be modified to show the accepted rate and accumulated delay information (See Section 4.5).

ACCEPT messages expect ACK's or ERROR-IN-RESPONSE's. ERROR-IN-RESPONSE will be returned if an ACCEPT is sent to an agent that has no knowledge of the connection. This may occur if an ACCEPT is generated at the same time that a DISCONNECT is being propagated.

6.1.3 [REFUSE.PTP] <NAME> <REASON>

REFUSE.PTP (OP-CODE = 7) is returned to indicate that agents have failed to set up a requested connection or that a previously established connection has been lost. REFUSE's are also returned to indicate routing failure, and in such a case may not end up

propagating back to the origin. TARGET's also issue REFUSE's to take down connections intentionally.

REASON (P-CODE = 6, P-LENGTH = 2) indicates the reason for connection refusal. REASON codes apply also to DISCONNECT messages and include the following:

CODE	EXPLANATION
0	No explanation
1	Target refuses connection
2	Target does not respond
3	Target cannot be reached
4	Connection preempted
5	STREAM INTERVAL too short
6	Requested data rate cannot be handled
7	Connection broken due to network fault
8	Connection broken by ORIGIN
9	Conflicting FLOW-SPECs in CONF connections

REFUSE's are ACK'ed and are propagated by intermediate agents if meaningful (i.e., the agents had tables for the connection). The backward propagation of a refuse may be halted at an intermediate agent if an alternate route exists that has not been tried, and the REASON indicates that it is reasonable to try the alternate route. (I.e., it does not indicate that the target refuses or does not respond).

6.1.4 [DISCONNECT.PTP] <NAME> <REASON>

DISCONNECT.PTP (OP-CODE = 8) is sent to request that a previously requested connection be taken down. It can be generated either by the originator of the CONNECT or by an intermediate agent that executes a preemption or detects a fault.

REASON uses the same codes as REFUSE although not all codes apply.

DISCONNECT expects an ACK and is propagated in the forward direction so long as agents are encountered that know about the connection.

A connection can be taken down either by a REFUSE or a DISCONNECT (or both) depending upon which end first decides to initiate the process. If both start within a propagation time of each other, neither message will reach the opposite end.

6.2 Control Messages for CONF Connections

CONF connections are set up and taken down with CONNECT, ACCEPT, REFUSE, and DISCONNECT messages, but the CONF versions of these messages have somewhat different parameters. In addition, CONF connection setup requires that agents communicate with access controllers by means of TELL-ME and INFO messages. These latter messages are sent as IP.ST datagrams. The former are sent as ST.DG packets with CID = 0.

6.2.1 [CONNECT.CONF] <NAME> <OBM> <TBM> <CID.B>

CONNECT.CONF (OP-CODE = 9) requests the setup (routing) of a CONF connection or asks for a change in flow specifications of a connection already routed. The parameters NAME and CID.B have the same form and interpretation as they do for CONNECT.PTP except that NAME is the name of the owner of the conference, not the originator of the CONNECT message. The new parameters OBM and TBM allow the message to deal with multiple ORIGIN and TARGET processes. The FLOW-SPEC for the connection is obtained from the access controller.

OBM (P-CODE = 8, P-LENGTH = 2-9) is the ORIGIN-BIT-MAP. Bits set in the map identify originating processes. When a CONNECT.CONF is first issued by a user process only one bit is set in OBM identifying the issuer. However, as the message propagates, intermediate agents may find that they have other CONNECT.CONF messages for the same connection on hand at the same time. In that case, they can merge the requests so that more bits become set as the message approaches its targets.

TBM (P-CODE = 9, P-LENGTH = 2-9) is the TARGET-BIT-MAP. Bits set in the map identify the target processes. In general, the user process will have set many bits in TBM when it first issues a CONNECT.CONF. As the message propagates it will split many times, each split reducing the number of bits left set in TBM. When the CONNECT.CONF's reach their targets only one bit will be left set in each.

Since the CONNECT.CONF message does not tell its receiver anything about the actual identities of the target processes, intermediate agents must get this information, as well as the FLOW-SPEC, from the access controller by sending TELL-ME messages and receiving INFO messages in response. The agents use the NAME to locate the AC, using a bit in the name to distinguish between a public or private AC. The NAME is the ST address of a process concatenated with a 16-bit number to make the NAME unique. It is proposed that the most significant bit of that 16-bit number be used to distinguish public from private ACs. A zero in that bit would indicate a private AC and in that case, agents would send

TELL-ME messages to the process address in the NAME. In the public case, the agent would communicate with an AC whose address was known a priori to the agent.

6.2.2 [ACCEPT.CONF] <NAME> <OBM> <TBM> <FLOW-SPEC> <CID.F>

ACCEPT.CONF (OP-CODE = 10) is similar in function to ACCEPT.PTP. NAME, FLOW-SPEC, and CID.F have the same form and interpretation. OBM specifies the set of originators to which the ACCEPT is to be propagated. TBM specifies the set of targets that have accepted the connection. This set may be a sub-set of the targets requested in the CONNECT to which an ACCEPT responds. The FLOW-SPEC is included in the ACCEPT because it reflects the actual resources granted to the connection.

6.2.3 [REFUSE.CONF] <NAME> <OBM> <TBM> <REASON>

REFUSE.CONF (OP-CODE = 11) is similar in function to REFUSE.PTP. As for ACCEPT.CONF, OBM specifies the set of originators to which the REFUSE is to be propagated. TBM specifies the set of targets that cannot be reached, have refused, etc. A single REASON applies to all the targets in the TBM. If more than one REASON applies to a set of targets, as many REFUSE's as REASON's will be sent.

6.2.4 [DISCONNECT.CONF] <NAME> <OBM> <TBM> <REASON>

DISCONNECT.CONF (OP-CODE = 12) is similar in function to DISCONNECT.PTP. As for REFUSE.CONF, OBM and TBM specify the sets of originators and targets to which the DISCONNECT applies.

6.2.5 [TELL-ME] <NAME> <PART-NUM> <FLOW-SPEC-REQ>

TELL-ME (OP-CODE = 13) is sent from an agent or a participant process to an access controller . The AC is expected to return an INFO message with the requested information. Either of the latter two parameters may be omitted.

PART-NUM (P-CODE = 10, P-LENGTH = 2) specifies the number of the first participant about which information is requested. The response will be a participant list starting with the specified participant and continuing until the maximum packet size is reached or the list is exhausted.

FLOW-SPEC-REQ (P-CODE = 11, P-LENGTH = 2) requests the AC to send the FLOW-SPEC for the connection.

6.2.6 [INFO] <NAME> <STATUS> <PART-LIST> <FLOW-SPEC>

INFO (OP-CODE = 14) is sent from an AC to an agent or participant process in response to a TELL-ME. It provides the requested information. STATUS is always present. PART-LIST and FLOW-SPEC are present only when requested by the TELL-ME.

STATUS (P-CODE = 12, P-LENGTH = 2) carries 2 bytes of information. Byte 1 is the CONF-TYPE. Byte 2 gives the length of the participant list. The following values for CONF-TYPE are defined:

Type	Meaning
0	No conference defined with this NAME
1	Conference with closed participant list
2	Conference with open list and password
3	Conference with completely open list (no password needed).

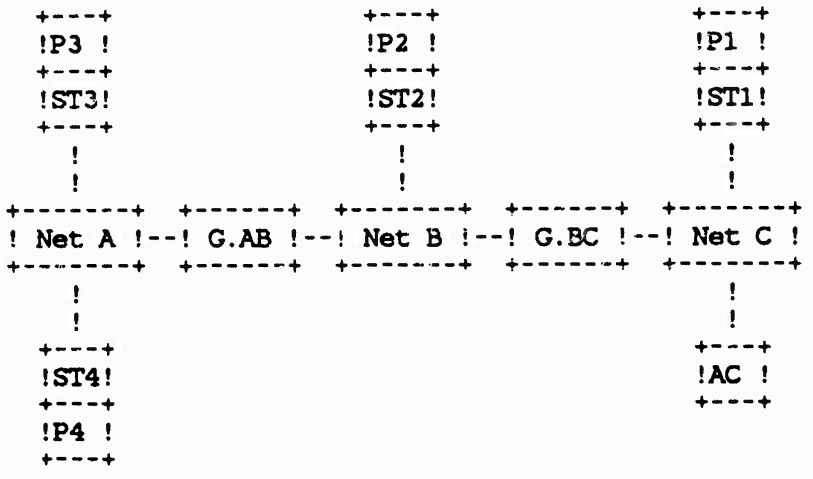
PART-LIST (P-CODE = 13, P-LENGTH = (4m + 2)) provides a section of the participant list starting at the location (PART-NUM) requested in the TELL-ME and continuing until either the end of the list or packet capacity is reached. The items in the PART-LIST are the ST addresses (64 bits) of the participating processes. The addresses are present whether or not the participants are active. The addresses are preceded by a word giving the number of the first participant on the list.

FLOW-SPEC is the nominal FLOW-SPEC for the conference.

7.0 AN EXAMPLE OF CONF CONNECTION SETUP

This section is a rather detailed example of the actions called for by ST in setting up a connection for a conference with four participants. In addition to showing the control message flow, it also indicates the information used and retained by gateways in supporting the connection. For the sake of simplicity, it is assumed that the flow requirements are always met. The ".CONF" suffix is omitted from OP-CODE's, and parameters such as NAME and FLOW-SPEC that are always the same are also omitted. In addition, ACK's are not shown but are assumed to occur where required.

The example uses the following network configuration:



Each participant (Pi) communicates through a host agent called "STi." The communications between the P's and their local ST's are written out as control messages to show the logical flow even though in actual implementations they might be handled very differently.

The actions involving ST start after the participants have joined the conference by communicating with the access controller (AC) and have received TARGET-BIT-MAPs (TBMs) telling each Pi to which other Pi's connections are to be set up. The notation "{ A, B, C }" is used to indicate a bit map with bits set for A, B, and C. The participants are assumed to have joined in the order of their numbers. Thus P1 got an empty TBM ({ }), and P4 got TBM = { P1, P2, P3 }. According to the rules, P1 issues no CONNECT messages, but waits for the others to connect to it. The action thus begins with P2 sending:

P2->ST2: [CONNECT] <OBM = { P2 }> <TBM = { P1 }> <CID.B = 3>

ST2->AC: [TELL-ME] <PART-NUM = 1> <FLOW-SPEC-REQ>

AC->ST2: [INFO] <PART-LIST = ADDR.P1, ADDR.P2, ADDR.P3, ADDR.P4>
<FLOW-SPEC> <STATUS>

These last two commands are executed independently by all agents when they first receive a CONNECT. They will be replaced by the phrase "X gets info" in the following.

ST2 observes that ADDR.P1 is not in its local net and lacking routing knowledge decides to try G.AB (the wrong direction).

ST2->G.AB: [CONNECT] <OBM = { P2 }> <TBM = { P1 }> <CID.B = 17>

G.AB gets info and decides that net C is unreachable except through net B from whence the CONNECT came.

G.AB->ST2: [REFUSE] <OBM = { P2 }> <TBM = { P1 }>
<REASON = 3 (Target cannot be reached)>

ST2 decides to try another gateway.

ST2->G.BC: [CONNECT] <OBM = { P2 }> <TBM = { P1 }> <CID.B = 17>

G.BC gets info, builds a connection entry, and sends:

G.BC->ST1: [CONNECT] <OBM = { P2 }> <TBM = { P1 }> <CID.B = 1001>

ST1 gets info and sends:

ST1->P1: [CONNECT] <OBM = { P2 }> <TBM = { P1 }> <CID.B = 1>

Since P1 has already joined the conference and recognizes P2 as another participant, it sends:

P1->ST1: [ACCEPT] <OBM = { P2 }> <TBM = { P1 }> <CID.F = 1>

ST1->G.BC: [ACCEPT] <OBM = { P2 }> <TBM = { P1 }> <CID.F = 32>

At this point G.BC would have the following stored information (neglecting bookkeeping items such as pointers).

1. A connection block with NAME, FLOW-SPEC, and CID.IN = 1001 (the same CID can be used for all inputs for the connection). This information is retained for the life of the connection. The PART-LIST used in processing may be discarded once an ACCEPT (or REFUSE) has been received and the forwarding tables have been created. However, since there are likely to be other CONNECT's to be processed, it would be efficient to keep the PART-LIST for a time (say several minutes).

2. Two forwarding tables, one for each packet that might be sent in response to an input.

ITEMS	#1	#2
NET-PORT	B	C
ADDRESS	ST2	ST1
MASK.OBM	{ P2 }	{ }
MASK.TBM	{ }	{ P1 }
CID.OUT	17	32

The principal function of the masks is to facilitate packet forwarding. When a packet arrives, the following computation is made for each forwarding table to compute the output FORWARDING-BIT-MAP (FBM):

FBM.OUT = FBM.IN & (MASK.OBM U MASK.TBM)

If FBM.OUT has no bits set, it is not necessary to send a packet to the address in the table. Otherwise a packet is sent using the NET-PORT, ADDRESS, and CID.OUT from the table.

Having built its tables, G.BC sends:

G.BC->ST2: [ACCEPT] <OBM = { P2 }> <TBM = { P1 }> <CID.F = 1001>

ST2->P2: [ACCEPT] <OBM = { P2 }> <TBM = { P1 }> <CID.F = 10>

At this point P2 and P1 are connected and could begin talking, if permitted by the higher level protocol.

In connecting P3 and P4 we will assume that both initiate requests at essentially the same time so that they propagate concurrently.

P3->ST3: [CONNECT] <OBM = { P3 }> <TBM = { P1, P2 }> <CID.B = 5>

P4->ST4: [CONNECT] <OBM = { P4 }> <TBM = { P1, P2, P3 }>
<CID.B = 1>

ST3 and ST4 get info. ST3 notices that P1, P2 both are outside the local net, but ST4 notices as well that P3 is on the same net as P4.

They send:

```
ST3->G.AB: [CONNECT] <OBM = { P3 }> <TBM = { P1, P2 }>
            <CID.B = 135>

ST4->ST3: [CONNECT] <OBM = { P4 }> <TBM = { P3 }> <CID.B = 27>

ST4->G.AB: [CONNECT] <OBM = { P4 }> <TBM = { P1, P2 }>
            <CID.B = 27>
```

ST3 forwards the CONNECT to P3, which accepts, and ST3 responds to ST4 with:

```
ST3->ST4: [ACCEPT] <OBM = { P4 }> <TBM = { P3 }> <CID.F = 135>
```

Meanwhile G.AB gets info and notices that it has two CONNECT's for the same NAME. It decides to merge them and sends:

```
G.AB->ST2: [CONNECT] <OBM = { P3, P4 }> <TBM = { P2 }>
            <CID.B = 2356>
```

and

```
G.AB->G.BC: [CONNECT] <OBM = { P3, P4 }> <TBM = { P1 }>
             <CID.B = 2356>
```

ST2 forwards the CONNECT to P2, which accepts, and ST2 sends:

```
ST2->G.AB: [ACCEPT] <OBM = { P3, P4 }> <TBM = { P2 }>
            <CID.F = 17>
```

Now G.AB will not continue to propagate the ACCEPT because the CONNECT on which it is working asked for connection to P1 as well as P2. It will wait for an ACCEPT or REFUSE from P1.

G.BC already knows about the connection to P1, but it does not assume that P1 will accept P3 and P4, so it propagates the CONNECT.

```
G.BC->ST1: [CONNECT] <OBM = { P3, P4 }> <TBM = { P1 }>
            <CID.B = 1001>
```

ST1 forwards to P1, which accepts, and ST1 responds:

```
ST1->G.BC: [ACCEPT] <OBM = { P3, P4 }> <TBM = { P1 }> <CID.F =
32>
```

In the latter exchange G.BC and ST1 used the same CID's they had used before for this connection. If either had chosen to use a different CID, the newer value would supercede the earlier one in the forwarding table.

It should be noted that the protocol could allow G.BC to accept the connection from P3 and P4 without forwarding the CONNECT to ST1 because G.BC already knows it has a connection to P1. This shortcut is not taken because it denies P1 the information about the connection requests from P3 and P4 and the opportunity to refuse those connections if desired.

To finish the setup we have:

```
G.BC->G.AB: [ACCEPT] <OBM = { P3, P4 }> <TBM = { P1 }>
             <CID.F = 1001>
```

G.AB will now accept for P1 and P2.

```
G.AB->ST3: [ACCEPT] <OBM = { P3 }> <TBM = { P1, P2 }>
            <CID.F = 2356>

G.AB->ST4: [ACCEPT] <OBM = { P4 }> <TBM = { P1, P2 }>
            <CID.F = 2356>
```

When ST3 and ST4 propagate the ACCEPT's to P3 and P4 the conference connection is complete.

At this point the forwarding tables in G.BC are the following:

ITEM	#1	#2	#3
NET-PORT	B	C	B
ADDRESS	ST2	ST1	G.AB
MASK.OBM	{ P2 }	{ }	{ P3, P4 }
MASK.TBM	{ }	{ P1 }	{ }
CID.OUT	17	32	2356

If at some later time G.BC should decide to preempt the connection, it would issue one message for each forwarding table entry:

```
G.BC->ST2: [REFUSE] <OBM = { P2 }> <TBM = { P1 }>
            <REASON = 4 (Connection preempted)>

G.BC->ST1: [DISCONNECT] <OBM = { P2, P3, P4 }> <TBM = { P1 }>
            <REASON = 4 (Connection preempted)>
```

G.BC->G.AB: [REFUSE] <OBM = { P3, P4 }> <TBM = { P1 }>
<REASON = 4 (Connection preempted)>

Having issued these messages and received ACKs in response (or timed out in the absence of an ACK), the gateway can delete the table entries and reclaim the CID for future use. The REFUSE sent to G.AB would, of course, be propogated to ST3 and ST4.

8.0 AREAS NEEDING FURTHER WORK

This document does not completely specify the protocol. Further work is needed to specify error conditions and their handling. The FLOW-SPEC parameter is not yet laid out in detail. Rerouting has not been thought through sufficiently. The whole area of routing strategies and the information to be exchanged among gateways has not been given much consideration. There is also a need for agents to exchange information (not yet specified) about local net resources. For example, if agents are to make use of local net multi-addressing capability, the selection of a CID for a connection is no longer at the discretion of an individual agent. A convention is needed to avoid conflicting use of CID's as well as requesting duplicate resources to serve a CONF connection. The CONNECT control message needs to be extended to allow agents to indicate local net resources that are already committed to a CONF connection.

NWG/RFC 741 DC 22 Nov 77 42444

SPECIFICATIONS FOR THE NETWORK VOICE PROTOCOL (NVP) and Appendix 1: The Definition of Tables-Set-#1 (for LPC) Appendix 2: Implementation Recommendations

NSC NOTE 68

(Revision of NSC Notes 26, 40, and 43)

Danny Cohen, ISI

January 29, 1976

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

CONTENTS

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

PREFACE

The major objective of ARPA's Network Secure Communications (NSC) project is to develop and demonstrate the feasibility of secure, high-quality, low-bandwidth, real-time, full-duplex (two-way) digital voice communications over packet-switched computer communications networks. This kind of communication is a very high priority military goal for all levels of command and control activities. ARPA's NSC projrct will supply digitized speech which can be secured by existing encryption devices. The major goal of this research is to demonstrate a digital high-quality, low-bandwidth, secure voice handling capability as part of the general military requirement for worldwide secure voice communication. The development at ISI of the Network Voice Protocol described herein is an important part of the total effort.

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

ACKNOWLEDGMENTS

The Network Voice Protocol (NVP), implemented first in December 1973, and has been in use since then for local and transnet real-time voice communication over the ARPANET at the following sites:

- Information Sciences Institute, for LPC and CVSD, with a PDP-11/45 and an SPS-41.
- Lincoln Laboratory, for LPC and CVSD, with a TX2 and the Lincoln FDP, and with a PDP-11/45 and the LDVT.
- Culler-Harrison, Inc., for LPC, with the Culler-Harrison MP32A and AP-90.
- Stanford Research Institute, for LPC, with a PDP-11/40 and an SPS-41.

The NVP's success in bridging the differences between the above systems is due mainly to the cooperation of many people in the ARPA-NSC community, including Jim Forgie (Lincoln Laboratory), Mike McCammon (Culler-Harrison), Steve Casner (ISI) and Paul Raveling (ISI), who participated heavily in the definition of the control protocol; and John Markel (Speech Communications Research Laboratory), John Makhoul (Bolt Beranek & Newman, Inc.) and Randy Cole (ISI), who participated in the definition of the data protocol. Many other people have contributed to the NVP-based effort, in both software and hardware support.

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

1. INTRODUCTION

Currently, computer communication networks are designed for data transfer. Since there is a growing need for communication of real-time interactive voice over computer networks, new communication discipline must be developed. The current HOST-to-HOST protocol of the ARPANET, which was designed (and optimized) for data transfer, was found unsuitable for real-time network voice communication. Therefore this Network Voice Protocol (NVP) was designed and implemented.

Important design objectives of the NVP are:

- Recovery of loss of any message without catastrophic effects. Therefore all answers have to be unambiguous, in the sense that it must be clear to which inquiry a reply refers.
- Design such that no system can tie up the resources of another system unnecessarily.
- Avoidance of end-to-end retransmission.
- Separation of control signals from data traffic.
- Separation of vocoding-dependent parts from vocoding-independent parts.
- Adaptation to the dynamic network performance.
- Optimal performance, i.e. guaranteed required bandwidth, and minimized maximum delay.
- Independence from lower level protocols.

The protocol consists of two parts:

(1) The control protocol,

(2) The data protocol.

Control messages are sent as controlled (TYPE 0/0) messages, and data messages may be sent as either controlled (TYPE 0/0) or uncontrolled (TYPE 0/3) messages (see BBN Report 1822 for definition of MESSAGE-TYPE).

Throughout this document a "word" means a "16-bit quantity".

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

2. THE CONTROL PROTOCOL

Throughout this document the 12-bit MESSAGE-ID (see BBN Report 1822) is referred to as LINK (its 8 MSBs) and SUB-LINK (its 4 LSBs).

The control protocol starts with an initial connection phase on link 377 and continues on other links assigned at run time.

Four links are used for each voice communication:

Link L	will be used for control,	from CALLER to ANSWERER.
Link K	will be used for control,	from ANSWERER to CALLER.
Link L+1	will be used for data,	from CALLER to ANSWERER.
Link K+1	will be used for data,	from ANSWERER to CALLER.

Both L and K should be between 340 and 375 (octal). L and K need not differ.

The first message (CALLER to ANSWERER) on link 377 indicates which user wants to talk to whom and specifies K. As a response (on K), the ANSWERER either refuses the call or accepts it and assigns L.

The CALLER then calls again (this time on link L). The ANSWERER initiates a negotiation session to verify the compatibility of the two parties.

The negotiation consists of suggestions put forth by one of the parties, which are either accepted or rejected by the other party. The suggesting party in the negotiation is called the NEGOTIATION MASTER. The other party is called the NEGOTIATION SLAVE. Usually the ANSWERER is the negotiation master, unless agreed otherwise by the method described later.

If the negotiation fails, either party may terminate the call by sending a "GOODBYE". If the negotiation is successfully ended, the ANSWERER rings bells to draw human attention and sends "RINGING" to the CALLER. When the call is answered (by a human), a "READY" is sent to the CALLER and the data starts flowing (on L+1 and K+1). However, a "READY" can be sent without a preceeding "RINGING".

This bell ringing occurs only after the initial call (not after renegotiation).

The assignment of L and K cannot be changed after the initial connection phase.

Only one control message can be sent in a network-message. Extra bits needed to fill the network-message are ignored.

The length of control messages should never exceed a single-packet (i.e., 1,007 data bits).

Control messages not recognized by their receiver should be ignored and should not cause any error condition resuting in termination of the connection. These messages may result from differences in implementation level between systems.

SUMMARY OF THE CONTROL MESSAGES

#1 "1,<WHO>,<WHOM>,K"

#2 "2,<CODE>" or only "2"

#3 "3,<WHAT>,<N>,<HOW(1),...HOW(N)>"

#4 "4,<WHAT>,<HOW>"

#5 "5,<WHAT>,<HOW>" or only "5,<WHAT>"

#6 "6,L" or only "6"

#7 "7"

#8 "8"

#9 "9"

#10 "10,<ID>"

#11 "11,<ID>"

#12 "12,<IM>"

#13 "13,<YM>,<OK>"

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

DEFINITION OF THE CONTROL MESSAGES

#1 CALLING (on 377 and L)

This call is issued first on link 377 and later on link L. Its format is "1,<WHO>,<WHOM>,K", where <WHO> and <WHOM> are words which identify respectively the calling party and the party that is being called, and K is as defined above. The format of the <WHO> and <WHOM> is:

(HHIIIIIIXXXXXXXX)

where HH are 2 bits identifying the HOST, followed by 6 bits identifying the IMP, followed by 8 bits identifying the extension (needed because there may be more than one communication unit on the same HOST).

The system which sends this message is defined as the CALLER, and the other system is defined as the ANSWERER.

#2 GOODBYE (TERMINATION, on L or K)

This message has the purpose of terminating calls at any stage.

ICP can be terminated (on K) either negatively by sending either a single word "2" ("GOODBYE") or the two words "2,<CODE>", or positively by sending the two words "6,L", as described later.

After the initial connection phase, calls can be terminated by either the CALLER (on L) or the ANSWERER (on K). This termination has two words: "2,<CODE>", where <CODE> is the reason for the termination, as specified here:

0. Other than the following.

1. I am busy.

2. I am not authorized to talk with you.

3. Request of my user.

4. We believe you are down.

5. Systems incompatibility (NEGOTIATION failure).

6. We have problems.

7. I am in a conference now.

8. You made a protocol error.

#3 NEGOTIATION INQUIRY (on L or K)

Sent by the NEGOTIATION MASTER for compatibility verification. The format is:

"3,<WHAT>,<LIST-LENGTH>,<HOW-LIST>", meaning

"CAN-YOU-DO,<WHAT>,<LIST-LENGTH>,<HOW-LIST>".

The <HOW-LIST> is a list of pointers into agreed-upon tables, as shown below.

#4 POSITIVE NEGOTIATION RESPONSE (on L or K)

Sent by the NEGOTIATION SLAVE in response to a NEGOTIATION INQUIRY. The format is:

"4,<WHAT>,<HOW>", meaning: "I-CAN-DO,<WHAT>,<HOW>".

#5 NEGATIVE NEGOTIATION RESPONSE (on L or K)

Sent by the NEGOTIATION SLAVE in response to a NEGOTIATION INQUIRY. The format is either:

"5,<WHAT>,0", meaning "I-CAN'T-DO-<WHAT>-IN-ANY-OF-THESE-WAYS",

or: "5,<WHAT>,N", meaning inability to accept any of the options offered in the INQUIRY, but using "N" as a suggestion to the ANSWERER about another possibility. Examples are presented later in this report.

#6 READY (on L or K)

Sent by either party to indicate readiness to accept data. Its format is "6,L" in the reply to the initial call, and "6" thereafter.

#7 NOT READY (on L or K)

Sent by either party to indicate unreadiness to accept data. It is always a single word: "7".

#8 INQUIRY (on L or K)

Sent by either party to inquire about the status of the other. It is always a single word: "8". It is answered by #6, #7, or #9.

#9 RINGING (on K)

Sent by the ANSWERER after the negotiations have been successfully terminated and human permission is needed to proceed further. The ringing will continue for 10 seconds, and then stop, UNLESS a #8 is received. This message is always a single word: "9".

#10 ECHO REQUEST (on L or K)

Sent by whichever party is interested in measuring the network delays. Its only purpose is to be echoed immediately. The format is "10,<ID>", where <ID> is any word used to identify the ECHO.

#11 ECHO (on L or K)

Sent in response to ECHO REQUEST. The format is "11,<ID>", where <ID> is the word specified by #10. The implementation of this feature is not compulsory, and no connection should be terminated due to lack of response to ECHO-REQUEST.

#12 RENEGOTIATION REQUEST (on L or K)

Can be sent by either party at ANY stage after LINKS are agreed upon. This message consists of the two words "12,<IM>". If the word <IM> (for I MASTER) is non-zero, the sender of this message requests to be the NEGOTIATION MASTER. If it is zero, the receiver of this message is requested to be the NEGOTIATION MASTER. Renegotiation is described later.

#13 RENEGOTIATION APPROVAL (on L or K)

This message may be sent by either party in response to RENEGOTIATION REQUEST. It consists of the three words "13,<YM>,<OK>". If <OK> is non-zero, this is a positive acknowledgment (approval). If it is zero, this is a negative acknowledgment (i.e., refusal). <YM> is set to be equal to the <IM> of #12, for identification purposes.

Messages #7, #8, and #9 are always a single word. Messages #1, #3, #4, and #5 are several words long. Messages #2 and #6 are either a single word or two words long. #10, #11 and #12 are always 2 words long. Message #13 is always 3 words long. Message #1 is always 4 words long.

Message #1 is sent only by the CALLER, #3 only by the NEGOTIATION MASTER, and #4 and #5 only by the NEGOTIATION SLAVE. Message #9 is

sent only by the ANSWERER. All the other control messages may be sent by either party.

The last <HOW> which was both suggested by the NEGOTIATION MASTER (in #3) and accepted by the NEGOTIATION SLAVE (in #4) for each <WHAT> is assumed to be in use.

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

DEFINITION OF THE <WHAT> AND <HOW> NEGOTIATION TABLES:

<WHAT>	<HOW>
1. VOCODING	* 1. LPC + 2. CVSD 3. RELP 4. DELCO
2. SAMPLE PERIOD	
(in microseconds)	N. N (*150) (+62)
3. VERSION	
	* 1. V1 (see definition below) + 2. V2 (see definition below)
4. MAX MSG LENGTH (in bits)	
NVP header included (32 bits) but not HOST/IMP leader and not HOST/IMP padding	N. N (*976 and +976)
5. If LPC:	
Degree	N. For N coefficients (*10)
If CVSD:	
Time Constant (in milliseconds)	N. N (+50)
6. Samples per Parcel	N. N (*128) (+224)
7. If LPC:	
Acoustic Coding	* 1. SIMPLE (see below) 2. OPTIMIZED
8. If LPC:	
Info Coding	* 1. SIMPLE (see below) 2. OPTIMIZED

9. If LPC:

Pre-emphasis 1 - mu x [Z**-1] N = 64 x mu	N. N (*58, for mu = 58/64 = 0.90625)

10. If LPC:

Table-set	N. N (*1) See definition of Set #1 in Appendix 1

(* indicates recommended options for LPC)
(+ indicates recommended options for CVSD)

No parameter (<WHAT>) should be inquired about by the NEGOTIATION MASTER if some option (<HOW>) for it has been previously accepted by the NEGOTIATION SLAVE implicitly in the "VERSION". The purpose of this restriction is to avoid a possible conflict between individual parameters and the VERSION-option.

Version 1 (V1) is defined as:

1-1	LPC
2-150	150 microseconds sampling
3-1	V1
5-10	10 coefficients
6-128	128 samples per parcel
7-1	SIMPLE acoustic coding
8-1	SIMPLE information coding
9-58	mu = 58/64 = 0.90625
10-1	Tables set #1

Version 2 (V2) is defined as:

1-2	CVSD
2-62	62 microseconds sampling (16 KHz sampling)
3-2	V2
5-50	50 msec time constant
6-192	192 samples per parcel

Note that this defines every negotiated parameter, except MAX MSG LENGTH.

SIMPLE and OPTIMIZED codings will be described below in Section 3.

All the negotiation is managed by the NEGOTIATION MASTER, who decides how much negotiation is needed, and what to do in case

some discrepancy (incompatibility) is discovered: either to try alternative options or to abort the connection. Upon completion of successful negotiation, the NEGOTIATION MASTER sends either #9 (RINGING) only if it is the ANSWERER and if this is an initial connection, else it sends #6 (READY-FOR-DATA), and probably inquires with #8 about the readiness of the other party. The inquiries (#8) before the successful completion of the negotiation are ignored. However, these inquiries after the first RINGING (#9) and before the first READY (#6) are needed to keep the ANSWERER ringing.

Note that the negotiation process can be shortened by using the VERSION option, as shown in the examples that follow.

ON RENEGOTIATION

At any stage after links are agreed upon, either party might request a RENEGOTIATION. If the request is approved by the other party, either party might become the NEGOTIATION MASTER, depending on the type of renegotiation request. When renegotiation starts, no previously negotiated agreements (except LINK numbers) hold, and all items have to be renegotiated from scratch. Note that renegotiation may entirely replace the negotiation phase and allows the CALLER to be the NEGOTIATION MASTER.

Upon issuance (or reception) of RENEGOTIATION REQUEST, all data messages are ignored until the positive indication of the successful completion of the renegotiation (#6).

After the completion of renegotiation, the frame-count (see the section on MESSAGE-HEADER) may be reset to zero.

THE HEADER OF DATA MESSAGES

Data messages are the messages which contain vocoded speech. The first 32 bits of each data message is the MESSAGE-HEADER, which carries sequence and timing information as described below.

For each vocoding scheme a "FRAME" is defined as the transmission interval (as agreed upon at the negotiation stage in <WHAT#6>). Since this interval is defined by the number of samples, its duration can be found by multiplying the sampling period <WHAT#2> by the interval length (in samples) <WHAT#6>. For example, in V1 the sampling period is 150 microseconds and the transmission interval is 128 samples, which yields:

128*150 microseconds = 19.2 milliseconds.

The data describing a FRAME is called a PARCEL. Each parcel has a

serial number. The first parcel created after the completion of the negotiation (or every RENEGOTIATION) has the serial number zero. Each message contains an integral number of parcels.

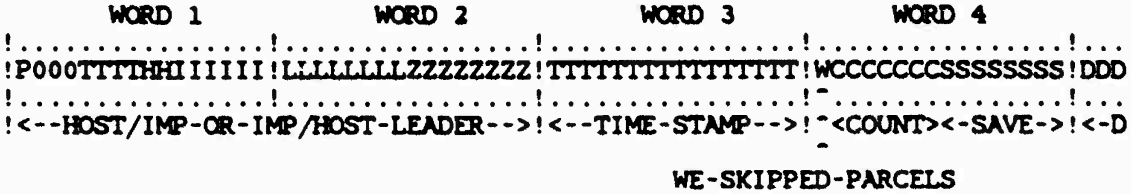
The serial number of the first parcel in the message is put in the first 16 bits of the message and is referred to as the MESSAGE-TIME-STAMP. Note that this time stamp is synchronized with the data stream. Note also that these 16 bits are actually the third word of the message, following the 2 words used as IMP-to-HOST leader (see BBN Report 1822).

The next bit in the header is the WE-SKIPPED-PARCELS bit, which is described later. The next 7 bits tell how many parcels there are in the message; this number is called the COUNT, or the PARCEL-COUNT.

Note that if message number N has the time stamp T(N) and the count C(N), then T(N+1) must be greater than or equal to T(N)+C(N). Usually T(N+1) = T(N)+C(N), unless the XMTR decided not to send some parcels due to silence. If this happens then the WE-SKIPPED-PARCELS bit is set to ONE, else it is set to ZERO. Hence, if T(N+1) is found by the RCVR to be greater than T(N)+C(N) and the WE-SKIPPED-PARCELS is zero, some message must be lost.

Note that by definition the time stamps on messages monotonically increase, except for wrap-around.

The message header structure is illustrated by the following diagram:



P = PRIORITY (one bit = 1)
T = MESSAGE TYPE (4 bits = 0011)
L = link ("L" OR "K", 8 bits, greater than 337 octal)
D = data bits (from here to the end of the message)

ZZZZZZZZ = 8 ZERO bits
HHIIIIII = HOST (8 bits, destination or source)
CCCCCCC = parcel COUNT (7 bits)
SSSSSSSS = 8 bits saved for future applications
TTTTTTTTTTTTTTTT = TIME STAMP (16 bits)

The first parcel sent by either party after the NEGOTIATION or RENEGOTIATION should have the serial number set to zero.

During silence periods, the XMTR might send a "6" or "7" message periodically. If it does not do so, the RCVR might interrogate the livelihood of the XMTR by sending periodically "8" ("ARE-YOU-THERE?") or #10 (ECHO-REQUEST) messages.

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

3. THE LPC DATA PROTOCOL

The DATA sent at each transmission interval is called a PARCEL.

Network messages always contain an integral number of PARCELs.

There are two independent issues in the coding. One is, obviously, the acoustic coding, i.e., which parameters have to be transmitted. SIMPLE acoustic coding is sending all the parameters at every transmission interval. OPTIMIZED acoustic coding sends only as little as acoustically needed. DELCO is an example of OPTIMIZED acoustic coding.

In this document only the format of the SIMPLE acoustic coding is defined.

All the transmitted parameters are sent as pointers into agreed-upon tables. These tables are defined as two lists of values. The transmitter table {X(J)} is used in the following way: The value V is coded as the code J if X(J-1) < V =< X(J). The receiver table {R(J) is used to retrieve the value R(J) if the code J was received. X(-1) is implicitly defined as minus-infinity, and X(Jmax) is explicitly defined as plus-infinity.

For each parameter, {X(J)} and {R(J)} may be defined independently.

The second coding issue is the information coding technique. The SIMPLE (information-wise) way of sending the information is to use binary coding for the codes representing the parameters. The OPTIMIZED way is to compute distributions for each parameter and to define the appropriate coding. It is very probable that the PITCH and GAIN will be decoded absolutely in the first PARCEL of each message, and incrementally thereafter.

At present, only the SIMPLE (information-wise) coding is used.

The details of the LPC data protocol and its Tables-Set-#1 can be found in Appendix 1.

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

Following is the definition for the format of the SIMPLE-SIMPLE coding, according to Tables-Set-#1:

For each parcel:

PITCH	6 bits	(PITCH=0 for UNVOICED)
GAIN	5 bits	
I(1)	7 bits	
I(2)	7 bits	
I(3)	6 bits	
I(4)	6 bits	
I(5)	5 bits	
I(6)	5 bits	
I(7)	5 bits	
I(8)	5 bits	
I(9)	5 bits	
I(10)	5 bits	

where each of the I(j) is an index for inverse sine coding. If K(j)=arcsin(Theta(j)) and N bits are assigned for its transmission, then I(j)=(Theta(j)/Pi)*2**N.

Hence at each transmission interval (128 samples times 150 microseconds) 67 bits are sent, which results in a data rate of 3490 bps. Since this bandwidth is well within the capabilities of the network, SIMPLE-SIMPLE coding is used, which requires the least computation by the hosts. Note that this data rate is a peak rate, without the use of silence.

4. EXAMPLES FOR THE CONTROL PROTOCOL

Here is an example for a connection:

(377)	C: 1,<WHO>,<WHOM>,340	Please talk to me on 340/341.
(340)	A: 2,1	I refuse, since I'm busy.

Another example:

(377)	C: 1,<WHO>,<WHOM>,360	Please talk to me on 360/361.
(360)	A: 6,350	OK. You talk to me on 350/351.
(350)	C: 1,<WHO>,<WHOM>	I want to talk to you.
(360)	A: 3,1,1,2	Can you do CVSD? (ANSWERER tries to be the NEGOTIATION MASTER)
(350)	C: 12,1	I want to be it.
(360)	A: 13,1	That's OK with me.
(350)	C: 3,1,1,2	Can you do CVSD?
(360)	A: 5,1,1	No, but I can do LPC.
(350)	C: 3,1,1,3	Can you do RELP?
(360)	A: 5,1,1	No, but I can do LPC.
(350)	C: 3,1,1,1	How about LPC?
(360)	A: 4,1,1	LPC is fine with me.
(350)	C: 3,2,1,150	Can you use 150 microseconds sampling?
(360)	A: 4,2,150	I can use 150 microseconds.
(350)	C: 3,4,3,976,1040,2016	Can you use 976, 1040, or 2016 bits/msg?
(360)	A: 4,4,976	I can use 976.
(350)	C: 3,5,1,10	Can you send 10 coefficients?
(360)	A: 4,5,10	I can send 10.

(350)	C: 3,6,1,64	Can you use a 64 sample transmission?
(360)	A: 4,6,64	I can use 64.
(350)	C: 3,7,2,1,2	SIMPLE or OPTIMIZED acoustic coding?
(360)	A: 4,7,2	OPTIMIZED!
(350)	C: 3,8,1,1	Can you do SIMPLE info coding?
(360)	A: 4,8,1	I can do SIMPLE.
(350)	C: 3,9,1,58	mu = 0.90625?
(360)	A: 4,9,58	Fine with me.
(350)	C: 3,10,1	Table set #1?
(360)	A: 4,10,1	Of course!
(350)	C: 6	I am ready. (Note: No "RINGING" sent)
(350)	C: 8	And you?
(360)	A: 6	I am ready, too.
	.....	Data is exchanged now,
	.....	on 351 and 361.
(350)	C: 10,1234	Echo it, please.
(360)	A: 11,1234	Here it comes!
	.....	
(360)	A: 10,3333	Now ANSWERER wants to measure
(350)	C: 11,3333	...the delays, too.
	.....	
(???)	X: 2,3	Termination by either user.

Another example:

(377)	C: 1,<WHO>,<WHOM>,360	Please talk to me on 360/361.
(360)	A: 6,340	Fine. You send on 340/341.
(340)	C: 1,<WHO>,<WHOM>	I want to talk to you.
(360)	A: 3,3,1,1	Can you use V1?
(340)	C: 4,3,1	Yes, V1 is OK.
(360)	A: 3,4,1,1984	Can you use up to 1984 bits/msg?
(340)	C: 5,4,976	No, but I can use 976.
(360)	A: 3,4,1,976	Can you use up to 976 bits/msg?
(340)	C: 4,4,976	I can use 976.
(360)	A: 9	Ringing (note how short this negotiation is!!).
	.....	
(340)	C: 8	Still there?
(360)	A: 9	Still ringing.
	.....	
(340)	C: 8	Still there?
(360)	A: 9	Still ringing.
	.....	
(340)	C: 8	How about it?
(360)	A: 9	Still ringing.
(340)	C: 2	Forget it! (No reason given.)

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

APPENDIX 1

THE DEFINITION OF:

TABLES-SET-#1

by

John D. Markel

Speech Communication Research Laboratory

Santa Barbara, California

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

TABLES-SET-#1

This set includes tables for:

PITCH - 64 values, PITCH table
GAIN - 32 values, GAIN table
I(1) - 128 values, INDEX7 table
I(2) - 128 values, INDEX7 table
I(3) - 64 values, INDEX6 table
I(4) - 64 values, INDEX6 table
I(5) - 32 values, INDEX5 table
I(6) - 32 values, INDEX5 table
I(7) - 32 values, INDEX5 table
I(8) - 32 values, INDEX5 table
I(9) - 32 values, INDEX5 table
I(10) - 32 values, INDEX5 table

These tables are defined specifically for a sampling period of 150 microseconds.

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

GENERAL COMMENTS

The following tables are arranged in three columns, {X(j)}, {j}, and {R(j)}. Note that the entries in the {X(j)} column are half a step off the other columns. This is to indicate that INTERVALS from X-domain (pitch, gain, and the Ks) are mapped into CODES {j}, which are transmitted over the network, to be translated by the receiver into the {R(j)}. These intervals are defined as OPEN-CLOSE intervals. For example, the PITCH value (at the transmitter) of 4131 belongs to the interval "(4024,4131]", hence it is coded as j=6 which is mapped by the receiver to the value 21. Similarly, the value of 2400 for INDEX7 is found to belong to the interval "(2009,2811]", coded into the CODE 3 and mapped back into 2411.

Note that if N bits are used by a certain CODE, then there are 2**N+1 entries in the X-table, but only 2**N entries in the R-table.

The transformation values used for PITCH, GAIN, and the K-parameters (in the X- and R-tables) are as defined in NSC Note 42.

Values above and below the range of the X-table are mapped into the maximum and minimum table indices, respectively.

Note that R(J) of INDEX5 is identical to R(2J) of INDEX6, and that R(J) of INDEX6 is identical to R(2J) of INDEX7. Therefore, it is possible to store only the R-table of INDEX7, without the R-tables of INDEX5 and INDEX6.

In the SPS-41 implementation there is no need to store any R-table for the K-parameters. The transmitted index can be used directly (with the appropriate scaling) as an index into the SPS built-in TRIG tables.

COMMENTS ON THE PITCH TABLE

The level J=0 defines the UNVOICED condition. The receiver maps it into the number of samples per frame (here 128).

This PITCH table differs significantly from previous tables and supersedes the table published in NSC Note 36. Details of the calculation of the table can be found in NSC Note 42. Immediate questions should be referred to John Markel.

COMMENTS ON THE GAIN TABLE

The level J=0 defines absolute silence.

This table is designed for a maximum of 12-bit A/D input, and allows for a dynamic range of 43.5 dB.

NSC Notes 36, 45, 56 and 58 supply background for the GAIN table. Gain is the energy of the pre-emphasized, windowed signal.

This table is the NEW GAIN table. NSC Notes 56 and 58 explain the reasoning behind the NEW GAIN.

COMMENTS ON THE INDEX7 TABLE

Positive values are coded into the range [0-63, decimal]. Negative values are coded into the 7-bits two's complement of the codes of their absolute value [65-127, decimal].

Note that all values -403 < V < 403 are coded as (and mapped into) 0. Note also that the code -64 (100 octal) is never used.

In SPS-41 implementation, the R-table is not needed, since TRIG(2J) is the needed value R(J).

COMMENTS ON THE INDEX6 TABLE

Positive values are coded into the range [0-31, decimal]. Negative values are coded into the 6-bits two's complement of the codes of their absolute values [33-63, decimal].

Note that all values -805 < V < 805 are coded as (and mapped into) 0. Note also that the code -32 (40 octal) is never used.

In SPS-41 implementation, the R-table is not needed, since TRIG(4J) is the needed value R(J).

COMMENTS ON THE INDEX5 TABLE

Positive numbers are coded into the range [0-15, decimal]. Negative numbers are coded into the 5-bits two's complement of their absolute values, i.e., [17-31, decimal].

Note that all values -1609 < V < 1609 are coded as (and mapped into) 0. Note also that the code -16 (20 octal) is never used.

In SPS-41 implementation, the R-table is not needed, since TRIG(8J) is the needed value R(J).

THE PITCH TABLE (as of 10-29-74)

X(J)	J	R(J)	X(J)	J	R(J)	X(J)	J	R(J)
0	0	128*	6002	21	33	10770	42	61
0	1	18	6168	22	34	11080	43	63
3630	2	19	6338	23	35	11399	44	65
3724	3	19	6515	24	36	11728	45	67
3821	4	20	6696	25	37	12067	46	69
3921	5	20	6883	26	38	12417	47	71
4024	6	21	7075	27	39	12776	48	73
4131	7	22	7274	28	40	13147	49	75
4240	8	22	7478	29	41	13529	50	77
4353	9	23	7689	30	43	13922	51	80
4469	10	24	7905	31	44	14327	52	82
4588	11	24	8129	32	45	14745	53	85
4711	12	25	8359	33	47	15175	54	87
4838	13	26	8596	34	48	15618	55	90
4969	14	27	8840	35	50	16075	56	93
5104	15	27	9092	36	51	16545	57	95
5242	16	28	9351	37	53	17029	58	98
5385	17	29	9618	38	54	17529	59	101
5533	18	30	9894	39	56	18043	60	104
5684	19	31	10177	40	57	18572	61	107
5841	20	32	10469	41	59	19118	62	111
6002			10770			19681	63	114
						infinity		

Note: This table has only 58 different intervals defined, since 5 values are repeated in the R(j) table.

* This value is the "Transmission Interval" (measured in samples) as defined in item #6 of the NEGOTIATION.

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

THE GAIN TABLE (as of 9-17-75)

X(J)	J	R(J)	X(J)	J	R(J)
0			225		
	0	0		16	245
20			266		
	1	20		17	289
22			315		
	2	24		18	342
26			372		
	3	28		19	404
30			439		
	4	33		20	478
36			519		
	5	39		21	565
42			614		
	6	46		22	667
50			725		
	7	54		23	789
59			857		
	8	64		24	932
70			1013		
	9	76		25	1101
83			1197		
	10	90		26	1301
98			1415		
	11	106		27	1538
116			1672		
	12	126		28	1818
137			1976		
	13	148		29	2148
161			2335		
	14	175		30	2539
191			2760		
	15	207		31	3000
255			infinity		

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

INDEX7 TABLE (as of 9-23-74)

X(J)	J	R(J)	X(J)	J	R(J)	X(J)	J	R(J)
0			15800			27897		
	0	0		21	16151		42	28106
402			16500			28311		
	1	804		22	16846		43	28511
1206			17190			28707		
	2	1608		23	17531		44	28899
2009			17869			29086		
	3	2411		24	18205		45	29269
2811			18538			29448		
	4	3212		25	18868		46	29622
3612			19195			29792		
	5	4011		26	19520		47	29957
4410			19841			30118		
	6	4808		27	20160		48	30274
5205			20475			30425		
	7	5602		28	20788		49	30572
5998			21097			30715		
	8	6393		29	21403		50	30853
6787			21706			30986		
	9	7180		30	22006		51	31114
7571			22302			31238		
	10	7962		31	22595		52	31357
8351			22884			31471		
	11	8740		32	23170		53	31581
9127			23453			31686		
	12	9512		33	23732		54	31786
9896			24008			31881		
	13	10279		34	24279		55	31972
10660			24548			32058		
	14	11039		35	24812		56	32138
11417			25073			32214		
	15	11793		36	25330		57	32286
12167			25583			32352		
	16	12540		37	25833		58	32413
12910			26078			32470		
	17	13279		38	26320		59	32522
13646			26557			32568		
	18	14010		39	26791		60	32610
14373			27020			32647		
	19	14733		40	27246		61	32679
15091			27467			32706		
	20	15447		41	27684		62	32729
15800			27897			32746		
							63	32758
						infinity		

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

INDEX6 TABLE (as of 9-23-74)

X(J)	J	R(J)	X(J)	J	R(J)
0			22595		
	0	0		16	23170
804			23732		
	1	1608		17	24279
2411			24812		
	2	3212		18	25330
4011			25833		
	3	4808		19	26320
5602			26791		
	4	6393		20	27246
7180			27684		
	5	7962		21	28106
8740			28511		
	6	9512		22	28899
10279			29269		
	7	11039		23	29622
11793			29957		
	8	12540		24	30274
13279			30572		
	9	14010		25	30853
14733			31114		
	10	15447		26	31357
16151			31581		
	11	16846		27	31786
17531			31972		
	12	18205		28	32138
18868			32286		
	13	19520		29	32413
20160			32522		
	14	20788		30	32610
21403			32679		
	15	22006		31	32729
22595			infinity		

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

INDEX5 TABLE (as of 9-23-74)

X(J)	J	R(J)	X(J)	J	R(J)
0	0	0	22006	8	23170
1608	1	3212	24279	9	25330
4808	2	6393	26320	10	27246
7962	3	9512	28106	11	28899
11039	4	12540	29622	12	30274
14010	5	15447	30853	13	31357
16846	6	18205	31786	14	32138
19520	7	20788	32413	15	32610
22006			infinity		

NWG/RFC 741 DC 22 Nov 77 42444
Specifications for the Network Voice Protocol (NVP)

APPENDIX 2

IMPLEMENTATION RECOMMENDATIONS

(1) It is recommended that the priority-bit be turned ON in the HOST/IMP header.

(2) It is recommended that in all abbreviations, "R" be used for Receiver and "X" for Transmitter.

(3) The following identifiers and values are recommended for implementations:

SLNCTH 30 SILENCE-THRESHOLD.

Used for LONG-SILENCE definition. See below. Measured in the same units as GAIN, in its X-table.

TBS 1.000 sec TIME-BEGIN-SILENCE.

LONG-SILENCE is declared if GAIN<SLNCTH for more than TBS.

TAS 0.500 sec TIME-AFTER-SILENCE.

A delay introduced by the receiver after the end of LONG-SILENCE, before restarting the playback.

TES 0.150 sec TIME-END-SILENCE.

The amount of time the transmitter backs up at the end of a LONG-SILENCE in order to ensure a smooth transition back to speech.

TRI 2.000 sec TIME-RESPONSE-INITIAL.

Time for waiting for response for an initial call (#1 and #3). The initial call is repeated every TRI until an answer arrives, or until TRIGU expires.

TRIGU 20.000 sec TIME-RESPONSE-INITIAL-GIVEUP.

If no response to an initial call is received within TRIGU after the FIRST initial call, the system gives up, assuming the other system is down.

TRQ 1.000 sec TIME-RESPONSE-INQUIRY.

If no response to an inquiry (#8) is received within TRQ, the inquiry is repeated.

TRQGU 10.000 sec TIME-RESPONSE-INQUIRY-GIVEUP.

If no response to an inquiry is received within TRQGU from the FIRST inquiry, the system gives up, assuming the other system is down.

TBDA 3.000 sec TIME-BETWEEN-DATA-ARRIVAL.

If no data arrives within TBDA, an INQUIRY (#8) is sent. This repeats every TBDA.

TNR 2.000 sec TIME-NOT-READY.

If the other system is in the NOT-READY (#7) state for more than TNR, an INQUIRY (#8) is sent. This repeats every TNR.

TNRGU 10.000 sec TIME-NOT-READY-GIVEUP.

If the other system is in the NOT-READY (#7) state for more than TNRGU, then the system gives up, assuming the other system is down.

TBIN 3.000 sec TIME-BUFFER-IN.

The input buffer size is equivalent to the time period TBIN (and its size is the DATA-RATE multiplied by the period TBIN). If the INPUT QUEUE ever gets to be longer than TBIN, data is discarded.

TBOUT 3.000 sec TIME-BUFFER-OUT.

The output buffer size is equivalent to the time period TBOUT (and its size is the DATA-RATE multiplied by the period TBOUT). If the OUTPUT QUEUE ever gets to be longer than TBOUT, data is discarded.

REFERENCES

Bolt Beranek & Newman, Inc., Report No. 1822, Interface Message Processor: Specifications for the Interconnection of a Host and an IMP.

NSC Note 42 (in progress).

NSC Note 36, Proposal for NSC-LPC Coding/Decoding Tables, by J. D. Markel, Speech Communications Research Laboratory, Inc., July 20, 1974.

NSC Note 45, Everything You Always Wanted to Know about Gain, by E. Randolph Cole, USC/Information Sciences Institute, October 11, 1974.

NSC Note 56, Nothing to Lose, but Lots to Gain, by John Makhoul and Lynn Cosell, Bolt Beranek & Newman, Inc., March 10, 1975.

NSC Note 58, Gain Again, by Randy Cole, USC/Information Sciences Institute. March 12, 1975.

Reliable Data Protocol

RFC-908

David Velten

Robert Hinden

Jack Sax

BBN Communications Corporation

July 1984

Status of This Memo

This RFC specifies a proposed protocol for the ARPA Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

RDP Specification

Table of Contents

RDP Specification

FIGURES

CHAPTER 1

Introduction

The Reliable Data Protocol (RDP) is designed to provide a reliable data transport service for packet-based applications such as remote loading and debugging. The protocol is intended to be simple to implement but still be efficient in environments where there may be long transmission delays and loss or non-sequential delivery of message segments.

Although this protocol was designed with applications such as remote loading and debugging in mind, it may be suitable for other applications that require reliable message services, such as computer mail, file transfer, transaction processing, etc.

Some of the concepts used come from a variety of sources. The authors wish credit to be given to Eric Rosen, Rob Gurwitz, Jack Haverty, and to acknowledge material adapted from "RFC-793, The Transmission Control Protocol", edited by Jon Postel. Thanks to John Linn for the checksum algorithm.

CHAPTER 2

General Description

2.1 Motivation

RDP is a transport protocol designed to efficiently support the bulk transfer of data for such host monitoring and control applications as loading/dumping and remote debugging. It attempts to provide only those services necessary, in order to be efficient in operation and small in size. Before designing the protocol, it was necessary to consider what minimum set of transport functions would satisfy the requirements of the intended applications.

The following is a list of requirements for such a transport protocol:

- Reliable delivery of packets is required. When loading or dumping a memory image, it is necessary that all memory segments be delivered. A 'hole' left in the memory image is not acceptable. However, the internet environment is a lossy one in which packets can get damaged or lost. So a positive acknowledgement and retransmission mechanism is a necessary component of the protocol.

- Since loading and dumping of memory images over the internet involves the bulk transfer of large amounts of data over a lossy network with potentially long delays, it is necessary that the protocol move data efficiently. In particular, unnecessary retransmission of segments should be avoided. If a single segment has been lost but succeeding segments correctly received, the protocol should not require the retransmission of all of the segments.

- Loading and dumping are applications that do not necessarily require sequenced delivery of segments, as long as all segments eventually are delivered. So the protocol need not force sequenced delivery. For these types of applications, segments may be delivered in the order in which they arrive.

- However, some applications may need to know that a particular piece of data has been delivered before sending the next. For example, a debugger will want to know that a command inserting a breakpoint into a host memory image has been delivered before sending a "proceed" command. If those segments arrived out of sequence, the intended results would not be achieved. The protocol should allow a user to optionally specify that a connection must deliver segments in sequence-number order.

- The loading/dumping and debugging applications are well-defined and lend themselves to easy packetization of the transferred data. They do not require a complex byte-stream transfer mechanism.

In order to combine the requirements for bulk transfers of data and reliable delivery, it is necessary to design a connection-oriented protocol using a three-way handshake to synchronize sequence numbers. The protocol seems to be approaching TCP in complexity, so why was TCP not, in fact, chosen? The answer is that TCP has some disadvantages for these applications. In particular:

- TCP is oriented toward a more general environment, supporting the transfer of a stream of bytes between two communicating parties. TCP is best suited to an environment where there is no obvious demarcation of data in a communications exchange. Much of the difficulty in developing a TCP implementation stems from the complexity of supporting this general byte-stream transfer, and thus a significant amount of complexity can be avoided by using another protocol. This is not just an implementation consideration, but also one of efficiency.

- Since TCP does not allow a byte to be acknowledged until all prior bytes have been acknowledged, it often forces unnecessary retransmission of data. Therefore, it does not meet another of the requirements stated above.

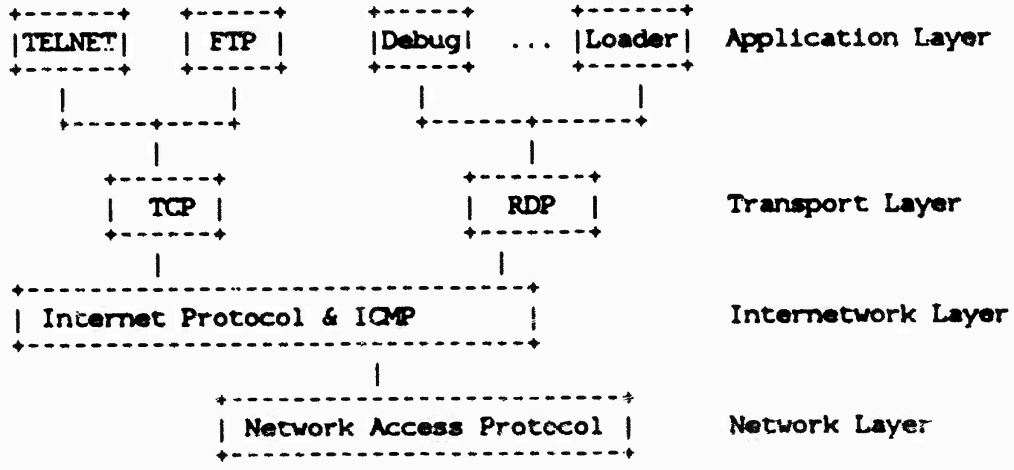
- TCP provides sequenced delivery of data to the application. If the application does not require such sequenced delivery, a large amount of resources are wasted in providing it. For example, buffers may be tied up buffering data until a segment with an earlier sequence number arrives. The protocol should not force its segment-sequencing desires on the application.

RDP supports a much simpler set of functions than TCP. The flow control, buffering, and connection management schemes of RDP are considerably simpler and less complex. The goal is a protocol that can be easily and efficiently implemented and that will serve a range of applications.

RDP functions can also be subset to further reduce the size of a particular implementation. For example, a target processor requiring down-loading from another host might implement an RDP module supporting only the passive Open function and a single connection. The module might also choose not to implement out-of-sequence acknowledgements.

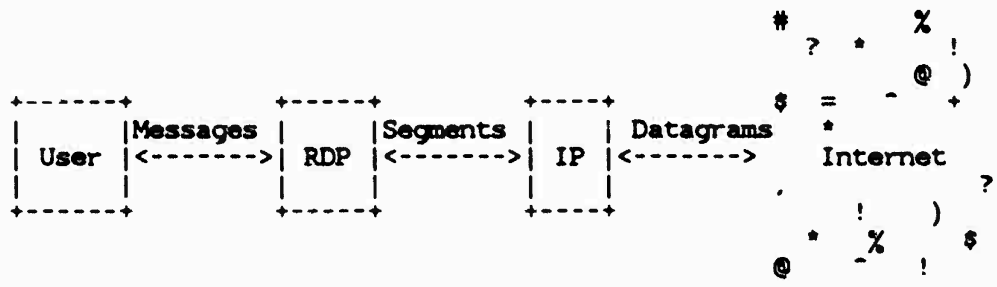
2.2 Relation to Other Protocols

RDP is a transport protocol that fits into the layered internet protocol environment. Figure 1 illustrates the place of RDP in the protocol hierarchy:



Relation to Other Protocols
Figure 1

RDP provides the application layer with a reliable message transport service. The interface between users and RDP transfers data in units of messages. When implemented in the internet environment, RDP is layered on the Internet Protocol (IP), which provides an unreliable datagram service to RDP. Data is passed across the RDP/IP interface in the form of segments. RDP uses the standard IP interface primitives to send and receive RDP segments as IP datagrams. At the internet level, IP exchanges datagrams with the network layer. An internet packet may contain an entire datagram or a fragment of a datagram.



Form of Data Exchange Between Layers
Figure 2

If internetwork services are not required, it should be possible to use the RDP without the IP layer. As long as the encapsulating protocol provides the RDP with such necessary information as addressing and protocol demultiplexing, it should be possible to run RDP layered on a variety of different protocols.

CHAPTER 3

Protocol Operation

3.1 Protocol Service Objectives

The RDP protocol has the following goals:

- RDP will provide a full-duplex communications channel between the two ports of each transport connection.
- RDP will attempt to reliably deliver all user messages and will report a failure to the user if it cannot deliver a message. RDP extends the datagram service of IP to include reliable delivery.
- RDP will attempt to detect and discard all damaged and duplicate segments. It will use a checksum and sequence number in each segment header to achieve this goal.
- RDP will optionally provide sequenced delivery of segments. Sequenced delivery of segments must be specified when the connection is established.
- RDP will acknowledge segments received out of sequence, as they arrive. This will free up resources on the sending side.

3.2 RDP Connection Management

RDP is a connection-oriented protocol in which each connection acts as a full-duplex communication channel between two processes. Segments from a sender are directed to a port on the destination host. The two 8-bit source and destination port identifiers in the RDP header are used in conjunction with the network source and destination addresses to uniquely identify each connection.

3.2.1 Opening a Connection

Connections are opened by issuing the Open request, which can be either active or passive. A passive Open request puts RDP into the Listen state, during which it passively listens for a request to open a connection from a remote site. The active Open request attempts to establish a connection with a specified port at a remote site.

The active Open request requires that a specific remote port and host address be specified with the request. The passive Open may optionally specify a specific remote port and network address, or it may specify that an open be accepted from anyone. If a specific remote port and host address were specified, an arriving request to open a connection must exactly match the specified remote port and address.

3.2.2 Ports

Valid port numbers range from 1 to 255 (decimal). There are two types of ports: "well known" ports and "allocable" ports. Well-known ports have numbers in the range 1 to 63 (decimal) and allocable ports are given numbers in the range 64 to 255.

The user, when issuing an active Open request, must specify both the remote host and port and may optionally specify the local port. If the local port was not specified, RDP will select an unused port from the range of allocable ports. When issuing a passive Open request, the user must specify the local port number. Generally, in this case the local port will be a well-known port.

3.2.3 Connection States

An RDP connection will progress through a series of states during its lifetime. The states are shown in Figure 3 and are individually described below. In Figure 3, the boxes represent the states of the RDP FSM and the arcs represent changes in state. Each arc is annotated with the event causing the state change and the resulting output.

CLOSED

The CLOSED state exists when no connection exists and there is no connection record allocated.

LISTEN

The LISTEN state is entered after a passive Open request is processed. A connection record is allocated and RDP waits for an active request to establish a connection from a remote site.

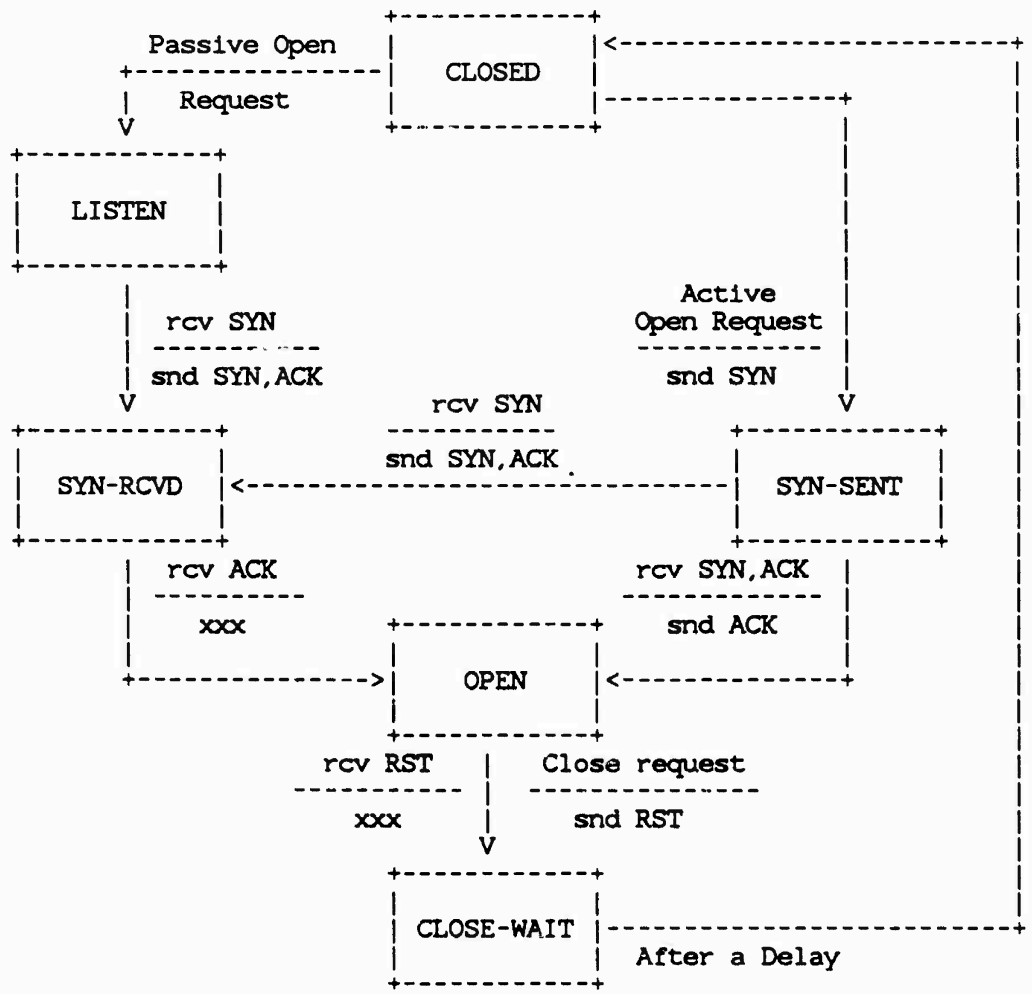
SYN-SENT

The SYN-SENT state is entered after processing an active Open request. A connection record is allocated, an initial sequence number is generated, and a SYN segment is sent to the remote site. RDP then waits in the SYN-SENT state for acknowledgement of its Open request.

SYN-RCVD

The SYN-RCVD state may be reached from either the LISTEN state or from the SYN-SENT state. SYN-RCVD is reached from the LISTEN state when a SYN segment requesting a connection is received from a remote host. In reply, the local RDP generates an initial sequence number for its side of the connection, and then sends the sequence number and an acknowledgement of the SYN segment to the remote site. It then waits for an acknowledgement.

The SYN-RCVD state is reached from the SYN-SENT state when a SYN segment is received from the remote host without an accompanying acknowledgement of the SYN segment sent to that remote host by the local RDP. This situation is caused by simultaneous attempts to open a connection, with the SYN segments passing each other in transit. The action is to repeat the SYN segment with the same sequence number, but now including an ACK of the remote host's SYN segment to indicate acceptance of the Open request.



RDP Connection State Diagram
Figure 3

OPEN

The OPEN state exists when a connection has been established by the successful exchange of state information between the two sides of the connection. Each side has exchanged and received such data as initial sequence number, maximum segment size, and maximum number of unacknowledged segments that may be outstanding. In the Open state data may be sent between the two parties of the connection.

CLOSE-WAIT

The CLOSE-WAIT state is entered from either a Close request or from the receipt of an RST segment from the remote site. RDP has sent an RST segment and is waiting a delay period for activity on the connection to complete.

3.2.4 Connection Record

The variables that define the state of a connection are stored in a connection record maintained for each connection. The following describes some of the variables that would be stored in a typical RDP connection record. It is not intended to be an implementation specification nor is it a complete description. The purpose of naming and describing some of the connection record fields is to simplify the description of RDP protocol operation, particularly event processing.

The connection record fields and their descriptions follow:

STATE

The current state of the connection. Legal values are OPEN, LISTEN, CLOSED, SYN-SENT, SYN-RCVD, and CLOSE-WAIT.

Send Sequence Number Variables:

SND.NXT

The sequence number of the next segment that is to be sent.

SND.UNA

The sequence number of the oldest unacknowledged segment.

SND.MAX

The maximum number of outstanding (unacknowledged) segments that can be sent. The sender should not send more than this number of segments without getting an acknowledgement.

SND.ISS

The initial send sequence number. This is the sequence number that was sent in the SYN segment.

Receive Sequence Number Variables:

RCV.CUR

The sequence number of the last segment received correctly and in sequence.

RCV.MAX

The maximum number of segments that can be buffered for this connection.

RCV.IRS

The initial receive sequence number. This is the sequence number of the SYN segment that established this connection.

RCVDSEQNO[n]

The array of sequence numbers of segments that have been received and acknowledged out of sequence.

Other Variables:

CLOSEWAIT

A timer used to time out the CLOSE-WAIT state.

SBUF.MAX

The largest possible segment (in octets) that can legally be sent. This variable is specified by the foreign host in the

SYN segment during connection establishment.

RBUF.MAX

The largest possible segment (in octets) that can be received. This variable is specified by the user when the connection is opened. The variable is sent to the foreign host in the SYN segment.

Variables from Current Segment:

SEG.SEQ

The sequence number of the segment currently being processed.

SEG.ACK

The acknowledgement sequence number in the segment currently being processed.

SEG.MAX

The maximum number of outstanding segments the receiver is willing to hold, as specified in the SYN segment that established the connection.

SEG.BMAX

The maximum segment size (in octets) accepted by the foreign host on a connection, as specified in the SYN segment that established the connection.

3.2.5 Closing a Connection

The closing of a connection can be initiated by a Close request from the user process or by receipt of an RST segment from the other end of the connection. In the case of the Close request, RDP will send an RST segment to the other side of the connection and then enter the CLOSE-WAIT state for a period of time. While in the CLOSE-WAIT state, RDP will discard segments received from the other side of the connection. When the time-out period expires, the connection record is deallocated and the connection ceases to exist. This simple connection closing facility requires that users determine that all data has been

reliably delivered before requesting a close of the connection.

3.2.6 Detecting an Half-Open Connection

If one side of a connection crashes, the connection may be left with the other side still active. This situation is termed to be an half-open connection. For many cases, the active RDP will eventually detect the half-open connection and reset. Two examples of recovery from half-open connections are provided in sections 5.7 and 5.8. Recovery is usually achieved by user activity or by the crashed host's attempts to re-establish the connection.

However, there are cases where recovery is not possible without action by the RDP itself. For example, if all connection blocks are in use, attempts to re-establish a broken connection will be rejected. In this case, the RDP may attempt to free resources by verifying that connections are fully open. It does this by sending a NUL segment to each of the other RDPs. An acknowledgement indicates the connection is still open and valid.

To minimize network overhead, verification of connections should only be done when necessary to prevent a deadlock situation. Only inactive connections should be verified. An inactive connection is defined to be a connection that has no outstanding unacknowledged segments, has no segments in the user input or output queues, and that has not had any traffic for some period of time.

3.3 Data Communication

Data flows through an RDP connection in the form of segments. Each user message submitted with a Send request is packaged for transport as a single RDP segment. Each RDP segment is packaged as an RDP header and one or more octets of data. RDP will not attempt to fragment a large user message into smaller segments and re-assemble the message on the receiving end. This differs from a byte-stream protocol such as TCP which supports the transfer of an indeterminate length stream of data between ports, buffering data until it is requested by the receiver.

At the RDP level, outgoing segments, as they are created, are queued as input to the IP layer. Each segment is held by the sending RDP until it is acknowledged by the foreign host. Incoming segments are queued as input to the user process through the user interface. Segments are acknowledged when they have been accepted by the receiving RDP.

The receiving end of each connection specifies the maximum segment size it will accept. Any attempt by the sender to transmit a larger segment is an error. If RDP determines that a buffer submitted with a Send request exceeds the maximum size segment permitted on the connection, RDP will return an error to the user. In addition, RDP will abort a connection with an RST segment if an incoming segment contains more data than the maximum acceptable segment size. No attempt will be made to recover from or otherwise overcome this error condition.

If sequenced delivery of segments is necessary for a connection, the requirement must be stated when the connection is established. Sequenced delivery is specified when the Open request is made. Sequenced delivery of segments will then be the mode of delivery for the life of the connection.

3.4 Reliable Communication

RDP implements a reliable message service through a number of mechanisms. These include the insertion of sequence numbers and checksums into segments, the positive acknowledgement of segment receipt, and timeout and retransmission of missing segments.

3.4.1 Segment Sequence Numbers

Each segment transporting data has a sequence number that uniquely identifies it among all other segments in the same connection. The initial sequence number is chosen when the connection is opened and is selected by reading a value from a monotonically increasing clock. Each time a segment containing data is transmitted, the sequence number is incremented. Segments containing no data do not increment the sequence number. However, the SYN and NUL segments, which cannot contain data, are exceptions. The SYN segment is always sent with a unique sequence number, the initial sequence number. The NUL segment is

sent with the next valid sequence number.

3.4.2 Checksums

Each RDP segment contains a checksum to allow the receiver to detect damaged segments. RDP uses a non-linear checksum algorithm to compute a checksum that is 32-bits wide and operates on data in units of four octets (32 bits). The area that is covered by the checksum includes both the RDP header and the RDP data area.

If a segment contains a number of header and data octets that is not an integral multiple of 4 octets, the last octet is padded on the right with zeros to form a 32-bit quantity for computation purposes. The padding zeros are not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros. The actual algorithm is described in Section 4.2.1.

3.4.3 Positive Acknowledgement of Segments

RDP assumes it has only an unreliable datagram service to deliver segments. To guarantee delivery of segments in this environment, RDP uses positive acknowledgement and retransmission of segments. Each segment containing data and the SYN and NUL segments are acknowledged when they are correctly received and accepted by the destination host. Segments containing only an acknowledgement are not acknowledged. Damaged segments are discarded and are not acknowledged. Segments are retransmitted when there is no timely acknowledgement of the segment by the destination host.

RDP allows two types of acknowledgement. A cumulative acknowledgement is used to acknowledge all segments up to a specified sequence number. This type of acknowledgement can be sent using fixed length fields within the RDP header. Specifically, the ACK control flag is set and the last acknowledged sequence number is placed in the Acknowledgement Number field.

The extended or non-cumulative acknowledgement allows the receiver to acknowledge segments out of sequence. This type of acknowledgement is sent using the EACK control flag and the

variable length fields in the RDP segment header. The variable length header fields are used to hold the sequence numbers of the acknowledged out-of-sequence segments.

The type of acknowledgement used is simply a function of the order in which segments arrive. Whenever possible, segments are acknowledged using the cumulative acknowledgement segment. Only out-of-sequence segments are acknowledged using the extended acknowledgement option.

The user process, when initiating the connection, cannot restrict the type of acknowledgement used on the connection. The receiver may choose not to implement out-of-sequence acknowledgements. On the other hand, the sender may choose to ignore out-of-sequence acknowledgements.

3.4.4 Retransmission Timeout

Segments may be lost in transmission for two reasons: they may be lost or damaged due to the effects of the lossy transmission media; or they may be discarded by the receiving RDP. The positive acknowledgement policy requires the receiver to acknowledge a segment only when the segment has been correctly received and accepted.

To detect missing segments, the sending RDP must use a retransmission timer for each segment transmitted. The timer is set to a value approximating the transmission time of the segment in the network. When an acknowledgement is received for a segment, the timer is cancelled for that segment. If the timer expires before an acknowledgement is received for a segment, that segment is retransmitted and the timer is restarted.

3.5 Flow Control and Window Management

RDP employs a simple flow control mechanism that is based on the number of unacknowledged segments sent and the maximum allowed number of outstanding (unacknowledged) segments. Each RDP connection has an associated set of flow control parameters that include the maximum number of outstanding segments for each side of a connection. These parameters are specified when the connection is opened with the Open request, with each side of the connection specifying its own parameters. The parameters are

passed from one host to another in the initial connection segments.

The values specified for these parameters should be based on the amount and size of buffers that the RDP is willing to allocate to a connection. The particular RDP implementation can set the parameters to values that are optimal for its buffering scheme. Once these parameters are set they remain unchanged throughout the life of the connection.

RDP employs the concept of a sequence number window for acceptable segment sequence numbers. The left edge of the window is the number of the last in-sequence acknowledged sequence number plus one. The right edge of the window is equal to the left edge plus twice the allowed maximum number of outstanding segments. The allowed maximum number of outstanding segments is the number of segments the transmitting RDP software is allowed to send without receiving any acknowledgement.

The flow control and window management parameters are used as follows. The RDP module in the transmitting host sends segments until it reaches the connection's segment limit specified by the receiving process. Once this limit is reached, the transmitting RDP module may only send a new segment for each acknowledged segment.

When a received segment has a sequence number that falls within the acceptance window, it is acknowledged. If the sequence number is equal to the left-hand edge (i.e., it is the next sequence number expected), the segment is acknowledged with a cumulative acknowledgement (ACK). The acceptance window is adjusted by adding one to the value of the edges. If the sequence number is within the acceptance window but is out of sequence, it is acknowledged with a non-cumulative acknowledgement (EACK). The window is not adjusted, but the receipt of the out-of-sequence segment is recorded.

When segments are acknowledged out of order, the transmitting RDP module must not transmit beyond the acceptance window. This could occur if one segment is not acknowledged but all subsequent segments are received and acknowledged. This condition will fix the left edge of the window at the sequence number of the unacknowledged segment. As additional segments are transmitted, the next segment to be sent will approach and eventually overtake the right window edge. At this point all transmission of new segments will cease until the unacknowledged segment is acknowledged.

3.6 User Interface

The user interface to RDP is implementation dependent and may use system calls, function calls or some other mechanism. The list of requests that follows is not intended to suggest a particular implementation.

OPEN Request

Opens a connection. Parameters include type (active or passive), local port, remote port, remote host address, maximum segment size, maximum number of unacknowledged segments, delivery mode (sequenced or non-sequenced). The connection id, including any allocated port number, is returned to the user.

SEND Request

Sends a user message. Parameters include connection identifier, buffer address and data count.

RECEIVE Request

Receives a user message. Parameters include connection identifier, buffer address and data count.

CLOSE Request

Closes a specified connection. The single parameter is the connection identifier.

STATUS Request

Returns the status of a connection. The parameters include the connection identifier and the address of the status buffer.

3.7 Event Processing

This section describes one possible sequence for processing events. It is not intended to suggest a particular implementation, but any actual implementation should vary from this description only in detail and not significantly in substance. The following are the kinds of events that may occur:

USER REQUESTS

Open
Send
Receive
Close
Status

ARRIVING SEGMENT

Segment Arrives

TIMEOUTS

Retransmission Timeout
Close-Wait Timeout

User request processing always terminates with a return to the caller, with a possible error indication. Error responses are given as a character string. A delayed response is also possible in some situations and is returned to the user by whatever event or pseudo interrupt mechanism is available. The term "signal" is used to refer to delayed responses.

Processing of arriving segments usually follows this general sequence: the sequence number is checked for validity and, if valid, the segment is queued and processed in sequence-number order. For all events, unless a state change is specified, RDP remains in the same state.

3.7.1 User Request Events

The following scenarios demonstrate the processing of events caused by the issuance of user requests:

Open Request

CLOSED STATE

```
Create a connection record
If none available
  Return "Error - insufficient resources"
Endif

If passive Open
  If local port not specified
    Return "Error - local port not specified"
  Endif
  Generate SND.ISS
  Set SND.NXT = SND.ISS + 1
      SND.UNA = SND.ISS
  Fill in SND.MAX, RMAX.BUF from Open parameters
  Set State = LISTEN
  Return
Endif


If active Open
  If remote port not specified
    Return "Error - remote port not specified"
  Endif
  Generate SND.ISS
  Set SND.NXT = SND.ISS + 1
      SND.UNA = SND.ISS
  Fill in SND.MAX, RMAX.BUF from Open parameters
  If local port not specified
    Allocate a local port
  Endif
  Send <SEQ=SND.ISS><MAX=SND.MAX><MAXBUF=RMAX.BUF><SYN>
  Set State = SYN-SENT
  Return (local port, connection identifier)
Endif
```

```
  LISTEN STATE
  SYN-SENT STATE
  SYN-RCVD STATE
  OPEN STATE
  CLOSE-WAIT STATE

    Return "Error - connection already open"


Close Request

  OPEN STATE

    Send <SEQ=SND.NXT><RST>
    Set State = CLOSE-WAIT
    Start TIMWAIT Timer
    Return

  LISTEN STATE

    Set State = CLOSED
    Deallocate connection record
    Return

  SYN-RCVD STATE
  SYN-SENT STATE

    Send <SEQ=SND.NXT><RST>
    Set State = CLOSED
    Return


  CLOSE-WAIT STATE

    Return "Error - connection closing"

  CLOSE STATE

    Return "Error - connection not open"
```

RDP Specification Protocol Operation

Receive Request

OPEN STATE

If Data is pending
Return with data
else
Return with "no data" indication
Endif

LISTEN STATE
SYN-RCVD STATE
SYN-SENT STATE

Return with "no data" indication

CLOSE STATE
CLOSE-WAIT STATE

Return "Error - connection not open"

Send Request

OPEN STATE

If SND.NXT < SND.UNA + SND.MAX
Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK><Data>
Set SND.NXT = SND.NXT + 1
Return
else
Return "Error - insufficient resources to send data"
Endif

LISTEN STATE
SYN-RCVD STATE
SYN-SENT STATE
CLOSE STATE
CLOSE-WAIT STATE

Return "Error - connection not open"

Status Request

Return with:

State of connection (OPEN, LISTEN, etc.)
Number of segments unacknowledged
Number of segments received not given to user
Maximum segment size for the send side of the connection
Maximum segment size for the receive side of the connection

3.7.2 Segment Arrival Events

The following scenarios describe the processing of the event caused by the arrival of a RDP segment from a remote host. The assumption is made that the segment was addressed to the local port associated with the connection record.

```
If State = CLOSED

  If RST set
    Discard segment
    Return
  Endif

  If ACK or NUL set
     Send <SEQ=SEG.ACK + 1><RST>
     Discard segment
     Return
   else
     Send <SEQ=0><RST><ACK=RCV.CUR><ACK>
     Discard segment
     Return
  Endif

Endif


If State = CLOSE-WAIT
  If RST set
     Set State = CLOSED
     Discard segment
     Cancel TIMWAIT timer
     Deallocate connection record
   else
     Discard segment
  Endif
  Return
Endif
```

RDP Specification Protocol Operation

```
If State = LISTEN

  If RST set
    Discard the segment
    Return
  Endif

  If ACK or NUL set
    Send <SEQ=SEG.ACK + 1><RST>
    Return
  Endif

  If SYN set
    Set RCV.CUR = SEG.SEQ
        RCV.IRS = SEG.SEQ
        SND.MAX = SEG.MAX
        SBUF.MAX = SEG.BMAX
    Send <SEQ=SND.ISS><ACK=RCV.CUR><MAX=RCV.MAX><BUFMAX=RBUF.MAX>
         <ACK><SYN>
    Set State = SYN-RCVD
    Return
  Endif

  If anything else (should never get here)
    Discard segment
    Return
  Endif
Endif

If State = SYN-SENT

  If ACK set
    If RST clear and SEG.ACK != SND.ISS
      Send <SEQ=SEG.ACK + 1><RST>
    Endif
    Discard segment; Return
  Endif

  If RST set
    If ACK set
      Signal "Connection Refused"
      Set State =  CLOSED
      Deallocate connection record
    Endif
    Discard segment
    Return
  Endif
```

```
  If SYN set
    Set RCV.CUR = SEG.SEQ
        RCV.IRS = SEG.SEQ
        SND.MAX = SEG.MAX
        RBUF.MAX = SEG.BMAX
    If ACK set
      Set SND.UNA = SEG.ACK
      State = OPEN
      Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
     else
      Set State = SYN-RCVD
      Send <SEQ=SND.ISS><ACK=RCV.CUR><MAX=RCV.MAX><BUFMAX=RBUF.MAX>
             <SYN><ACK>
    Endif
    Return
  Endif

  If anything else
    Discard segment
    Return
  Endif
Endif

If State = SYN-RCVD

  If RCV.IRS < SEG.SEQ =< RCV.CUR + (RCV.MAX * 2)
    Segment sequence number acceptable
   else
    Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
    Discard segment
    Return
  Endif


  If RST set
    If passive Open
       Set State = LISTEN
    else
       Set State = CLOSED
       Signal "Connection Refused"
       Discard segment
       Deallocate connection record
    Endif
    Return
  Endif
```

RDP Specification Protocol Operation

```
If SYN set
  Send <SEQ=SEG.ACK + 1><RST>
  Set State = CLOSED
  Signal "Connection Reset"
  Discard segment
  Deallocate connection record
  Return
Endif

If EACK set
   Send <SEQ=SEG.ACK + 1><RST>
   Discard segment
   Return
Endif

If ACK set
  If SEG.ACK = SND.ISS
     Set State = OPEN
   else
     Send <SEQ=SEG.ACK + 1><RST>
     Discard segment
     Return
  Endif
 else
  Discard segment
  Return
Endif

If Data in segment or NUL set
  If the received segment is in sequence
     Copy the data (if any) to user buffers
     Set RCV.CUR=SEG.SEQ
     Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
   else
     If out-of-sequence delivery permitted
        Copy the data (if any) to user buffers
     Endif
     Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK><EACK><RCVDSEQNO1>
                  ...<RCVDSEQNOn>
  Endif
 Endif

Endif
```

```
If State = OPEN

  If RCV.CUR < SEG.SEQ =< RCV.CUR + (RCV.MAX * 2)
    Segment sequence number acceptable
   else
    Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
    Discard segment and return
  Endif

  If RST set
    Set State = CLOSE-WAIT
    Signal "Connection Reset"
    Return
  Endif

  If NUL set
    Set RCV.CUR=SEG.SEQ
    Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
    Discard segment
    Return
  Endif

  If SYN set
    Send <SEQ=SEG.ACK + 1><RST>
    Set State = CLOSED
    Signal "Connection Reset"
    Discard segment
    Deallocate connection record
    Return
  Endif

  If ACK set
    If SND.UNA =< SEG.ACK < SND.NXT
      Set SND.UNA = SEG.ACK
      Flush acknowledged segments
    Endif
  Endif

  If EACK set
    Flush acknowledged segments
  Endif
```

```
  If Data in segment
   If the received segment is in sequence
     Copy the data to user buffers
     Set RCV.CUR=SEG.SEQ
     Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK>
    else
     If out-of-sequence delivery permitted
        Copy the data to user buffers
     Endif
     Send <SEQ=SND.NXT><ACK=RCV.CUR><ACK><EACK><RCVDSEQNO1>
                  ...<RCVDSEQNOn>
   Endif
  Endif
Endif
```

3.7.3 Timeout Events

Timeout events occur when a timer expires and signals the RDP. Two types of timeout events can occur, as described below:

RETRANSMISSION TIMEOUTS

```
  If timeout on segment at head of retransmission queue
     Resend the segment at head of queue
     Restart the retransmission timer for the segment
     Requeue the segment on retransmission queue
     Return
  Endif
```

CLOSE-WAIT TIMEOUTS

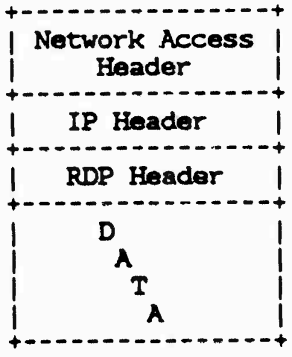
```
  Set State = CLOSED
  Deallocate connection record
  Return
```

RFC-908 July 1984

CHAPTER 4

RDP Segments and Formats

The segments sent by the application layer are encapsulated in headers by the transport, internet and network layers, as follows:



Segment Format
Figure 4

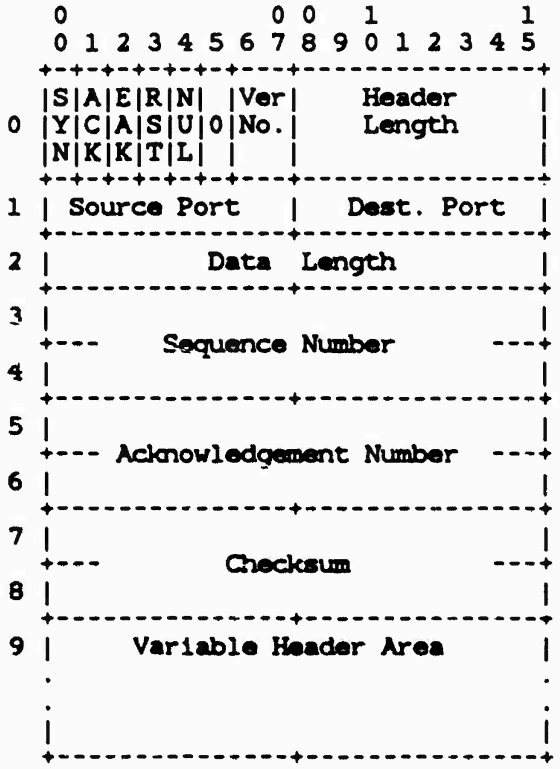
4.1 IP Header Format

When used in the internet environment, RDP segments are sent using the version 4 IP header as described in RFC791, "Internet Protocol." The RDP protocol number is ??? (decimal). The time-to-live field should be set to a reasonable value for the network.

All other fields should be set as specified in RFC-791.

4.2 RDP Header Format

Every RDP segment is prefaced with an RDP header. The format of the header is shown in Figure 5 below. The RDP header is variable in length and its size is indicated by a field in a fixed location within the header.



RDP Header Format
Figure 5

4.2.1 RDP Header Fields

Control Flags

This 8-bit field occupies the first octet of word one in the header. It is bit encoded with the following bits currently defined:

Bit #	Bit Name	Description
0	SYN	Establish connection and synchronize sequence numbers.
1	ACK	Acknowledge field significant.
2	EACK	Non-cumulative (Extended) acknowledgement.
3	RST	Reset the connection.
4	NUL	This is a null (zero data length) segment.
5		Unused.

Note that the SYN and RST are sent as separate segments and may not contain any data. The ACK may accompany any message. The NUL segment must have a zero data length, but may be accompanied by ACK and EACK information. The other control bit is currently unused and is defined to be zero.

Version Number

This field occupies bits 6-7 of the first octet. It contains the version number of the protocol described by this document. Current value is one (1).

Header Length

The length of the RDP header in units of two (2) octets, including this field. This field allows RDP to find the start of the Data field, given a pointer to the head of the segment. This field is 8 bits in length. For a segment with no variable header section, the header length field will have the value 9.

Source and Destination Ports

The Source and Destination Ports are used to identify the processes in the two hosts that are communicating with each other. The combination of the port identifiers with the source and destination addresses in the network access

protocol header serves to fully qualify the connection and constitutes the connection identifier. This permits RDP to distinguish multiple connections between two hosts. Each field is 8 bits in length, allowing port numbers from 0 to 255 (decimal).

Data Length

The length in octets of the data in this segment. The data length does not include the RDP header. This field is 16 bits in length.

Sequence Number

The sequence number of this segment. This field is 32 bits in length.

Acknowledgement Number

If the ACK bit is set in the header, this is the sequence number of the segment that the sender of this segment last received correctly and in sequence. Once a connection is established this should always be sent. This field is 32 bits in length.

Checksum

This field is a 32-bit checksum of the segment header and data. The algorithm description below includes two variables, the checksum accumulator and the checksum pointer. The checksum accumulator is an actual 32-bit register in which the checksum is formed. The checksum pointer is included for purposes of description, to represent the operation of advancing through the data four octets (32-bits) at a time. It need not be maintained in a register by an implementation.

1) The checksum pointer is set to zero, to correspond to the beginning of the area to be checksummed. The checksum accumulator is also initialized to zero before beginning the computation of the checksum.

2) The 32-bit memory word located at the address referenced by the checksum pointer is added arithmetically to the checksum accumulator. Any carry propagated out of the checksum accumulator is ignored. The checksum field itself is replaced with zeros when being added to the checksum

accumulator.

3) The checksum accumulator is rotated left one bit position. The checksum pointer is advanced to correspond to the address of the next 32-bit word in the segment.

4) Steps 2 and 3 are repeated until the entire segment has been summed. If a segment contains a number of header and data octets that is not an integral multiple of 4 octets, the last octet is padded on the right with zeros to form a 32-bit quantity for computation purposes.

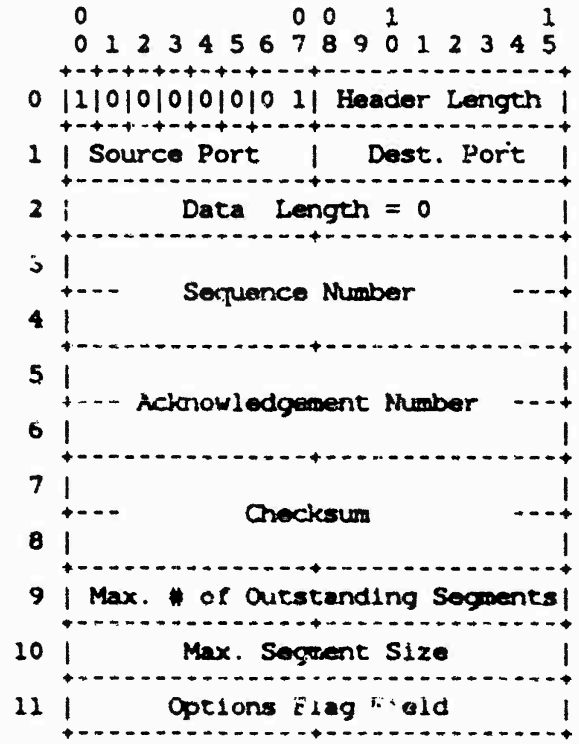
Variable Header Area

This area is used to transmit parameters for the SYN and EACK segments.

4.3 SYN Segment

The SYN is used to establish a connection and synchronize sequence numbers between two hosts. The SYN segment also contains information to inform the remote host of the maximum number of segments the local RDP is willing to accept and the maximum segment size it can accept. The SYN may be combined with an ACK in a segment but is never combined with user data.

4.3.1 SYN Segment Format

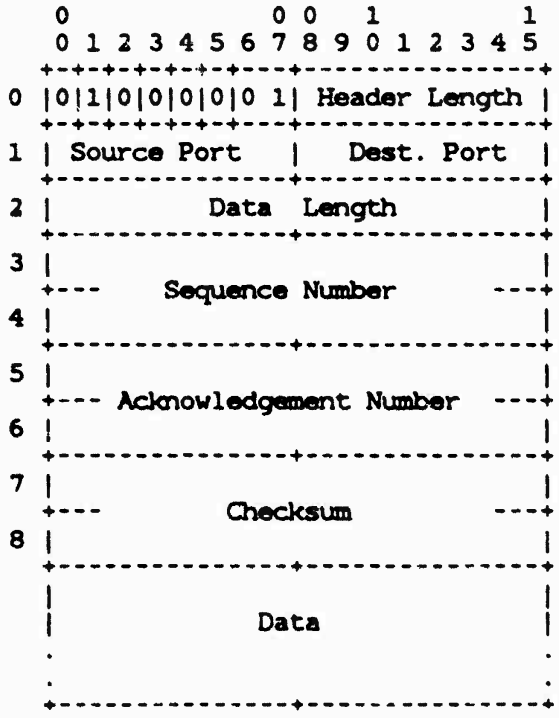


SYN Segment Format
Figure 6

4.4 ACK Segment

The ACK segment is used to acknowledge in-sequence segments. It contains both the next send sequence number and the acknowledgement sequence number in the RDP header. The ACK segment may be sent as a separate segment, but it should be combined with data whenever possible. Data segments must always include the ACK bit and Acknowledgement Number field.

4.4.1 ACK Segment Format



ACK Segment Format
Figure 7

4.3.2 SYN Segment Fields

Sequence Number

Contains the initial sequence number selected for this connection.

Acknowledgement Number

This field is valid only if the ACK flag is set. In that case, this field will contain the sequence number of the SYN segment received from the other RDP.

Maximum Number of Outstanding Segments

The maximum number of segments that should be sent without getting an acknowledgement. This is used by the receiver as a means of flow control. The number is selected during connection initiation and may not be changed later in the life of the connection.

Maximum Segment Size

The maximum size segment in octets that the sender should send. It informs the sender how big the receiver's buffers are. The specified size includes the length of the IP header, RDP header, and data. It does not include the network access layer's header length.

Options Flag Field

This field of two octets contains a set of options flags that specify the set of optional functions that are desired for this connection. The flags are defined as follows:

Bit #	Bit Name	Description
0	SDM	Sequenced delivery mode.

The sequenced delivery mode flag specifies whether delivery of segments to the user is sequenced (delivered in sequence-number order) or non-sequenced (delivered in arrival order, regardless of sequence number). A value of 0 specifies non-sequenced delivery of segments, and a value of 1 specifies sequenced delivery.

4.4.2 ACK Segment Fields

Data Length

A non-zero Data Length field indicates that there is data present in the segment.

Sequence Number

The value of the Sequence Number field is advanced to the next sequence number only if there is data present in the segment. An ACK segment without data does not use sequence number space.

Acknowledgement Number

The Acknowledgement Number field contains the sequence number of the last segment received in sequential order.

4.5 Extended ACK Segment

The EACK segment is used to acknowledge segments received out of sequence. It contains the sequence numbers of one or more segments received with a correct checksum, but out of sequence. The EACK is always combined with an ACK in the segment, giving the sequence number of the last segment received in sequence. The EACK segment may also include user data.

4.5.1 EACK Segment Format

The EACK segment has the format shown in Figure 8.

4.5.2 EACK Segment Fields

Data Length

A non-zero Data Length field indicates that there is data present in the segment.

Sequence Number

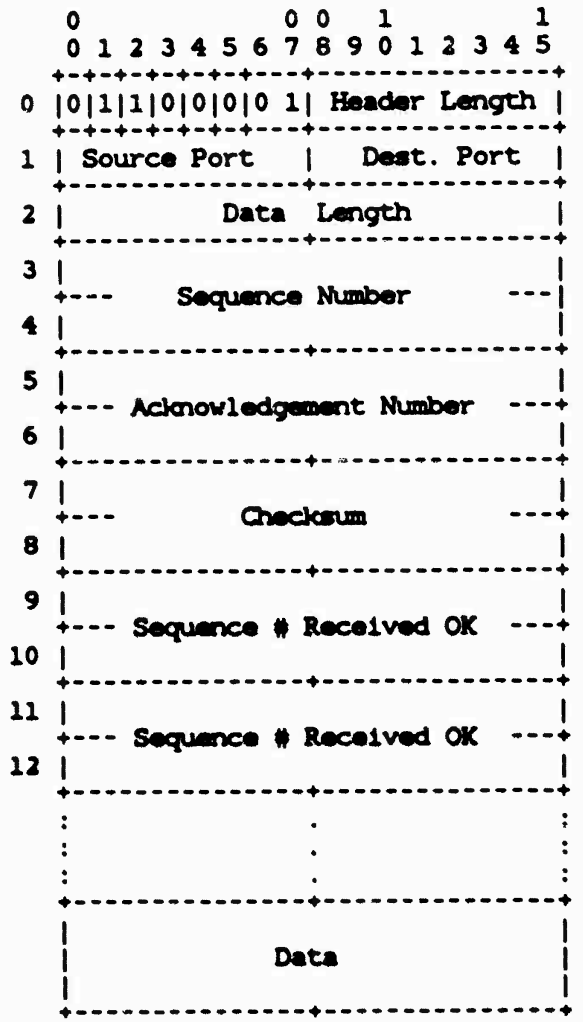
The value of the Sequence Number field is advanced to the next sequence number only if there is data present in the segment. An EACK segment without data does not use sequence number space.

Acknowledgement Number

The Acknowledgement Number field contains the sequence number of the last segment received in sequential order.

Sequence # Received OK

Each entry is the sequence number of a segment that was received with a correct checksum, but out of sequence.

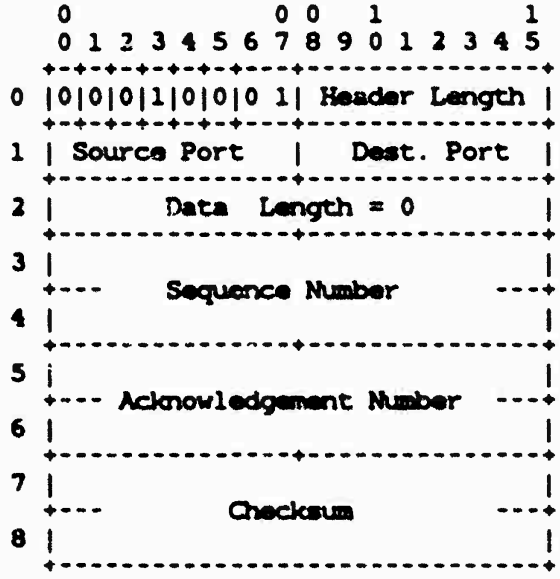


EACK Segment Format
Figure 8

4.6 RST Segment

The RST segment is used to close or reset a connection. Upon receipt of an RST segment, the sender must stop sending and must abort any unserviced requests. The RST is sent as a separate segment and does not include any data.

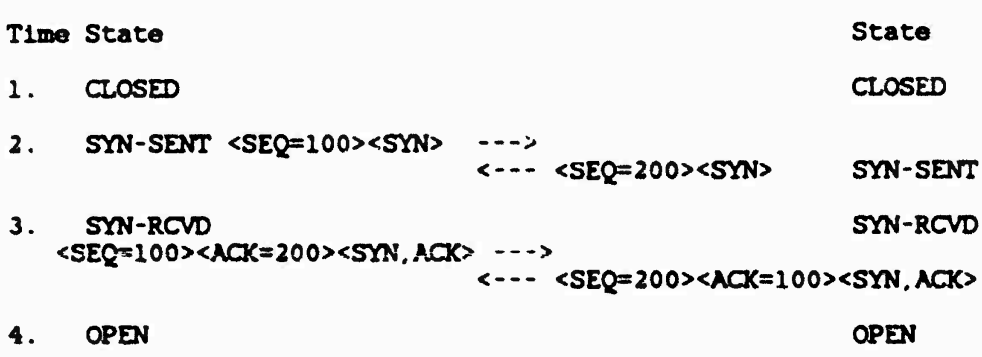
4.6.1 RST Segment Format



RST Segment Format
Figure 9

5.2 Simultaneous Connection Establishment

This is an example of two hosts trying to establishing connections to each other at the same time. Host A sends a SYN request to Host B at the same time Host B sends a SYN request to Host A.



CHAPTER 5

Examples of Operation

5.1 Connection Establishment

This is an example of a connection being established between Host A and Host B. Host B has done a passive Open and is in LISTEN state. Host A does an active Open to establish the connection.

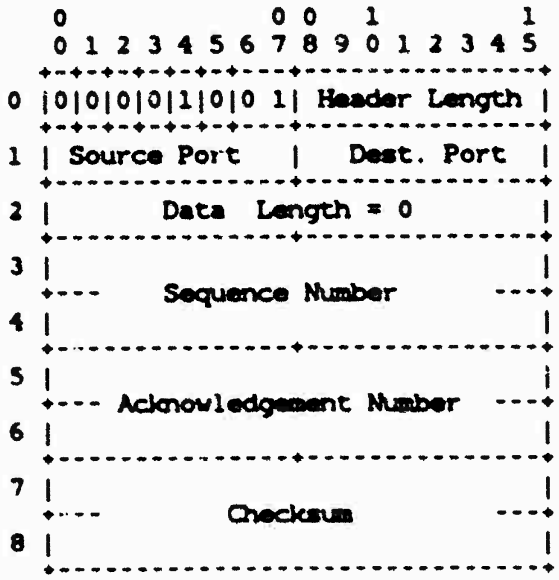
	Host A		Host B
Time	State		State
1.	CLOSED		LISTEN
2.	SYN-SENT <SEQ=100><SYN>	--->	
3.		<--- <SEQ=200><ACK=100><SYN,ACK>	SYN-RCVD
4.	OPEN <SEQ=101><ACK=200>	--->	OPEN
5.	<SEQ=101><ACK=200><Data>	--->	
6.		<--- <SEQ=201><ACK=101>	

RFC-908 July 1984

4.7 NUL Segment

The NUL segment is used to determine if the other side of a connection is still active. When a NUL segment is received, an RDP implementation must acknowledge the segment if a valid connection exists and the segment sequence number falls within the acceptance window. The segment is then discarded. The NUL may be combined with an ACK in a segment but is never combined with user data.

4.7.1 NUL segment format



NUL Segment Format
Figure 10

5.3 Lost Segments

This is an example of what happens when a segment is lost. It shows how segments can be acknowledged out of sequence and that only the missing segment need be retransmitted. Note that in this and the following examples "EA" stands for "Out of Sequence Acknowledgement."

Time	Host A		Host B
1.	<SEQ=100><ACK=200><Data>	--->	
2.		<---	<SEQ=201><ACK=100>
3.	<SEQ=101><ACK=200><Data>	(segment lost)	
4.			
5.	<SEQ=102><ACK=200><Data>	--->	
6.		<--	<SEQ=201><ACK=100><EA=102>
7.	<SEQ=103><ACK=200><Data>	--->	
8.		<---	<SEQ=201><ACK=100><EA=102,103>
9.	<SEQ=101><ACK=200><Data>	--->	
10.		<---	<SEQ=201><ACK=103>
11.	<SEQ=104><ACK=200><Data>	--->	
12.		<---	<SEQ=201><ACK=104>

5.4 Segments Received Out of Order

This an example of segments received out of order. It further illustrates the use of acknowledging segments out of order to prevent needless retransmissions.

Time	Host A		Host B
1.	<SEQ=100><ACK=200><Data>	--->	
2.		<---	<SEQ=201><ACK=100>
3.	<SEQ=101><ACK=200><Data>	(delayed)	
4.			
5.	<SEQ=102><ACK=200><Data>	--->	
6.		<---	<SEQ=201><ACK=100><EA=102>
7.	<SEQ=103><ACK=200><Data>	--->	
		--->	(delayed segment 101 arrives)
8.		<---	<SEQ=201><ACK=103>
9.	<SEQ=104><ACK=200><Data>	--->	
10.		<---	<SEQ=201><ACK=104>

5.5 Communication Over Long Delay Path

This is an example of a data transfer over a long delay path. In this example, Host A is permitted to have as many as five unacknowledged segments. The example shows that it is not necessary to wait for an acknowledgement in order to send additional data.

Time	Host A		Host B
1.	<SEQ=100><ACK=200><Data>	-1->	
2.	<SEQ=101><ACK=200><Data>	-2->	
3.	<SEQ=102><ACK=200><Data>	-3->	
		-1->	(received)
4.		<-4-	<SEQ=201><ACK=100>
5.	<SEQ=103><ACK=200><Data>	-5->	
		-2->	(received)
6.		<-6-	<SEQ=201><ACK=101>
7.	<SEQ=104><ACK=200><Data>	-7->	
		-3->	(received)
8.		<-8-	<SEQ=201><ACK=102>
	(received)	<-4-	
9	<SEQ=105><ACK=200><Data>	-9->	
		-5->	(received)
10.		<-10-	<SEQ=201><ACK=103>
	(received)	<-6-	
11.	<SEQ=106><ACK=200><Data>	-11->	
		-7->	(received)
12.		<-12-	<SEQ=201><ACK=104>
	(received)	<-8-	
13.		-9->	(received)
14.		<-13-	<SEQ=201><ACK=105>
	(received)	<-10-	
15.		-11->	(received)
16.		<-14-	<SEQ=201><ACK=106>
	(received)	<-12-	
17.	(received)	<-13-	
18.	(received)	<-14-	

5.6 Communication Over Long Delay Path With Lost Segments

This is an example of communication over a long delay path with a lost segment. It shows that by acknowledging segments out of sequence, only the lost segment need be retransmitted.

```
Time       Host A                          Host B

1. <SEQ=100><ACK=200><Data>  -1->
2. <SEQ=101><ACK=200><Data>  -2->
3. <SEQ=102><ACK=200><Data>  -3->
                              -1-> (received)
4.                           <-4-  <SEQ=201><ACK=100>
5. <SEQ=103><ACK=200><Data> (segment lost)
                              -2-> (received)
6.                           <-5-  <SEQ=201><ACK=101>
7. <SEQ=104><ACK=200><Data>  -6->
                              -3-> (received)
8.                           <-7-  <SEQ=201><ACK=102>
                  (received) <-4-
9. <SEQ=105><ACK=200><Data>  -8->
10.
                  (received) <-5-
11. <SEQ=106><ACK=200><Data> -10->
                              -6-> (received)
12.                          <-11- <SEQ=201><ACK=102><EA=104>
                  (received) <-7-
                              -8-> (received)
13.                          <-12- <SEQ=201><ACK=102><EA=104,105>
                              -10-> (received)
14.                          <-13- <SEQ=201><ACK=102><EA=104-106>
                  (received) <-11-
15. <SEQ=103><ACK=200><Data> -14->
                  (received) <-12-
16.               (received) <-13-
                              -14-> (received)
17.                          <-15- <SEQ=201><ACK=106>
18.
19.               (received) <-15-
```

5.7 Detecting a Half-Open Connection on Crash Recovery

This is an example of a host detecting a half-open connection due to the crash and subsequent restart of the host. In this example, Host A crashes during a communication session, then recovers and tries to reopen the connection. During the reopen attempt, it discovers that a half-open connection still exists and it then resets the other side. Both sides were in the OPEN state prior to the crash.

	Host A		Host B
Time			
1.	OPEN (crash!)	<--- <SEQ=200><ACK=100><ACK>	OPEN
2.	CLOSED (recover)		OPEN
3.	SYN-SENT	<SEQ=400><SYN> --->	OPEN (?)
4.	SYN-SENT (!)	<--- <SEQ=200><ACK=100><ACK>	OPEN
5.	SYN-SENT	<SEQ=101><RST> --->	OPEN (abort)
6.	SYN-SENT		CLOSED
7.	SYN-SENT <SEQ=400><SYN> --->		

5.8 Detecting a Half-Open Connection from the Active Side

This is another example of detecting a half-open connection due to the crash and restart of a host involved in a connection. In this example, host A again crashes and restarts. Host B is still active and tries to send data to host A. Since host A has no knowledge of the connection, it rejects the data with an RST segment, causing host B to reset the connection.

	Host A		Host B
Time			
1.	(crash!)		OPEN
2.	CLOSED	<--- <SEQ=200><ACK=100><Data>	OPEN
3.	CLOSED	<SEQ=101><RST> --->	(abort)
4.	CLOSED		CLOSED

APPENDIX A

Implementing a Minimal RDP

It is not necessary to implement the entire RDP specification to be able to use RDP. For simple applications such as a loader, where size of the protocol module may be important, a subset of RDP may be used. For example, an implementation of RDP for loading may employ the following restrictions:

- Only one connection and connection record is supported. This is the connection used to load the device.
- A single, well-known port is used as the loader port. Allocable ports are not implemented.
- Only the passive Open request is implemented. Active Opens are not supported.
- The sequenced delivery option is not supported. Messages arriving out of order are delivered in the order they arrive.
- If efficiency is less important than protocol size, the extended acknowledgement feature need not be supported.

INDEX

RDP Specification Examples of Operation

Network Working Group
Request for Comments: 904

D.L. Mills
April 1984

Exterior Gateway Protocol Formal Specification

0. Status of this Memo

This RFC is the specification of the Exterior Gateway Protocol (EGP). This document updates RFCs 827 and 888. This RFC specifies a standard for the DARPA community. Interactions between gateways of different autonomous systems in the ARPA-Internet must follow this protocol.

1. Introduction

This document is a formal specification of the Exterior Gateway Protocol (EGP), which is used to exchange net-reachability information between Internet gateways belonging to the same or different autonomous systems. The specification is intended as a reference guide for implementation, testing and verification and includes suggested algorithmic parameters suitable for operation over a wide set of configurations, including the ARPANET and many local-network technologies now part of the Internet system.

Specifically excluded in this document is discussion on the background, application and limitations of EGP, which have been discussed elsewhere (RFC-827, RFC-888). If, as expected, EGP evolves to include topologies not restricted to tree-structures and to incorporate full routing capabilities, this specification will be amended or obsoleted accordingly. However, it is expected that, as new features are added to EGP, the basic protocol mechanisms described here will remain substantially unchanged, with only the format and interpretation of the Update message (see below) changed.

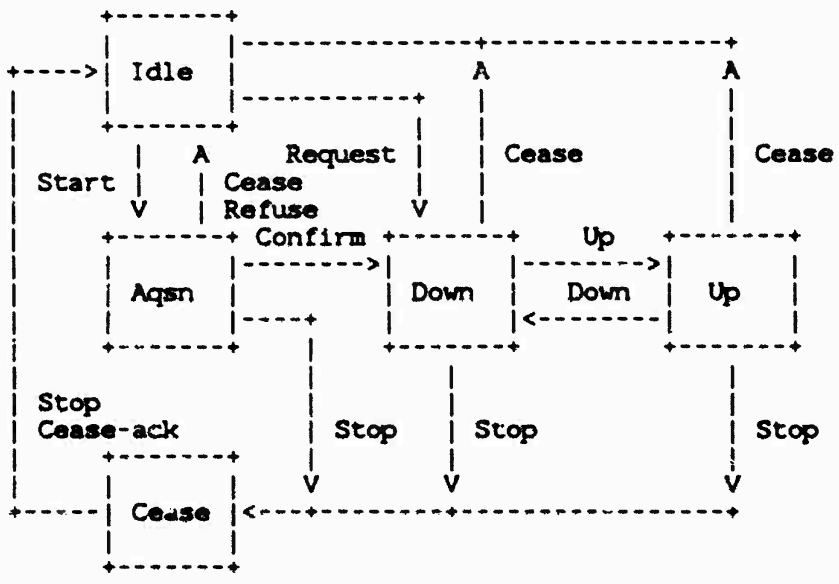
Section 2 of this document describes the nomenclature used, while Section 3 describes the state-machine model, including events, actions, parameters and state transitions. Section 4 contains a functional description of the operation of the machine, together with specific procedures and algorithms. Appendix A describes the EGP message formats, while Appendix B contains a summary of the minor differences between these and the formats described in RFC-888. Appendix C presents a reachability analysis including a table of composite state transitions for a system of two communicating EGP gateways.

1.1. Summary and Overview

EGP exists in order to convey net-reachability information between neighboring gateways, possibly in different autonomous systems. The protocol includes mechanisms to acquire neighbors, monitor neighbor reachability and exchange net-reachability information in the form of Update messages. The protocol is based on periodic polling using Hello/I-Heard-You (I-H-U) message exchanges to monitor neighbor reachability and Poll commands to solicit Update responses.

Specification of EGP is based on a formal model consisting of a

finite-state automaton with defined events, state transitions and actions. The following diagram shows a simplified graphical representation of this machine (see Section 3.4 for a detailed state transition table).



Following is a brief summary and overview of gateway operations by state as determined by this model.

Idle State (0)

In the Idle state the gateway has no resources (table space) assigned to the neighbor and no protocol activity of any kind is in progress. It responds only to a Request command or a Start event (system or operator initiated) and ignores all other commands and responses. The gateway may optionally return a Cease-ack response to a Cease command in this state.

Upon receipt of a Request command the gateway initializes the state variables as described in Section 3.1, sends a Confirm response and transitions to the Down state, if resource committments permit, or sends a Refuse response and returns to the Idle state if not. Upon receipt of a Start event it sends a Request command and transitions to the Acquistion state.

Acquisition State (1)

In the Acquisition state the gateway periodically retransmits Request commands. Upon receiving a Confirm response it initializes

the state variables and transitions to the Down state. Upon receiving a Refuse response it returns to the Idle state. The gateway does not send any other commands or responses in this state, since not all state variables have yet been initialized.

Down State (2)

In the Down state the gateway has received a Request command or a Confirm response has been received for a previously sent Request. The neighbor-reachability protocol has declared the neighbor to be down. In this state the gateway processes Request, Cease and Hello commands and responds as required. It periodically retransmits Hello commands if enabled. It does not process Poll commands and does not send them, but may optionally process an unsolicited Update indication.

Up State (3)

In the Up state the neighbor-reachability protocol has declared the neighbor to be up. In this state the gateway processes and responds to all commands. It periodically retransmits Hello commands, if enabled, and Poll commands.

Cease State (4)

A Stop event causes a Cease command to be sent and a transition to the Cease state. In this state the gateway periodically retransmits the Cease command and returns to the Idle state upon receiving a Cease-ack response or a another Stop event. The defined state transitions are designed to ensure that the neighbor does with high probability receive the Cease command and stop the protocol.

In following sections of this document document a state machine which can serve as a model for implementation is described. It may happen that implementators may deviate from this model while conforming to the protocol specification; however, in order to verify conformance to the specification, the state-machine model is intended as the reference model.

Although not mentioned specifically in this document, it should be understood that all Internet gateways must include support for the Internet Control Message Protocol (ICMP), specifically ICMP Redirect and ICMP Destination Unreachable messages.

2. Nomenclature

The following EGP message types are recognized in this document. The format of each of these messages is described in Appendix A.

Name	Function
Request	request acquisition of neighbor and/or initialize polling variables
Confirm	confirm acquisition of neighbor and/or initialize polling variables
Refuse	refuse acquisition of neighbor
Cease	request de-acquisition of neighbor
Cease-ack	confirm de-acquisition of neighbor
Hello	request neigbor reachability
I-H-U	confirm neigbor reachability
Poll	request net-reachability update
Update	net-reachability update
Error	error

EGP messages are classed as commands which request some action, responses, which are sent to indicate the status of that action, and indications, which are similar to responses, but may be sent at any time. Following is a list of commands along with their possible responses.

Command	Corresponding Responses
Request	Confirm, Refuse, Error
Cease	Cease-ack, Error
Hello	I-H-U, Error
Poll	Update, Error

The Update and Error messages are classed both as responses and indications. When sent in reply to a previous command, either of these messages is classed as a response. In some circumstances an unsolicited Update message can be sent, in which case it is classed as an indication. The use of the Error message other than as a response to a previous command is a topic for further study.

3. State Machine

This section describes the state-machine model for EGP, including the variables and constants which establish the state at any time, the events which cause the state transitions, the actions which result from these transitions and the state-transition table which defines the behavior.

3.1. State Variables

The state-machine model includes a number of state variables which establish the state of the protocol between the gateway and each of its neighbors. Thus, a gateway maintaining EGP with a number of neighbors must maintain a separate set of these state variables for each neighbor. The current state, events and actions of the state machine apply to each

neighbor separately.

The model assumes that system resources, including the set of state variables, are allocated when the state machine leaves the Idle state, either because of the arrival of a Request specifying a new neighbor addreess, or because of a Start event specifying a new neighbor address. When either of these events occur the values of the state variables are initialized as indicated below. Upon return to the Idle state all resources, including the set of state variables, are deallocated and returned to the system. Implementators may, of course, elect to dedicate resources and state variables permananently.

Included among the set of state variables are the following which determine the state transitions of the model. Initial values for all of the variables except the send sequence number S are set during the initial Request/Confirm exchange. The initial value for S is arbitrary.

Name	Function
R	receive sequence number
S	send sequence number
T1	interval between Hello command retransmissions
T2	interval between Poll command retransmissions
T3	interval during which neighbor-reachability indications are counted
M	hello polling mode
t1	timer 1 (used to control Request, Hello and Cease command retransmissions)
t2	timer 2 (used to control Poll command retransmissions)
t3	timer 3 (abort timer)

Additional state variables may be necessary to support various timer and similar internal housekeeping functions. The function and management of the cited variables are discussed in Section 4.

3.2. Fixed Parameters

This section defines several fixed parameters which characterize the gateway functions. Included is a suggested value for each parameter based on experimental implementations in the Internet system. These values may or may not be appropriate for the individual configuration.

Following is a list of time-interval parameters which control retransmissions and other time-dependent functions.

Name	Value	Description
P1	30 sec	minimum interval acceptable between successive Hello commands received
P2	2 min	minimum interval acceptable between successive Poll commands recieved
P3	30 sec	interval between Request or Cease command retransmissions
P4	1 hr	interval during which state variables are maintained in the absence of commands or responses in the Down and Up states.
P5	2 min	interval during which state variables are maintained in the absence of responses in the Acquisition and Cease states

Parameters P4 and P5 are used only if the abort-timer option is implemented. Parameter P4 establishes how long the machine will remain in the Down and Up states in the absence of commands or responses and would ordinarily be set to sustain state information while the neighbor is dumped and restarted, for example. Parameter P5 establishes how long the machine will remain in the Acquisition or Cease states in the absence of responses and would ordinarily be set in the same order as the expected value of T3 variables.

Following is a list of other parameters of interest.

Name	Active	Passive	Description
j	3	1	neighbor-up threshold
k	1	4	neighbor-down threshold

The j and k parameters establish the "noise immunity" of the neighbor-reachability protocol described later. The values in the Active column are suggested if the gateway elects to do hello polling, while the values in the Passive column are suggested otherwise.

3.3. Events

Following is a list of events that can cause state transitions in the model.

Name	Event
Up	At least j neighbor-reachability indications have been received within the last T3 seconds.
Down	At most k neighbor-reachabilitiy indications have been received within the last T3 seconds.
Request	Request command has been received.
Confirm	Confirm command has been received.
Refuse	Refuse response has been received.
Cease	Cease command has been received.
Cease-ack	Cease-ack response has been received.
Hello	Hello command has been received.
I-H-U	I-H-U response has been received.
Poll	Poll command has been received.
Update	Update response has been received.
Start	Start event has been recognized due to system or operator intervention.
Stop/t3	Stop event has been recognized due to (a) system or operator intervention or (b) expiration of the abort timer t3.
t1	Timer t1 has counted down to zero.
t2	Timer t2 has counted down to zero.

There is one special event, called a neighbor-reachability indication, which occurs when:

1. The gateway is operating in the active mode (hello polling enabled) and either a Confirm, I-H-U or Update response is received.

2. The gateway is operating in the passive mode (hello polling disabled) and either a Hello or Poll command is received with the "Up state" code in the Status field.

3.4. State Transition Table

The following table summarizes the state transitions that can occur in response to the events listed above. Transitions are shown in the form n/a, where n is the next state and a represents the action.

	0 Idle	1 Aqsn	2 Down	3 Up	4 Cease
Up	0	1	3/Poll	3	4
Down	0	1	2	2	4
Request	2/Confirm *	2/Confirm	2/Confirm	2/Confirm	4/Cease
Confirm	0/Cease **	2	2	3	4
Refuse	0/Cease **	0	2	3	4
Cease	0/Cease-ack	0/Cease-ack	0/Cease-ack	0/Cease-ack	0/Cease-ack
Cease-ack	0	1	2	3	0
Hello	0/Cease **	1	2/I-H-U	3/I-H-U	4
I-H-U	0/Cease **	1	2/Process	3/Process	4
Poll	0/Cease **	1	2	3/Update	4
Update	0/Cease **	1	2	3/Process	4
Start	1/Request	1/Request	1/Request	1/Request	4
Stop/t3	0	0	4/Cease	4/Cease	0
t1	0	1/Request	2/Hello	3/Hello	4/Cease
t2	0	1	2	3/Poll	4

Note *: The transition shown applies to the case where the neighbor-acquisition request is accepted. The transition "0/Refuse" applies to the case where the request is rejected.

Note **: The Cease action shown is optional.

3.5. State Transitions and Actions

The following table describes in detail the transitions of the state machine and the actions evoked.

Event	Next State	Message Sent	Actions
Idle State (0)			
Request	2	Confirm Hello	Initialize state variables and reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
(or)	0	Refuse	Return resources.
Cease	0	Cease-ack	Return resources.
Start	1	Request	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
Acquisition State (1)			
Request	2	Confirm Hello	Initialize state variables and reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
Confirm	2	Hello	Initialize state variables and

			reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
Refuse	0		Stop timers and return resources.
Cease	0	Cease-ack	Stop timers and return resources.
Start	1	Request	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
Stop/t3	0		Stop timers and return resources.
t1	1	Request	Reset timer t1 to P3 seconds.

Down State (2)
Note: Reset timer t3 to P4 seconds on receipt of a reachability indication.

Up	3	Poll	Reset timer t2 to T2 seconds.
Request	2	Confirm Hello	Reinitialize state variables and reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
Cease	0	Cease-ack	Stop timers and return resources.
Hello	2	I-H-U	
I-H-U	2		Process neighbor-reachability info.
Start	1	Request	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
Stop/t3	4	Cease	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
t1	2	Hello	Reset timer t1 to T1 seconds.

Up State (3)
Note: Reset timer t3 to P4 seconds on receipt of a reachability indication.

Down	2		Stop timer t2.
Request	2	Confirm Hello	Reinitialize state variables and reset timer t1 to T1 seconds and reset timer t3 to P5 seconds.
Cease	0	Cease-ack	Stop timers and return resources.
Hello	3	I-H-U	
I-H-U	3		Process neighbor-reachability info.
Poll	3	Update	
Update	3		Process net-reachability info.
Start	1	Request	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.
Stop/t3	4	Cease	Reset timer t1 to P3 seconds and reset timer t3 to P5 seconds.

t1	3	Hello	Reset timer t1 to T1 seconds.
t2	3	Poll	Reset timer t2 to T2 seconds.

Cease State (4)

Request	4	Cease	
Cease	0	Cease-ack	Stop timers and return resources.
Cease-ack	0		Stop timers and return resources.
Stop/t3	0		Stop timers and return resources.
t1	4	Cease	Reset timer t1 to P3 seconds.

4. Functional Description

This section contains detailed descriptions of the various procedures and algorithms used to manage the protocol.

4.1. Managing the State Variables

The state variables which characterize the protocol are summarized in Section 3.1. This section describes the detailed management of these variables, including sequence numbers, polling intervals and timers.

4.1.1. Sequence Numbers

All EGP commands and replies carry a sequence number. The state variable R records the last sequence number received in a command from that neighbor. The current value of R is used as the sequence number for all replies and indications sent to the neighbor until a command with a different sequence number is received from that neighbor.

Implementors are free to manage the sequence numbers of the commands sent; however, it is suggested that a separate send state variable S be maintained for each EGP neighbor and that its value be incremented just before the time an Poll command is sent and at no other times. The actions upon receipt of a response or indication with sequence number not equal to S is not specified; however, it is recommended these be discarded.

4.1.1. Polling Intervals

As part of the Request/Confirm exchange a set of polling intervals are established including T1, which establishes the interval between Hello command retransmissions, and T2, which establishes the interval between Poll retransmissions.

Each gateway configuration is characterized by a set of fixed parameters, including P1, which specifies the minimum polling interval

at which it will respond to Hello commands, and P2, which specifies the minimum polling interval at which it will respond to Poll commands. P1 and P2 are inserted in the Hello Interval (S1) and Poll Interval (S2) fields, respectively, of Request commands and Confirm responses.

A gateway receiving a Request command or Confirm response uses the S1 and S2 fields in the message to calculate its own T1 and T2 state variables, respectively. Implementors are free to perform this calculation in arbitrary ways; however, the following constraints must be observed:

1. If T1 < S1 the neighbor may discard Hello commands. If T2 < S2 the neighbor may discard Poll commands.

2. The time window T3 in which neighbor-reachability indications are counted is dependent on T1. In the case where two neighbors select widely differing values for their T3 state variables, the neighbor-reachability algorithm may not work properly. This can be avoided if T1 > max(P1, S1).

3. If either S1 or S2 or both are unacceptable for some reason (e.g. exceed useful limits), the neighbor may either send a Refuse response or declare a Stop event, depending on state.

It is suggested that T3 be computed as four times the value of T1, giving a window of four neighbor-reachability indications, which has been found appropriate in the experimental implementations. Implementors may choose to make T3 a fixed parameter in those cases where the path between the neighbors has well-known characteristics.

Note that, if a gateway attempts to send Hello commands near the rate max(P1, S1) or Poll commands near the rate max(P2, S2), the neighbor may observe their succeeding arrivals to violate the polling restrictions due to bunching in the net. For this reason the gateway should send at rates somewhat below these. Just how much below these rates is appropriate depends on many factors beyond the scope of this specification.

4.1.3. Hello Polling Mode

The neighbor-reachability algorithm can be used in either the active or passive mode. In the active mode Hello commands are sent periodically along with Poll commands, with reachability determined by the corresponding I-H-U and Update responses. In the passive mode Hello commands are not sent and I-H-U responses are not expected. Reachability is then determined from the Status field of received Hello or Poll commands or Update responses.

The M state variable specifies whether the gateway operates in the active or passive mode. At least one of the two neighbors sharing the

protocol must operate in the active mode; however, the neighbor-reachability protocol is designed to work even if both neighbors operate in the active mode. The value of M is determined from the Status field of a Request command or Confirm response. The sender sets this field according to whether the implementation supports the active mode, passive mode or both:

Status	Sender capabilities
0	either active or passive
1	active only
2	passive only

The receiver inspects this field and sets the value of M according to its own capabilities as follows:

Status field	Receiver capabilites 0	1	2
0	*	active	passive
1	passive	active	passive
2	active	active	**

In the case of "*" the mode is determined by comparing the autonomous system numbers of the neigbors. The neighbor with the smallest such number assumes active mode, while the other neighbor assumes passive mode. In the case of "**" the neighbor may either send a Refuse response or declare a Stop event, depending on state.

4.1.4. Timers

There are three timers defined in the state machine: t1, used to control retransmission of Request, Hello and Cease messages, t2, used to control retransmission of Poll commands, and t3, which serves as an abort-timer mechanism should the protocol hang indefinately. The timers are set to specified values upon entry to each state and count down to zero.

In the case of t1 and t2 state-dependent events are declared when the timer counts down to zero, after which the timer is reset to the specified value and counts down again. In the case of t3 a Stop event is declared when the timer counts down to zero. Implementors may choose not to implement t3 or, if so, may choose to implement it only in certain states, with the effect that Request, Hello and/or Cease commands may be retransmitted indefinately.

The following table shows the initial values for each of the timers in each state. A missing value indicates the timer is not used in that state. Note that timer t3 is set to P4 upon receipt of a neighbor-reachability indication when in either the Down or Up states.

Timer	Idle 0	Aqsn 1	Down 2	Up 3	Cease 4
t1		P3	T1		P3
t2				T2	
t3		P5	P5		P5

4.2. Starting and Stopping the Protocol

The Start and Stop events are intrinsic to the system environment of the gateway. They can be declared as the result of the gateway process being started and stopped by the operator, for example. A Start event has meaning only in some states; however, a Stop event has meaning in all states.

In all except the Idle state the abort timer t3 is presumed running. This timer is initialized at P5 seconds upon entry to any state and at P4 seconds upon receipt of a neighbor-reachability indication in the Down and Up states. If it expires a Stop event is declared. A Stop event can also be declared by an intrinsic system action such as a resource problem or operator command.

If the abort timer is not implemented a manually-initiated Stop event can be used to stop the protocol. If this is done in the Down or Up states, the machine will transition to the Cease state and emit a Cease command. If the neighbor does not respond to this command the machine will stay in the Cease state indefinately; however, a second Stop event can be used in this state to force a transition to the Idle state.

A Cease command received in any state will cause the gateway to immediately send the Cease-ack response and transition to the Idle state. This causes the protocol to be stopped and all system resources committed to the gateway process to be released. The interval between the time the gateway enters the Idle state as the result of receiving a Cease command and the time when it next sends a Request command to resume the protocol is not specified; however, it is recommended this interval be at least P5 seconds.

It may happen that the Cease-ack response is lost in the network, causing the neighbor to retransmit the Cease response indefinately, at least if it has not implemented the abort-timer option. In order to reduce the likelihood of this happening, it is suggested that a gateway in the Idle state be prepared to reply to a Cease command with a Cease-ack response whenever possible.

4.3. Determining Neighbor Reachability

The purpose of the neighbor-reachability algorithm is to confirm that the neighbor can safely be considered operational and capable of

providing reliable net-reachability information. An equally important purpose is to filter noisy reachability information before sending it on to the remainder of the Internet gateway system, thus avoiding unneccesary reachability changes.

As described above, a gateway operating in the active mode sends periodic Hello commands and listens for I-H-U responses in order to determine neighbor-reachability indications. A gateway operating in the passive mode determines reachability indications by means of the Status field in received Hello commands. Poll commands and Update responses can be used in lieu of Hello commands and I-H-U responses respectively, since they contain the same Status-field information.

The neighbor-reachability algorithm runs continuously while the gateway is in the Down and Up states and operates as follows. Define a moving window in time starting at the present and extending backwards for t seconds. Then count the number n of neighbor-reachability indications which have occured in that window. If n increases to j, then declare a Up event. If n decreases to k, then declare a Down event. The number n is set to zero upon entering the Down state from any state other than the Up state.

The window t in this algorithm is defined as T3 seconds, the value of which is suggested as four times T1, which itself is determined during the Request/Confirm exchange. For proper operation of the algorithm only one neighbor-reachability indication is significant in any window of T1 seconds and additional ones are ignored. Note that the only way n can increase is as the result of a new neighbor-reachability indication and the only way it can decrease is as the result of an old neighbor-reachability indication moving out of the window.

The behavior of the algorithm described above and using the suggested fixed parameters j and k differs depending on whether the gateway is operating in the active or passive mode. In the active mode (j = 3, k = 1 and T3/T1 = 4), once the neighbor has been declared down it will be forced down for at least two T1 intervals and, once it has been declared up it will be forced up for at least two T1 intervals. It will not change state unless at least three of the last four determinations of reachability have indicated that change.

In the passive mode (j = 1, k = 4 and T3/T1 = 4), the neighbor will be considered up from the first time the Status field of a Hello or Poll command or Update response indicates "Up state" until four successive T1 intervals have passed without such indication. This design, suggested by similar designs used in the ARPANET, has proven effective in the experimental implementations, but may need to be adjusted for other configurations.

It is convenient for the active gateway to send Hello commands at a rate of one every T1 seconds and substitute a Poll command for a Hello

command approximately once every T2 seconds, with the neighbor-reachability indication generated by the corresponding I-H-U or Update responses. Its passive neighbor generates neighbor-reachability indications from the Status field of received Hello and Poll commands and Update responses.

Implementors may find the following model useful in the understanding and implementation of this algorithm. Consider an n-bit shift register that shifts one bit to the right each T1-second interval. If a neighbor-reachability indication was received during the preceeding T1-second interval a one bit is shifted into the register at the end of the interval; otherwise, a zero bit is shifted. A table of 2**n entries indexed by the contents of the register can be used to calculate the number of one bits, which can then be used to declare the appropriate event to the state machine. A value of n equal to four has been found useful in the experimental implementations.

4.4. Determining Network Reachability

Network reachability information is encoded into Update messages in the form of lists of nets and gateways. The IP Source Address field of the Poll command is used to specify a network common to the autonomous systems of each of the neighbors, which is usually, but not necessarily, the one common to the neighbors themselves. The Update response includes a list of gateways on the common net. Associated with each gateway is a list of the networks reachable via that gateway together with corresponding hop counts.

It is important to understand that, at the present state of development as described in RFC-827 and RFC-888, the EGP architectural model restricts the interpretation of "reachable" in this context. This consideration, as well as the implied topological restrictions, are beyond the scope of discussion here. The reader is referred to the RFCs for further discussion.

Two types of gateway lists can be included in the Update response, the format of which is described in Appendix A. Both lists include only those gateways directly connected to the net specified in the IP Source Network field of the last-received Poll command. The internal list includes some or all of the gateways in the same autonomous system as the sender, together with the nets which are reachable via these gateways, with the sending gateway listed first. A net is reachable in this context if a path exists to that net including only gateways in the system. The external list includes those gateways in other autonomous systems known to the sender. It is important to realize that the hop counts do not represent a routing metric and are comparable between different gateways only if those gateways belong to the same autonomous system; that is, are in the internal list.

According to the current system architectural model, only gateways belonging to a designated system, called the core system, may include the external list in their Update responses. All other gateways may include only those gateways belonging to the same system and can claim reachability for a particular net only if that net is reachable in the same system.

The interval between successive Poll commands T2 is determined during the Request/Confirm exchange. However, the specification permits at most one unsolicited Update indication between succeeding Poll commands received from the neighbor. It is the intent of the model here that an Update indication is sent (a) upon entry to the Up state and (b) when a change in the reachability data base is detected, subject to this limitation.

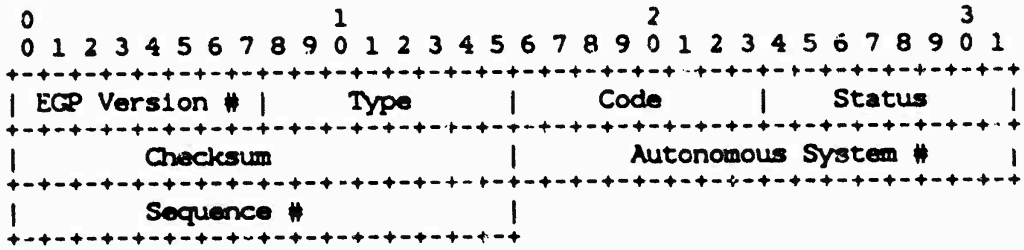
Occasionally it may happen that a Poll command or Update response is lost in the network, with the effect that net-reachability information may not be available until after another T2 interval. As an implementation option, the gateway sending a Poll command and not receiving an Update response after T1 seconds may send another Poll. The gateway receiving this Poll may either (a) send an Update response if it never received the original Poll for that interval, (b) send a second Update response (which counts as the unsolicited Update indication mentioned in the preceeding paragraph) or (c) send an Error response or not respond at all in other cases.

4.5. Error Messages

Error messages can be used to report problems such as described in Appendix A in connection with the Error Response/Indication message format. In general, an Error message is sent upon receipt of another command or response with bad format, content or ordering, but never in response to another Error message. Receipt of an Error message should be considered advisory and not result in change of state, except possibly to evoke a Stop event.

Appendix A. EGP Message Formats

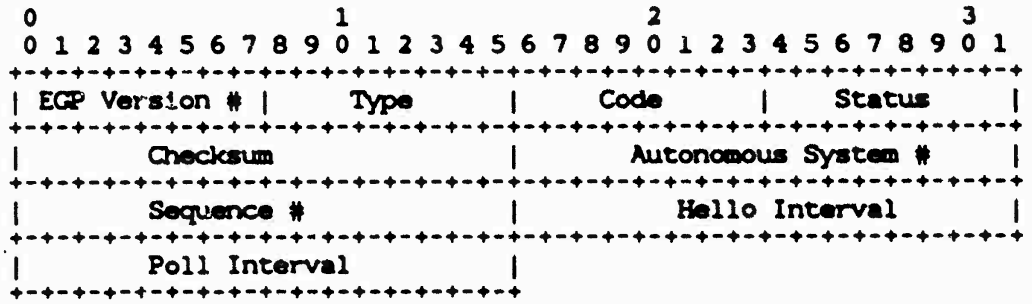
The formats for the various EGP messages are described in this section. All EGP messages include a ten-octet header of six fields, which may be followed by additional fields depending on message type. The format of the header is shown below along with a description of its fields.



EGP Version #	assigned number identifying the EGP version (currently 2)
Type	identifies the message type
Code	identifies the message code (subtype)
Status	contains message-dependent status information
Checksum	The EGP checksum is the 16-bit one's complement of the one's complement sum of the EGP message starting with the EGP version number field. When computing the checksum the checksum field itself should be zero.
Autonomous System #	assigned number identifying the particular autonomous system
Sequence #	send state variable (commands) or receive state variable (responses and indications)

Following is a description of each of the message formats. Note that the above description applies to all formats and will not be repeated.

A.1. Neighbor Acquisition Messages



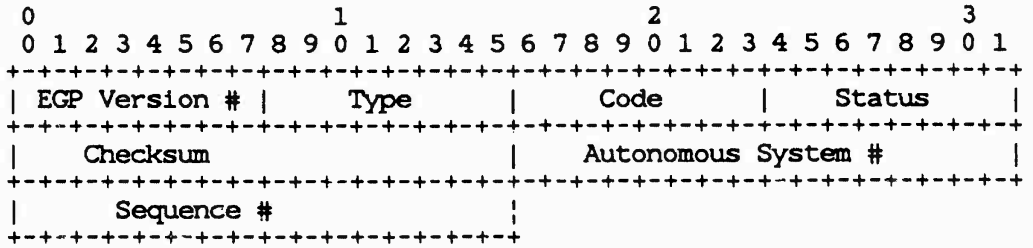
Note: the Hello Interval and Poll Interval fields are present only in Request and Confirm messages.

Type	3	
Code	0	Request command
	1	Confirm response
	2	Refuse response
	3	Cease command
	4	Cease-ack response
Status (see below)	0	unspecified
	1	active mode
	2	passive mode
	3	insufficient resources
	4	administratively prohibited
	5	going down
	6	parameter problem
	7	protocol violation
Hello Interval	minimum Hello command polling interval (seconds)	
Poll Interval	minumum Poll command polling interval (seconds)	

Following is a summary of the assigned Status codes along with a list of scenarios in which they might be used.

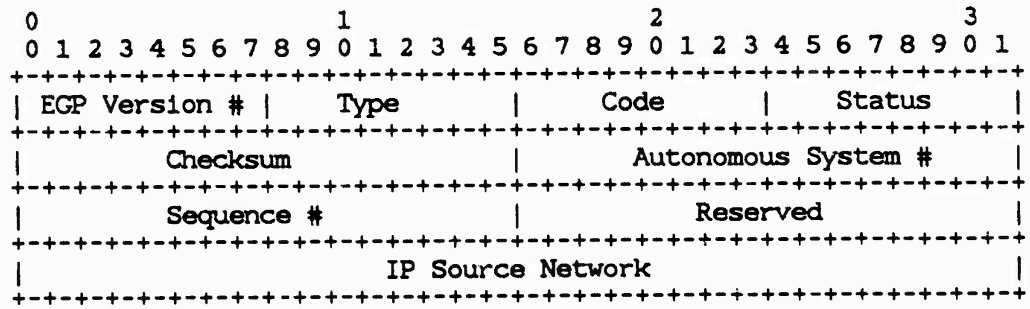
Code	Status	Scenarios
0	unspecified	when nothing else fits
1	active mode	Request/Confirm only
2	passive mode	Request/Confirm only
3	insufficient resources	1. out of table space 2. out of system resources
4	administratively prohibited	1. unknown Autonomous System 2. use another gateway
5	going down	1. operator initiated Stop 2. abort timeout
6	parameter problem	1. nonsense polling parameters 2. unable to assume compatible mode
7	protocol violation	1. Invalid command or response received in this state

A.2. Neighbor Reachability Messages



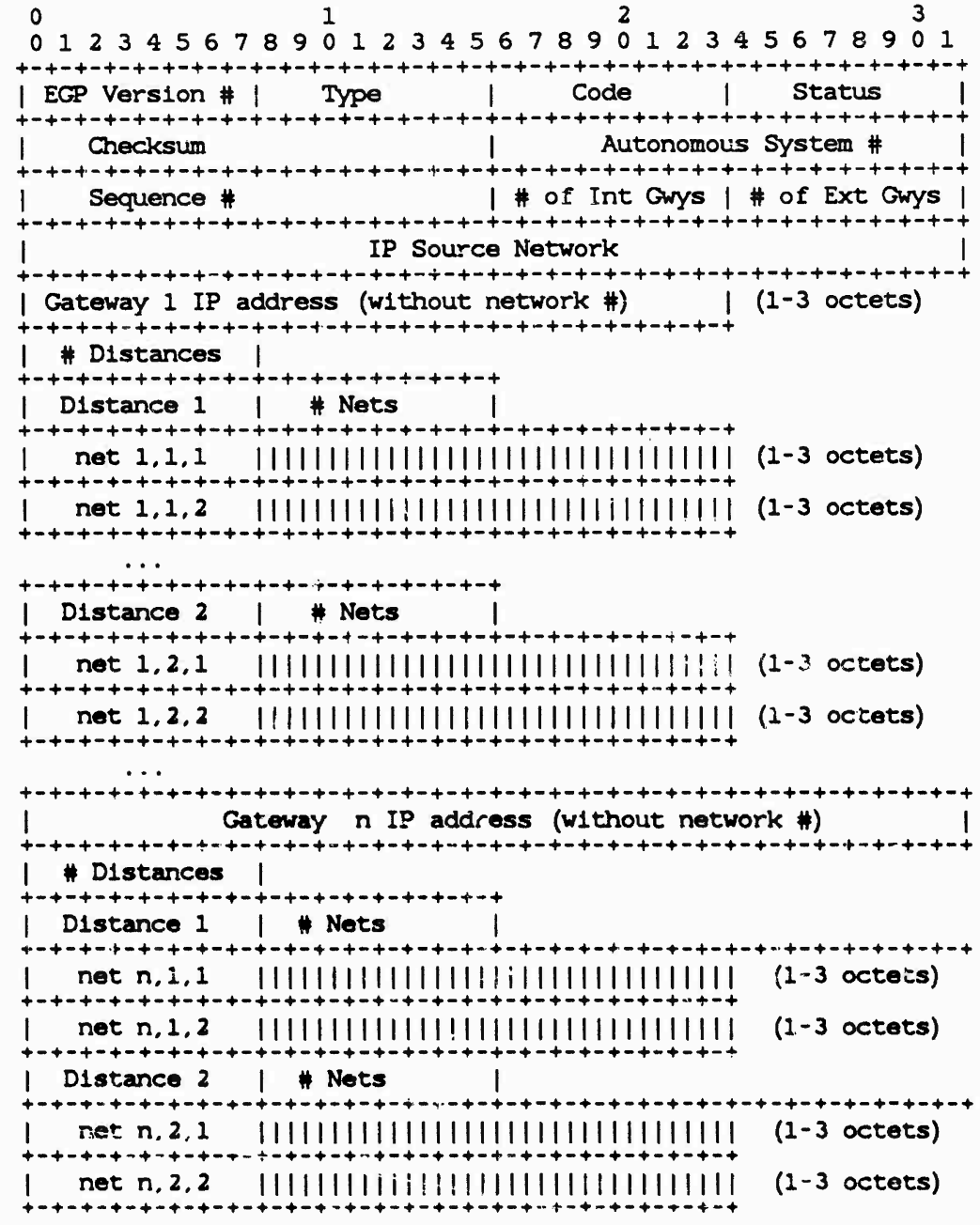
Type	5	
Code	0	Hello command
	1	I-H-U response
Status	0	indeterminate
	1	Up state
	2	Down state

A.3. Poll Command



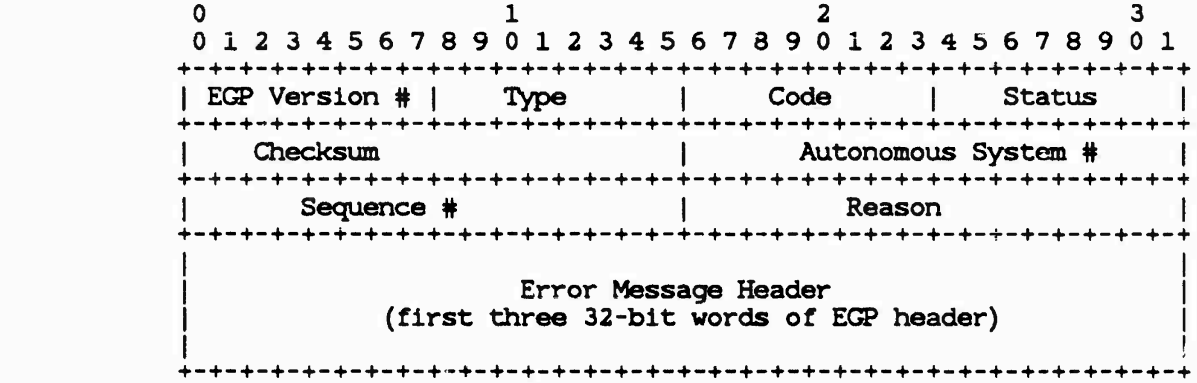
Type	2	
Code	0	
Status	0	indeterminate
	1	Up state
	2	Down state
IP Source Network	IP network number of the network about which reachability information is being requested (coded as 1, 2 or 3 octets, left justified with trailing zeros)	

A.4. Update Response/Indication



Type	1
Code	0
Status	0 indeterminate 1 Up state 2 Down state 128 unsolicited message bit
# of Int Gwys	number of interior gateways appearing in this message
# of Ext Gwys	number of exterior gateways appearing in this message
IP Source Network	IP network number of the network about which reachability information is being supplied (coded as 1, 2 or 3 octets, left justified with trailing zeros)
Gateway IP addresses	IP address (without network number) of the gateway block (coded as 1, 2 or 3 octets)
# of Distances	number of distances in the gateway block
Distances	numbers depending on autonomous system architecture
# of Nets	number of nets at each distance
Nets	IP network number reachable via the gateway

A.5. Error Response/Indication



Type	8	
Code	0	
Status	0	indeterminate
	1	Up state
	2	Down state
	128	unsolicited message bit
Reason (see below)	0	unspecified
	1	bad EGP header format
	2	bad EGP data field format
	3	reachability info unavailable
	4	excessive polling rate
	5	no response
Error Message Header	first three 32-bit words of EGP header	

Following is a summary of the assigned Reason codes along with a list of scenarios in which they might be used.

Code	Reason	Scenarios
0	unspecified	when nothing else fits
1	bad EGP header format	1. bad message length 2. invalid Type, Code or Status fields Notes: The recipient can determine which of the above hold by inspecting the EGP header included in the message. An instance of a wrong EGP version or bad checksum should not be reported, since the original recipient can not trust the header format. An instance of an unknown autonomous system should be caught at acquistion time.
2	bad EGP data field format	1. nonsense polling rates (Request/Confirm) 2. invalid Update message format 3. response IP Net Address field does not match command (Update) Notes: An instance of nonsense polling intervals (e.g. too long to be useful) specified in a Request or Confirm should result in a Refuse or Cease with this cause specified.
3	reachability info unavailable	1. no info available on net specified in IP Net Address field (Poll)
4	excessive polling rate	1. two or more Hello commands received within minimum specified polling interval 2. two or more Poll commands received within minimum specified polling interval 3. two or more Request commands received within some (reasonably short) interval Notes: The recipient can determine which of the above hold by inspecting the EGP header included in the message.
5	no response	1. no Update received for Poll within some (reasonably long) interval

Appendix B. Comparison with RFC-888

Minor functional enhancements are necessary in the RFC-888 message formats to support certain features assumed of the state-machine model, in particular the capability to request a neighbor to suppress Hello commands. In addition, the model suggests a mapping between its states and certain status and error indications which clarifies and generalizes the interpretation.

All of the header fields except the Status field (called the Information field at some places in RFC-888) remain unchanged. The following table summarizes the suggested format changes in the Status field for the various messages by (Type, Code) class.

Class	Messages	Status Codes	
3,0	Request	0	unspecified
3,1	Confirm	1	active mode
3,2	Refuse	2	passive mode
3,3	Cease	3	insufficient resources
3,4	Cease-ack	4	administratively prohibited
		5	going down
		6	parameter problem
5,0	Hello	0	indeterminate
5,1	I-H-U	1	Up state
2,0	Poll	2	Down state
1,0	Update	128	unsolicited message bit
8,0	Error		

The changes from RFC-888 are as follows:

1. The status codes have been combined in two classes, one for those messages involved in starting and stopping the protocol and the other for those messages involved in maintaining the protocol and exchanging reachability information. Some messages of either class may not use all the status codes assigned.

2. The status codes for the Request and Confirm indicate whether the sender can operate in active or passive mode. In RFC-888 this field must be zero; however, RFC-888 does not specify any mechanism to decide how the neighbors poll each other.

3. The status codes for the Cease, Refuse and Cease-ack have the same interpretation. This provides a clear and unambiguous indication when the protocol is terminated due to an unusual situation, for instance if the NOC dynamically repartitions the ARPANET. The assigned codes are not consistent with RFC-888, since the codes for the Refuse and Cease were assigned conflicting values; however, the differences are minor and should cause no significant problems.

4. The status codes for the Hello, I-H-U, Poll, Update and Error have the same interpretation. Codes 0 through 2 are mutually exclusive and are chosen solely on the basis of the state of the sender. In the case of the Update (and possibly Error) one of these codes can be combined with the "unsolicited bit," which corresponds to code 128. In RFC-888 this field is unused for the Poll and Error and may contain only zero or 128 for the Update, so that the default case is to assume that reciprocal reachability cannot be determined by these messages.

5. Some of the reachability codes defined in RFC-888 have been removed as not applicable.

Appendix C. Reachability Analysis

The following table shows the state transitions which can occur in a system of two neighboring EGP gateways. Besides being useful in the design and verification of the protocol, the table is useful for implementation and testing.

The system of two neighboring EGP gateways is modelled as a finite-state automaton constructed as the cartesian product of two state machines as defined above. Each state of this machine is represented as [i,j], where i and j are states of the original machine. Each line of the table shows one state transition of the machine in the form:

[i1,j1] -> [i2,j2] E A

which specifies the machine in state [i1,j1] presented with event E transitions to state [i2,j2] and generates action A. Multiple actions are separated by the "/" symbol. The special symbol "*" represents the set of lines where all "*"s in the line take on the (same) values 0 - 4 in turn.

The table shows only those transitions which can occur as the result of events arriving at one of the two neighbors. The full table includes a duplicate set of lines for the other neighbor as well, with each line derived from a line of the table below using the transformation:

[i1,j1] -> [i2,j2] E A => [j1,i1] -> [j2,i2] E A

State		State	Event	Actions
[*,4]	->	[0,4]	Cease	Cease-ack
[0,1]	->	[2,1]	Request	Confirm/Hello/Up/t1
[0,1]	->	[0,1]	Request	Refuse
[0,*]	->	[1,*]	Start	Request/t1
[1,1]	->	[2,1]	Request	Confirm/Hello/Up/t1
[1,2]	->	[2,2]	Confirm	Hello/Up/t1
[1,3]	->	[2,3]	Confirm	Hello/Up/t1
[1,0]	->	[0,0]	Refuse	Null
[1,*]	->	[1,*]	Start	Request/r1
[1,*]	->	[0,*]	Stop	Null
[1,*]	->	[1,*]	t1	Request/t1
[2,1]	->	[3,1]	Up	Down/Hello/Poll/t1/t2
[2,1]	->	[2,1]	Request	Confirm/Hello/Up/t1
[2,2]	->	[2,2]	Hello	I-H-U
[2,3]	->	[2,3]	Hello	I-H-U
[2,2]	->	[2,2]	I-H-U	Process

[2,3]	->	[2,3]	I-H-U	Process
[2,*]	->	[1,*]	Start	Request/r1
[2,*]	->	[4,*]	Stop	Cease/t1
[2,1]	->	[2,1]	t1	Hello/t1
[2,2]	->	[2,2]	t1	Hello/t1
[2,3]	->	[2,3]	t1	Hello/t1
[3,1]	->	[2,1]	Down	Null
[3,2]	->	[2,2]	Down	Null
[3,3]	->	[2,3]	Down	Null
[3,1]	->	[2,1]	Request	Confirm/Hello/Up/t1
[3,2]	->	[3,2]	Hello	I-H-U
[3,3]	->	[3,3]	Hello	I-H-U
[3,2]	->	[3,2]	I-H-U	Process
[3,3]	->	[3,3]	I-H-U	Process
[3,3]	->	[3,3]	Poll	Update
[3,3]	->	[3,3]	Update	Process
[3,*]	->	[1,*]	Start	Request/r1
[3,*]	->	[4,*]	Stop	Cease/t1
[3,1]	->	[3,1]	t1	Hello/t1
[3,2]	->	[3,2]	t1	Hello/t1
[3,3]	->	[3,3]	t1	Hello/t1
[3,1]	->	[3,1]	t2	Poll/t2
[3,2]	->	[3,2]	t2	Poll/t2
[3,3]	->	[3,3]	t2	Poll/t2
[4,1]	->	[4,1]	Request	Cease
[4,*]	->	[0,*]	Cease	Cease-ack
[4,0]	->	[0,0]	Cease-ack	Null
[4,*]	->	[0,*]	Stop	Null
[4,*]	->	[4,*]	t1	Cease/t1

In the state-machine model defined in this document all states of the above machine are reachable; however, some are reachable only in extreme cases when one neighbor crashes, for example. In the common case where only one of the neighbors initiates and terminates the protocol and neither one crashes, for example, not all states are reachable. Following is a matrix showing the states which can be reached in this case, where the neighbor that initiates and terminates the protocol is called the active gateway and the other the passive gateway.

Active Gateway	Passive Gateway: 0 Idle	1 Aqsn	2 Down	3 Up	4 Cease
0 Idle	stable				unstable
1 Aqsn	unstable	unstable	unstable	unstable	unstable
2 Down			stable	unstable	
3 Up			unstable	stable	
4 Cease	unstable	unstable	unstable	unstable	unstable

In the above matrix the blank entries represent unreachable states, while those marked unstable represent transient states which cannot persist for long, due to retransmission of Request and Hello messages, for example.

Request for Comments: 823
Obsoletes IEN-30 and IEN-109

THE DARPA INTERNET GATEWAY

RFC 823

Robert Hinden
Alan Sheltzer

Bolt Beranek and Newman Inc.
10 Moulton St.
Cambridge, Massachusetts 02238

September 1982

Prepared for

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

This RFC is a status report on the Internet Gateway developed by BBN. It describes the Internet Gateway as of September 1982. This memo presents detailed descriptions of message formats and gateway procedures, however this is not an implementation specification, and such details are subject to change.

DARPA Internet Gateway
RFC 823

September 1982

Table of Contents

DARPA Internet Gateway
RFC 823

September 1982

1 INTRODUCTION

This document explains the design of the Internet gateway used in the Defense Advanced Research Project Agency (DARPA) Internet program. The gateway design was originally documented in IEN-30, "Gateway Routing: An Implementation Specification" [2], and was later updated in IEN-109, "How to Build a Gateway" [3]. This document reflects changes made both in the internet protocols and in the gateway design since these documents were released. It supersedes both IEN-30 and IEN-109.

The Internet gateway described in this document is based on the work of many people; in particular, special credit is given to V. Strazisar, M. Brescia, E. Rosen, and J. Haverty.

The gateway's primary purpose is to route internet datagrams to their destination networks. These datagrams are generated and processed as described in RFC 791, "Internet Protocol - DARPA Internet Program Protocol Specification" [1]. This document describes how the gateway forwards datagrams, the routing algorithm and protocol used to route them, and the software structure of the current gateway. The current gateway implementation is written in macro-11 assembly language and runs in the DEC PDP-11 or LSI-11 16-bit processor.

2 BACKGROUND

The gateway system has undergone a series of changes since its inception, and it is continuing to evolve as research proceeds in the Internet community. This document describes the implementation as of mid-1982.

Early versions of gateway software were implemented using the BCPL language and the ELF operating system. This implementation evolved into one which used the MOS operating system for increased performance. In late 1981, we began an effort to produce a totally new gateway implementation. The primary motivation for this was the need for a system oriented towards the requirements of an operational communications facility, rather than the research testbed environment which was associated with the BCPL implementation. In addition, it was generally recognized that the complexity and buffering requirements of future gateway configurations were beyond the capabilities of the PDP-11/LSI-11 and BCPL architecture. The new gateway implementation therefore had a second goal of producing a highly space-efficient implementation in order to provide space for buffers and for the extra mechanisms, such as monitoring, which are needed for an operational environment.

This document describes the implementation of this new gateway which incorporates several mechanisms for operations activities, is coded in assembly language for maximum space-efficiency, but otherwise is fundamentally the same architecture as the older, research-oriented, implementations.

One of the results of recent research is the thesis that gateways should be viewed as elements of a gateway system, where the gateways act as a loosely-coupled packet-switching communications system. For reasons of maintainability and operability, it is easiest to build such a system in an homogeneous fashion where all gateways are under a single authority and control, as is the practice in other network implementations.

In order to create a system architecture that permitted multiple sets of gateways with each set under single control but acting together to implement a composite single Internet System, new protocols needed to be developed. These protocols, such as the "Exterior Gateway Protocol," will be introduced in the later releases of the gateway implementation.

We also anticipate further changes to the gateway architecture and implementation to introduce support for new

capabilities, such as large numbers of networks, access control, and other requirements which have been proposed by the Internet research community. This document represents a snapshot of the current implementation, rather than a specification.

3 FORWARDING INTERNET DATAGRAMS

This section describes how the gateway forwards datagrams between networks. A host computer that wants an IP datagram to reach a host on another network must send the datagram to a gateway to be forwarded. Before it is sent into the network, the host attaches to the datagram a local network header containing the address of the gateway

3.1 Input

When a gateway receives a message, the gateway checks the message's local network header for possible errors and performs any actions required by the host-to-network protocol. This processing involves functions such as verifying the local network header checksum or generating a local network acknowledgment message. If the header indicates that the message contains an Internet datagram, the datagram is passed to the Internet header check routine. All other messages received that do not pass these tests are discarded.

3.2 IP Header Checks

The Internet header check routine performs a number of validity tests on the IP header. Datagrams that fail these tests are discarded causing an HMP trap to be sent to the Internet Network Operations Center (INOC) [7]. The following checks are currently performed:

- o Proper IP Version Number
- o Valid IP Header Length (>= 20 bytes)
- o Valid IP Message Length
- o Valid IP Header Checksum
- o Non-Zero Time to Live field

After a datagram passes these checks, its Internet destination address is examined to determine if the datagram is addressed to the gateway. Each of the gateway's internet addresses (one for each network interface) is checked against the destination address in the datagram. If a match is not found, the datagram is passed to the forwarding routine.

If the datagram is addressed to the gateway itself, the IP options in the IP header are processed. Currently, the gateway supports the following IP options:

- o NOP
- o End of Option List
- o Loose Source and Record Route
- o Strict Source and Record Route

The datagram is next processed according to the protocol in the IP header. If the protocol is not supported by the gateway, it replies with an ICMP error message and discards the datagram. The gateway does not support IP reassembly, so fragmented datagrams which are addressed to the gateway are discarded.

3.3 Routing

The gateway must make a routing decision for all datagrams that are to be to forwarded. The routing algorithm provides two pieces of information for the gateway: 1) the network interface that should be used to send this datagram and 2) the destination address that should be put in the local network header of the datagram.

The gateway maintains a dynamic Routing Table which contains an entry for each reachable network. The entry consists of a network number and the address of the neighbor gateway on the shortest route to the network, or else an indication that the

gateway is directly connected to the network. A neighbor gateway is one which shares a common network with this gateway. The distance metric that is used to determine which neighbor is closest is defined as the "number of hops," where a gateway is considered to be zero hops from its directly connected networks, one hop from a network that is reachable via one other gateway, etc. The Gateway-to-Gateway Protocol (GGP) is used to update the Routing Table (see Section 4.4 describing the Gateway-to-Gateway Protocol).

The gateway tries to match the destination network address in the IP header of the datagram to be forwarded, with a network in its Routing Table. If no match is found, the gateway drops the datagram and sends an ICMP Destination Unreachable message to the IP source. If the gateway does find an entry for the network in its table, it will use the network address of the neighbor gateway entry as the local network destination address of the datagram. However, if the final destination network is one that the gateway is directly connected to, the destination address in the local network header is created from the destination address in the IP header of the datagram.

3.4 Redirects

If the routing procedure decides that an IP datagram is to be sent back out the same network interface that it was read in, then this gateway is not on the shortest path to the IP final destination. Nevertheless, the datagram will still be forwarded to the next address chosen by the routing procedure. If the datagram is not using the IP Source Route Option, and the IP source network of the datagram is the same as the network of the next gateway chosen by the routing procedure, an ICMP Redirect message will be sent to the IP source host indicating that another gateway should be used to send traffic to the final IP destination.

3.5 Fragmentation

The datagram is passed to the fragmentation routine after the routing decision has been made. If the next network through which the datagram must pass has a maximum message size that is smaller than the size of the datagram, the datagram must be fragmented. Fragmentation is performed according to the algorithm described in the Internet Protocol Specification [1]. Certain IP options must be copied into the IP header of all

fragments, and others appear only in the first fragment according to the IP specification. If a datagram must be fragmented, but the Don't fragment bit is set, the datagram is discarded and an ICMP error message is sent to the IP source of the datagram.

3.6 Header Rebuild

The datagram (or the fragments of the original datagram if fragmentation was needed) is next passed to a routine that rebuilds the Internet header. The Time to Live field is decremented by one and the IP checksum is recomputed.

The local network header is now built. Using the information obtained from its routing procedure, the gateway chooses the network interface it considers proper to send the datagram and to build the destination address in the local network header.

3.7 Output

The datagram is now enqueued on an output queue for delivery towards its destination. A limit is enforced on the size of the output queue for each network interface so that a slow network

does not unfairly use up all of the gateway's buffers. If a datagram cannot be enqueued due to the limit on the output queue length, it is dropped and an HMP trap is sent to the INOC. These traps, and others of a similar nature, are used by operational personnel to monitor the operations of the gateways.

4 PROTOCOLS SUPPORTED BY THE GATEWAY

A number of protocols are supported by the gateway to provide dynamic routing, monitoring, debugging, and error reporting. These protocols are described below.

4.1 Cross-Net Debugging Protocol

The Cross-Net Debugging Protocol (XNET) [8] is used to load the gateway and to examine and deposit data. The gateway supports the following XNET op-codes:

- o NOP
- o Debug
- o End Debug
- o Deposit
- o Examine
- o Create Process

4.2 Host Monitoring Protocol

The Host Monitoring Protocol (HMP) [6] is used to collect measurements and status information from the gateways. Exceptional conditions in the gateways are reported in HMP traps. The status of a gateway's interfaces, neighbors, and the networks which it can reach are reported in the HMP status message.

Two types of gateway statistics, the Host Traffic Matrix and the gateway throughput, are currently defined by the HMP. The Host Traffic Matrix records the number of datagrams that pass through the gateway with a given IP source, destination, and protocol number. The gateway throughput message collects a number of important counters that are kept by the gateway. The current gateway reports the following values:

- Datagrams dropped because destination net unreachable
- Datagrams dropped because destination host unreachable

- Per Interface:
  - Datagrams received with IP errors
  - Datagrams received for this gateway
  - Datagrams received to be forwarded
  - Datagrams looped
  - Bytes received
  - Datagrams sent, originating at this gateway
  - Datagrams sent to destination hosts
  - Datagrams dropped due to flow control limitations
  - Datagrams dropped due to full queue
  - Bytes sent

- Per Neighbor:
  - Routing updates sent to
  - Routing updates received from
  - Datagrams sent, originating here
  - Datagrams forwarded to
  - Datagrams dropped due to flow control limitations
  - Datagrams dropped due to full queue
  - Bytes sent

4.3 ICMP

The gateway will generate the following ICMP messages under appropriate circumstances as defined by the ICMP specification [4]:

- o Echo Reply
- o Destination Unreachable
- o Source Quench
- o Redirect
- o Time Exceeded
- o Parameter Problem
- o Information Reply

4.4 Gateway-to-Gateway Protocol

The gateway uses the Gateway-to-Gateway Protocol (GGP) to determine connectivity to networks and neighbor gateways; it is also used in the implementation of a dynamic, shortest-path routing algorithm. The current GGP message formats (for release 1003 of the gateway software) are presented in Appendix A.

4.4.1 Determining Connectivity to Networks

When a gateway starts running it assumes that all its neighbor gateways are "down," that it is disconnected from

networks to which it is attached, and that the distance reported in routing updates from each neighbor to each network is "infinity."

The gateway first determines the state of its connectivity to networks to which it is physically attached. The gateway's connection to a network is declared up if it can send and receive internet datagrams on its interface to that network. Note that the method that the gateway uses to determine its connectivity to a network is network-dependent. In some networks, the host-to-network protocol determines whether or not datagrams can be sent and received on the host interface. In these networks, the gateway simply checks-status information provided by the protocol in order to determine if it can communicate with the network. In other networks, where the host-to-network protocols are less sophisticated, it may be necessary for the gateway to send datagrams to itself to determine if it can communicate with the network. In these networks, the gateways periodically poll the network using GGP network interface status messages [Appendix A] to determine if the network interface is operational.

The gateway has two rules relevant to computing distances to networks: 1) if the gateway can send and receive traffic on its

network interface, its distance to the network is zero; 2) if it cannot send and receive traffic on the interface, its distance to the network is "infinity." Note that if a gateway's network interface is not working, it may still be able to send traffic to the network on an alternate route via one of its neighbor gateways.

4.4.2 Determining Connectivity to Neighbors

The gateway determines connectivity to neighbors using a "K out of N" algorithm. Every 15 seconds, the gateway sends GGP Echo messages [Appendix A] to each of its neighbors. The neighbors respond by sending GGP echo replies. If there is no reply to K out of N (current values are K=3 and N=4) echo messages sent to a neighbor, the neighbor is declared down. If a neighbor is down and J out of M (current values are J=2 and M=4) echo replies are received, the neighbor is declared to be up. The values of J,K,M,N and the time interval are operational parameters which can be adjusted as required.

4.4.3 Exchanging Routing Information

The gateway sends routing information in GGP Routing Update messages. The gateway receives and transmits routing information reliably using sequence-numbered messages and a retransmission and acknowledgment scheme as explained below. For each neighbor, the gateway remembers the Receive Sequence Number, R, that it received in the most recent routing update from that neighbor. This value is initialized with the sequence number in the first Routing Update received from a neighbor after that neighbor's status is set to "up." On receipt of a routing update from a neighbor, the gateway subtracts the Receive Sequence Number, R, from the sequence number in the routing update, S. If this value (S-R) is greater than or equal to zero, then the gateway accepts the routing update, sends an acknowledgment (see Appendix A) to the neighbor containing the sequence number S, and replaces the Receive Sequence Number, R, with S. If this value (S-R) is less than zero, the gateway rejects the routing update and sends a negative acknowledgment [Appendix A] to the neighbor with sequence number R.

The gateway has a Send Sequence Number, N, for sending routing updates to all of its neighbors. This sequence number

can be initialized to any value. The Send Sequence Number is incremented each time a new routing update is created. On receiving an acknowledgment for a routing update, the gateway subtracts the sequence number acknowledged, A, from the Send Sequence Number, N. If the value (N-A) is non-zero, then an old routing update is being acknowledged. The gateway continues to retransmit the most recent routing update to the neighbor that sent the acknowledgment. If (N-A) is zero, the routing update has been acknowledged. Note that only the most recent routing update must be acknowledged; if a second routing update is generated before the first routing update is acknowledged, only the second routing update must be acknowledged.

If a negative acknowledgment is received, the gateway subtracts the sequence number negatively acknowledged, A, from its Send Sequence Number, N. If this value (N-A) is less than zero, then the gateway replaces its Send Sequence Number, N, with the sequence number negatively acknowledged plus one, A+1, and retransmits the routing update to all of its neighbors. If (N-A) is greater than or equal to zero, then the gateway continues to retransmit the routing update using sequence number N. In order to maintain the correct sequence numbers at all gateways, routing updates must be retransmitted to all neighbors if the Send

Sequence Number chan s, even if the routing information does not change.

The gateway retransmits routing updates periodically until they are acknowledged and whenever its Send Sequence Number changes. The gateway sends routing updates only to neighbors that are in the "up" state.

4.4.4 Computing Routes

A routing update contains a list of networks that are reachable through this gateway, and the distance in "number of hops" to each network mentioned. The routing update only contains information about a network if the gateway believes that it is as close or closer to that network then the neighbor which is to receive the routing update. The network address may be an internet class A, B, or C address.

The information inside a routing update is processed as follows. The gateway contains an N x K distance matrix, where N is the number of networks and K is the number of neighbor gateways. An entry in this matrix, represented as dm(I,J), is the distance to network I from neighbor J as reported in the most

recent routing update from neighbor J. The gateway also contains a vector indicating the connectivity between itself and its neighbor gateways. The values in this vector are computed as discussed above (see Section 4.4.2, Determining Connectivity to Neighbors). The value of the Jth entry of this vector, which is the connectivity between the gateway and the Jth neighbor, is represented as d(J).

The gateway copies the routing update received from the Jth neighbor into the appropriate row of the distance matrix, then updates its routes as follows. The gateway calculates a minimum distance vector which contains the minimum distance to each network from the gateway. The Ith entry of this vector, represented as MinD(I) is:

MinD(I) = minimum over all neighbors of d(J) + dm(I,J)

where d(J) is the distance between the gateway and the Jth neighbor, and dm(I,J) is the distance from the Jth neighbor to the Ith network. If the Ith network is attached to the gateway and the gateway can send and receive traffic on its network interface (see Section 4.4.2), then the gateway sets the Ith entry of the minimum distance vector to zero.

Using the minimum distance vector, the gateway computes a list of neighbor gateways through which to send traffic to each network. The entry for a given network contains one of the neighbors that is the minimum distance away from that network.

After updating its routes to the networks, the gateway computes the new routing updates to be sent to its neighbors. The gateway reports a network to a neighbor only if it is as close to or closer to that network than its neighbor. For each network I, the routing update contains the address of the network and the minimum distance to that network which is MinD(I).

Finally, the gateway must determine whether it should send routing updates to its neighbors. The gateway sends new updates to its neighbors if every one of the following three conditions occurs: 1) one of the gateway's interfaces changes state, 2) one of the gateway's neighbor gateways changes state, and 3) the gateway receives a routing update from a neighbor that is different from the update that it had previously received from that neighbor. The gateway sends routing updates only to neighbors that are currently in the "up" state.

The gateway requests a routing update from neighbors that are in the "up" state, but from which it has yet received a

routing update. Routing updates are requested by setting the appropriate bit in the routing update being sent [Appendix A]. Similarly, if a gateway receives from a neighbor a routing update in which the bit requesting a routing update is set, the gateway sends the neighbor the most recent routing update.

4.4.5 Non-Routing Gateways

A Non-routing Gateway is a gateway that forwards internet traffic, but does not implement the GGP routing algorithm. Networks that are behind a Non-routing Gateway are known a-priori to Routing Gateways. There can be one or more of these networks which are considered to be directly connected to the Non-routing Gateway. A Routing Gateway will forward a datagram to a Non-routing Gateway if it is addressed to a network behind the Non-routing Gateway. Routing Gateways currently do not send Redirects for Non-routing Gateways. A Routing Gateway will always use another Routing Gateway as a path instead of a Non-routing Gateways if both exist and are the same number of hops away from the destination network. The Non-routing Gateways path will be used only when the Routing Gateway path is down; when the Routing Gateway path comes back up, it will be used again.

4.4.6 Adding New Neighbors and Networks

Gateways dynamically add routing information about new neighbors and new networks to their tables. The gateway maintains a list of neighbor gateway addresses. When a routing update is received, the gateway searches this list of addresses for the Internet source address of the routing update message. If the Internet source address of the routing update is not contained in the list of neighbor addresses, the gateway adds this address to the list of neighbor addresses and sets the neighbor's connectivity status to "down." Routing updates are not accepted from neighbors until the GGP polling mechanism has determined that the neighbor is up.

This strategy of adding new neighbors requires that one gateway in each pair of neighbor gateways must have the neighbor's address configured in its tables. The newest gateway can be given a complete list of neighbors, thus avoiding the need to re-configure older gateways when new gateways are installed.

Gateways obtain routing information about new networks in several steps. The gateway has a list of all the networks for which it currently maintains routing information. When a routing update is received, if the routing update contains information

about a new network, the gateway adds this network to the list of networks for which it maintains routing information. Next, the gateway adds the new network to its distance matrix. The distance matrix comprises the is the matrix of distances (number of hops) to networks as reported in routing updates from the neighbor gateways. The gateway sets the distance to all new networks to "infinity," and then computes new routes and new routing updates as outlined above.

4.5 Exterior Gateway Protocol

The Exterior Gateway Protocol (EGP) is used to permit other gateways and gateway systems to pass routing information to the DARPA Internet gateways. The use of the EGP permits the user to perceive all of the networks and gateways as part of one total Internet system, even though the "exterior" gateways are disjoint and may use a routing algorithm that is different and not compatible with that used in the "interior" gateways. The important elements of the EGP are:

o Neighbor Acquisition

> The procedure by which a gateway requests that it become a neighbor of another gateway. This is used when a gateway wants to become a neighbor of another in order to pass

routing information. This includes the capability to accept or refuse the request.

o Neighbor Up/Down

The procedure by which a gateway decides if another gateway is up or down.

o Network Reachability Information

The facility used to pass routing and neighbor information between gateways.

o Gateway Going Down

The ability of a gateway to inform other gateways that it is going down and no longer has any routes to any other networks. This permits a gateway to go down in an orderly way without disrupting the rest of the Internet system.

A complete description of the EGP can be found in IEN-209, the "Exterior Gateway Protocol" [10].

5 GATEWAY SOFTWARE

The DARPA Internet Gateway runs under the MOS operating system [9] which provides facilities for:

- o Multiple processes
- o Interprocess communication
- o Buffer management
- o Asynchronous input/output
- o Shareable real-time clock

There is a MOS process for each network that the gateway is directly connected to. A data structure called a NETBLOCK contains variables of interest for each network and pointers to local network routines. Network processes run common gateway code while network-specific functions are dispatched to the routines pointed to in the NETBLOCK. There are also processes for gateway functions which require their own timing, such as GGP and HMP.

5.1 Software Structure

The gateway software can be divided conceptually into three parts: MOS Device Drivers, Network software, and Shared Gateway software.

5.1.1 Device Drivers

The gateway has a set of routines to handle sending and receiving data for each type of hardware interface. There are routines for initialization, initiation, and interruption for both the transmit and receive sides of a device. The gateway supports the following types of devices:

a) ACC LSI-11 1822
b) DEC IMP11a 1822
c) ACC LHDH 1822
d) ACC VDH11E
e) ACC VDH11C
f) Proteon Ring Network
g) RSRE HDLC
h) Interlan Ethernet
i) BBN Fibernet
j) ACC XQ/CP X.25 **
k) ACC XQ/CP HDH **

5.1.2 Network Software

For each connected network, the gateway has a set of eight routines which handle local network functions. The network routines and their functions are described briefly below.

** Planned, not yet supported.

Up.net Perform local network initialization such as flapping the 1822 ready line.

Sg.net Handle specific local network timing functions such as timing out 1822 Destination Deads.

Rc.net A message has been received from the network interface. Check for any input errors.

Wc.net A message has been transmitted to the network interface. Check for any output errors.

Rs.net Set up a buffer (or buffers) to receive messages on the network interface.

Ws.net Transmit a message to the network interface.

Hc.net Check the local network header of the received message. Perform any local network protocol tasks.

Hb.net Rebuild the local network header.

There are network routines for the following types of networks:

- Arpanet (a,b,c,k)
- Satnet (d,e,k)
- Proteon Ring Network (f)
- Packet Radio Network (a,b,c)
- Rsre HDLC Null Network (g)
- Ethernet (h)
- Fibernet (i)
- Telenet X.25 (j) **

Note: The letters in parentheses refer to the device drivers used

** Planned, not yet supported.

for each type of network as described in the previous section.

5.1.3 Shared Gateway Software

The internet processing of a datagram is performed by a body of code which is shared by the network processes. This code includes routines to check the IP header, perform IP fragmentation, calculate the IP checksum, forward a datagram, and implement the routing, monitoring, and error reporting protocols.

5.2 Gateway Processes

5.2.1 Network Processes

When the gateway starts up, each network process calls its local network initialization routine and read start routine. The read start routine sets up two maximum network size buffers for receiving datagrams. The network process then waits for an input complete signal from the network device driver.

When a message has been received, the MOS Operating System signals the appropriate network process with an input complete signal. The network process wakes up and executes the net read

complete routine. After the message has been processed, the network process waits for more input.

The net read complete routine is the major message processing loop in the gateway. The following actions are performed when a message has been received:

- o Call Local Network Read Complete Routine
- o Start more reads
- o Check local Network Header
- o Check Internet header
- o Check if datagram is for the gateway
- o Forward the datagram if necessary
- o Send ICMP error message if necessary.

5.2.2 GGP Process

The GGP process periodically sends GGP echos to each of the gateway's neighbors to determine neighbor connectivity, and sends interface status messages addressed to itself to determine network connectivity. The GGP process also sends out routing updates when necessary. The details of the algorithms currently implemented by the GGP process are given in Section 4.4, Gateway-to-Gateway Protocol, and in Appendix C.

5.2.3 HMP Process

The HMP process handles timer-based gateway statistics collection and the periodic transmission of traps.

APPENDIX A. GGP Message Formats

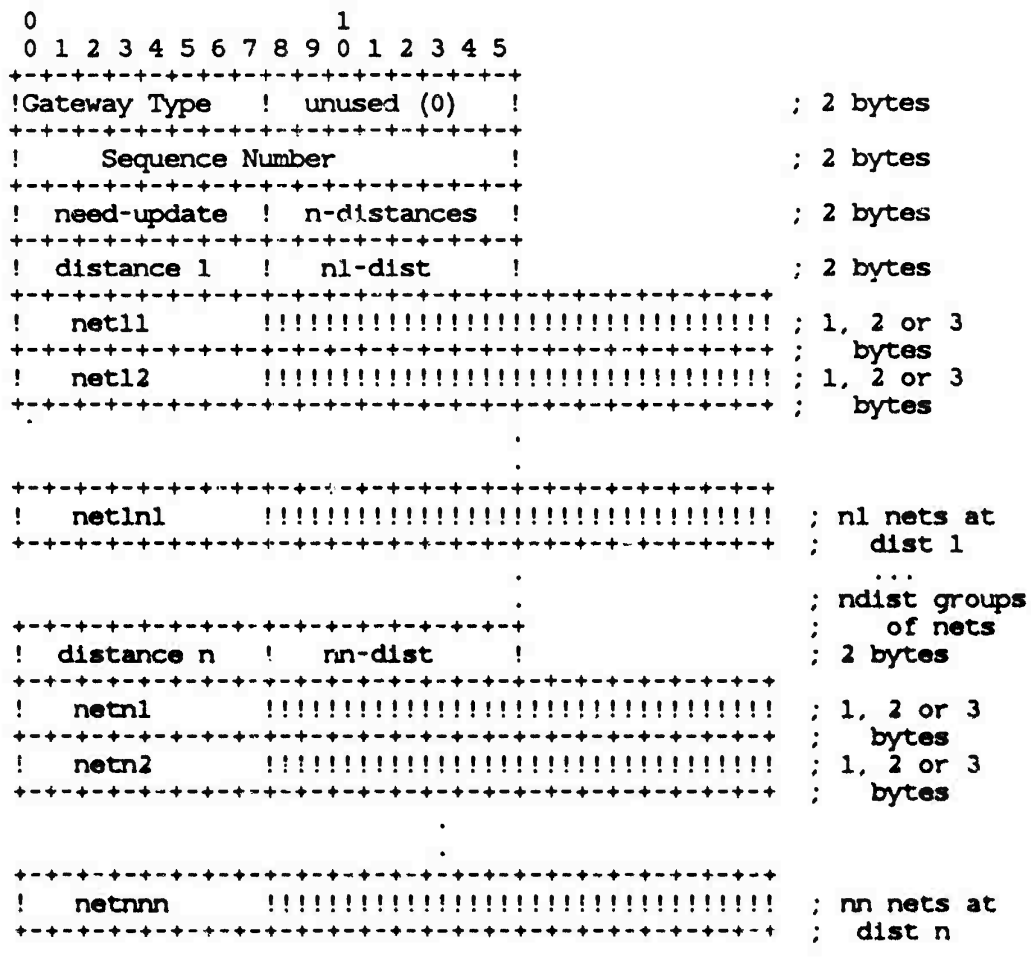
Note that the GGP protocol is currently undergoing extensive changes to introduce the Exterior Gateway Protocol facility; this is the vehicle needed to permit gateways in other systems to exchange routing information with the gateways described in this document.

Each GGP message consists of an Internet header followed by one of the messages explained below. The values (in decimal) in the Internet header used in a GGP message are as follows.

Version	4.
IHL	Internet header length in 32-bit words.
Type of Service	0.
Total Length	Length of Internet header and data in octets.
ID, Flags, Fragment Offset	0.
Time to Live	Time to live in seconds. This field is decremented at least once by each machine that processes the datagram.
Protocol	Gateway Protocol = 3.
Header Checksum	The 16 bit one's complement of the one's complement sum of all 16-bit words in the header. For computing the checksum, the checksum field should be zero.

Source Address	The address of the gateway's interface from which the message is sent.
Destination Address	The address of the gateway to which the message is sent.

ROUTING UPDATE



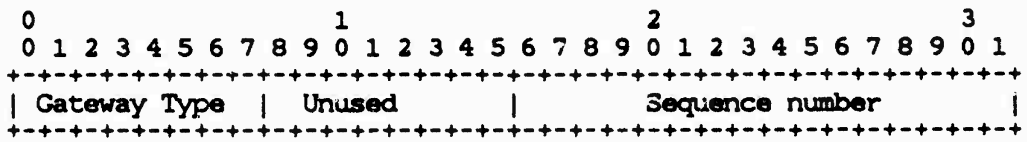
Gateway Type — 12 (decimal)

Sequence Number — The 16-bit sequence number used to identify routing updates.

need-update — An 8-bit field. This byte is set to 1

	if the source gateway requests a routing update from the destination gateway, and set to 0 if not.
n-distances	An 8-bit field. The number of distance-groups reported in this update. Each distance-group consists of a distance value and a number of nets, followed by the actual net numbers which are reachable at that distance. Not all distances need be reported.
distance 1	hop count (or other distance measure) which applies to this distance-group.
n1-dist	number of nets which are reported in this distance-group.
net11	1, 2, or 3 bytes for the first net at distance "distance 1".
net12	second net
...	
net1n1	etc.

ACKNOWLEDGMENT or NEGATIVE ACKNOWLEDGMENT



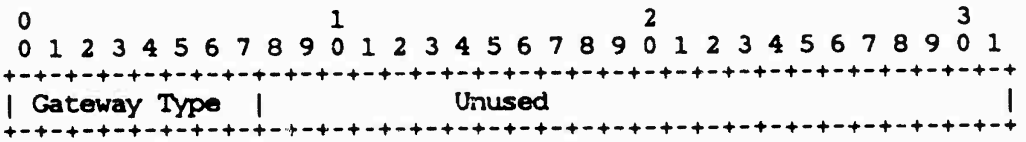
Gateway Type — Acknowledgments are type 2. Negative acknowledgments are type 10.

Sequence Number — The 16-bit sequence number that the gateway is acknowledging or negatively acknowledging.

DARPA Internet Gateway
RFC 823

September 1982

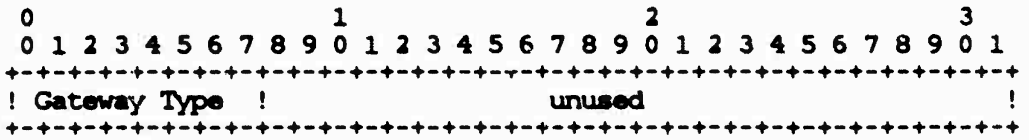
GGP ECHO and ECHO REPLY



Gateway Type — 8 for echo message; 0 for echo reply.

Source Address — In an echo message, this is the address of the gateway on the same network as the neighbor to which it is sending the echo message. In an echo reply message, the source and destination addresses are simply reversed, and the remainder is returned unchanged.

NETWORK INTERFACE STATUS



Gateway Type 9

Source Address
Destination Address The address of the gateway's network interface. The gateway can send Net Interface Status messages to itself to determine if it is able to send and receive traffic on its network interface.

APPENDIX B. Information Maintained by Gateways

In order to implement the shortest-path routing algorithm, gateways must maintain information about their connectivity to networks and other gateways. This section explains the information maintained by each gateway; this information can be organized into the following tables and variables.

- Number of Networks

  The number of networks for which the gateway maintains routing information and to which it can forward traffic.

- Number of Neighbors

  The number of neighbor gateways with which the gateway exchanges routing information.

- Gateway Addresses

  The addresses of the gateway's network interfaces.

- Neighbor Gateway Addresses

  The address of each neighbor gateway's network interface that is on the same network as this gateway.

- Neighbor Connectivity Vector

  A vector of the connectivity between this gateway and each of its neighbors.

- Distance Matrix

  A matrix of the routing updates received from the neighbor gateways.

- Minimum Distance Vector

  A vector containing the minimum distance to each network.

- Routing Updates from Non-Routing Gateways

  The routing updates that would have been received from each non-routing neighbor gateway which does not participate in this routing strategy.

- Routing Table

  A table containing, for each network, a list of the neighbor gateways on a minimum-distance route to the network.

- Send Sequence Number

  The sequence number that will be used to send the next routing update.

- Receive Sequence Numbers

  The sequence numbers that the gateway received in the last routing update from each of its neighbors.

- Received Acknowledgment Vector

  A vector indicating whether or not each neighbor has acknowledged the sequence number in the most recent routing update sent.

APPENDIX C. GGP Events and Responses

The following list shows the GGP events that occur at a gateway and the gateway's responses. The variables and tables referred to are listed above.

- Connectivity to an attached network changes.

  a. Update the Minimum Distance Vector.
  b. Recompute the Routing Updates.
  c. Recompute the Routing Table.
  d. If any routing update has changed, send the new routing updates to the neighbors.

- Connectivity to a neighbor gateway changes.

  a. Update the Neighbor Connectivity Vector.
  b. Recompute the Minimum Distance Vector.
  c. Recompute the Routing Updates.
  d. Recompute the Routing Table.
  e. If any routing update has changed, send the new routing updates to the neighbors.

- A Routing Update message is received.

  a. Compare the Internet source address of the Routing Update message to the Neighbor Addresses. If the address is not on the list, add it to the list of Neighbor Addresses, increment the Number of Neighbors, and set the Receive Sequence Number for this neighbor to the sequence number in the Routing Update message.

  b. Compare the Receive Sequence Number for this neighbor to the sequence number in the Routing Update message to determine whether or not to accept this message. If the message is rejected, send a Negative Acknowledgment message. If the message is accepted, send an Acknowledgment message and proceed with the following steps.

c. Compare the networks reported in the Routing Update message to the Number of Networks. If new networks are reported, enter them in the network vectors, increase the number of networks, and expand the Distance Matrix to account for the new networks.

d. Copy the routing update received into the appropriate row of the Distance Matrix.

e. Recompute the Minimum Distance Vector.

f. Recompute the Routing Updates.

g. Recompute the Routing Table.

h. If any routing update has changed, send the new routing updates to the neighbors.

o An Acknowledgment message is received.

Compare the sequence number in the message to the Send Sequence Number. If the Send Sequence Number is acknowledged, update the entry in the Received Acknowledgment Vector for the neighbor that sent the acknowledgment.

o A Negative Acknowledgment message is received.

Compare the sequence number in the message to the Send Sequence Number. If necessary, replace the Send Sequence Number, and retransmit the routing updates.

REFERENCES

[1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

[2] Strazisar, V., "Gateway Routing: An Implementation Specification," IEN-30, Bolt Beranek and Newman Inc., August 1979.

[3] Strazisar, V., "How to Build a Gateway," IEN-109, Bolt Beranek and Newman Inc., August 1979.

[4] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," RFC 792, USC/Information Sciences Institute, September 1981.

[5] Postel, J., "Assigned Numbers," RFC 790, USC/Information Sciences Institute, September 1981.

[6] Littauer, B., Huang, A., Hinden, R., "A Host Monitoring Protocol," IEN-197, Bolt Beranek and Newman Inc., September 1981.

[7] Santos, P., Chalstrom, H., Linn, J., Herman, J., "Architecture of a Network Monitoring, Control and Management System," Proc. of the 5th Int. Conference on Computer Communication, October 1980.

[8] Haverty, J., "XNET Formats for Internet Protocol Version 4," IEN-158, Bolt Beranek and Newman Inc., October 1980.

[9] Mathis, J., Klemba, K., Poggio, "TIU Notebook- Volume 2, Software Documentation," SRI, May 1979.

[10] Rosen, E., "Exterior Gateway Protocol," IEN-209, Bolt Beranek and Newman Inc., August 1982.

SECTION 8. APPLICATION LEVEL PROTOCOLS

This section includes RFCs pertaining to major applications (implemented by most hosts), minor applications (implemented by many hosts), and miscellaneous applications (implemented by few hosts).

Network Working Group
Request for Comments: 854

Obsoletes: NIC 18639

J. Postel
J. Reynolds
ISI

May 1983

TELNET PROTOCOL SPECIFICATION

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

INTRODUCTION

The purpose of the TELNET Protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation).

GENERAL CONSIDERATIONS

A TELNET connection is a Transmission Control Protocol (TCP) connection used to transmit data with interspersed TELNET control information.

The TELNET Protocol is built upon three main ideas: first, the concept of a "Network Virtual Terminal"; second, the principle of negotiated options; and third, a symmetric view of terminals and processes.

1. When a TELNET connection is first established, each end is assumed to originate and terminate at a "Network Virtual Terminal" or NVT. An NVT is an imaginary device which provides a standard network-wide, intermediate representation of a canonical terminal. This eliminates the need for "server" and "user" hosts to keep information about the characteristics of each other's terminals and terminal handling conventions. All hosts, both user and server, map their local device characteristics and conventions so as to appear to be dealing with an NVT over the network, and each can assume a similar mapping by the other party. The NVT is intended to strike a balance between being overly restricted (not providing hosts a rich enough vocabulary for mapping into their local character sets) and being overly inclusive (penalizing users with modest terminals).

NOTE: The "user" host is the host to which the physical terminal is normally attached, and the "server" host is the host which is normally providing some service. As an alternate point of view,

ERRATA

Insert this page following 2-575.

applicable even in terminal-to-terminal or process-to-process communications, the "user" host is the host which initiated the communication.

2. The principle of negotiated options takes cognizance of the fact that many hosts will wish to provide additional services over and above those available within an NVT, and many users will have sophisticated terminals and would like to have elegant, rather than minimal, services. Independent of, but structured within the TELNET Protocol are various "options" that will be sanctioned and may be used with the "DO, DON'T, WILL, WON'T" structure (discussed below) to allow a user and server to agree to use a more elaborate (or perhaps just different) set of conventions for their TELNET connection. Such options could include changing the character set, the echo mode, etc.

The basic strategy for setting up the use of options is to have either party (or both) initiate a request that some option take effect. The other party may then either accept or reject the request. If the request is accepted the option immediately takes effect; if it is rejected the associated aspect of the connection remains as specified for an NVT. Clearly, a party may always refuse a request to enable, and must never refuse a request to disable some option since all parties must be prepared to support the NVT.

The syntax of option negotiation has been set up so that if both parties request an option simultaneously, each will see the other's request as the positive acknowledgment of its own.

3. The symmetry of the negotiation syntax can potentially lead to nonterminating acknowledgment loops -- each party seeing the incoming commands not as acknowledgments but as new requests which must be acknowledged. To prevent such loops, the following rules prevail:

a. Parties may only request a change in option status; i.e., a party may not send out a "request" merely to announce what mode it is in.

b. If a party receives what appears to be a request to enter some mode it is already in, the request should not be acknowledged. This non-response is essential to prevent endless loops in the negotiation. It is required that a response be sent to requests for a change of mode -- even if the mode is not changed.

c. Whenever one party sends an option command to a second party, whether as a request or an acknowledgment, and use of the option will have any effect on the processing of the data being sent from the first party to the second, then the command must be inserted in the data stream at the point where it is desired that it take

effect. (It should be noted that some time will elapse between the transmission of a request and the receipt of an acknowledgment, which may be negative. Thus, a host may wish to buffer data, after requesting an option, until it learns whether the request is accepted or rejected, in order to hide the "uncertainty period" from the user.)

Option requests are likely to flurry back and forth when a TELNET connection is first established, as each party attempts to get the best possible service from the other party. Beyond that, however, options can be used to dynamically modify the characteristics of the connection to suit changing local conditions. For example, the NVT, as will be explained later, uses a transmission discipline well suited to the many "line at a time" applications such as BASIC, but poorly suited to the many "character at a time" applications such as NLS. A server might elect to devote the extra processor overhead required for a "character at a time" discipline when it was suitable for the local process and would negotiate an appropriate option. However, rather than then being permanently burdened with the extra processing overhead, it could switch (i.e., negotiate) back to NVT when the detailed control was no longer necessary.

It is possible for requests initiated by processes to stimulate a nonterminating request loop if the process responds to a rejection by merely re-requesting the option. To prevent such loops from occurring, rejected requests should not be repeated until something changes. Operationally, this can mean the process is running a different program, or the user has given another command, or whatever makes sense in the context of the given process and the given option. A good rule of thumb is that a re-request should only occur as a result of subsequent information from the other end of the connection or when demanded by local human intervention.

Option designers should not feel constrained by the somewhat limited syntax available for option negotiation. The intent of the simple syntax is to make it easy to have options -- since it is correspondingly easy to profess ignorance about them. If some particular option requires a richer negotiation structure than possible within "DO, DON'T, WILL, WON'T", the proper tack is to use "DO, DON'T, WILL, WON'T" to establish that both parties understand the option, and once this is accomplished a more exotic syntax can be used freely. For example, a party might send a request to alter (establish) line length. If it is accepted, then a different syntax can be used for actually negotiating the line length -- such a "sub-negotiation" might include fields for minimum allowable, maximum allowable and desired line lengths. The important concept is that

such expanded negotiations should never begin until some prior (standard) negotiation has established that both parties are capable of parsing the expanded syntax.

In summary, WILL XXX is sent, by either party, to indicate that party's desire (offer) to begin performing option XXX, DO XXX and DON'T XXX being its positive and negative acknowledgments; similarly, DO XXX is sent to indicate a desire (request) that the other party (i.e., the recipient of the DO) begin performing option XXX, WILL XXX and WON'T XXX being the positive and negative acknowledgments. Since the NVT is what is left when no options are enabled, the DON'T and WON'T responses are guaranteed to leave the connection in a state which both ends can handle. Thus, all hosts may implement their TELNET processes to be totally unaware of options that are not supported, simply returning a rejection to (i.e., refusing) any option request that cannot be understood.

As much as possible, the TELNET protocol has been made server-user symmetrical so that it easily and naturally covers the user-user (linking) and server-server (cooperating processes) cases. It is hoped, but not absolutely required, that options will further this intent. In any case, it is explicitly acknowledged that symmetry is an operating principle rather than an ironclad rule.

A companion document, "TELNET Option Specifications," should be consulted for information about the procedure for establishing new options.

THE NETWORK VIRTUAL TERMINAL

The Network Virtual Terminal (NVT) is a bi-directional character device. The NVT has a printer and a keyboard. The printer responds to incoming data and the keyboard produces outgoing data which is sent over the TELNET connection and, if "echoes" are desired, to the NVT's printer as well. "Echoes" will not be expected to traverse the network (although options exist to enable a "remote" echoing mode of operation, no host is required to implement this option). The code set is seven-bit USASCII in an eight-bit field, except as modified herein. Any code conversion and timing considerations are local problems and do not affect the NVT.

TRANSMISSION OF DATA

Although a TELNET connection through the network is intrinsically full duplex, the NVT is to be viewed as a half-duplex device operating in a line-buffered mode. That is, unless and until

options are negotiated to the contrary, the following default conditions pertain to the transmission of data over the TELNET connection:

1) Insofar as the availability of local buffer space permits, data should be accumulated in the host where it is generated until a complete line of data is ready for transmission, or until some locally-defined explicit signal to transmit occurs. This signal could be generated either by a process or by a human user.

The motivation for this rule is the high cost, to some hosts, of processing network input interrupts, coupled with the default NVT specification that "echoes" do not traverse the network. Thus, it is reasonable to buffer some amount of data at its source. Many systems take some processing action at the end of each input line (even line printers or card punches frequently tend to work this way), so the transmission should be triggered at the end of a line. On the other hand, a user or process may sometimes find it necessary or desirable to provide data which does not terminate at the end of a line; therefore implementers are cautioned to provide methods of locally signaling that all buffered data should be transmitted immediately.

2) When a process has completed sending data to an NVT printer and has no queued input from the NVT keyboard for further processing (i.e., when a process at one end of a TELNET connection cannot proceed without input from the other end), the process must transmit the TELNET Go Ahead (GA) command.

This rule is not intended to require that the TELNET GA command be sent from a terminal at the end of each line, since server hosts do not normally require a special signal (in addition to end-of-line or other locally-defined characters) in order to commence processing. Rather, the TELNET GA is designed to help a user's local host operate a physically half duplex terminal which has a "lockable" keyboard such as the IBM 2741. A description of this type of terminal may help to explain the proper use of the GA command.

The terminal-computer connection is always under control of either the user or the computer. Neither can unilaterally seize control from the other; rather the controlling end must relinquish its control explicitly. At the terminal end, the hardware is constructed so as to relinquish control each time that a "line" is terminated (i.e., when the "New Line" key is typed by the user). When this occurs, the attached (local)

computer processes the input data, decides if output should be generated, and if not returns control to the terminal. If output should be generated, control is retained by the computer until all output has been transmitted.

The difficulties of using this type of terminal through the network should be obvious. The "local" computer is no longer able to decide whether to retain control after seeing an end-of-line signal or not; this decision can only be made by the "remote" computer which is processing the data. Therefore, the TELNET GA command provides a mechanism whereby the "remote" (server) computer can signal the "local" (user) computer that it is time to pass control to the user of the terminal. It should be transmitted at those times, and only at those times, when the user should be given control of the terminal. Note that premature transmission of the GA command may result in the blocking of output, since the user is likely to assume that the transmitting system has paused, and therefore he will fail to turn the line around manually.

The foregoing, of course, does not apply to the user-to-server direction of communication. In this direction, GAs may be sent at any time, but need not ever be sent. Also, if the TELNET connection is being used for process-to-process communication, GAs need not be sent in either direction. Finally, for terminal-to-terminal communication, GAs may be required in neither, one, or both directions. If a host plans to support terminal-to-terminal communication it is suggested that the host provide the user with a means of manually signaling that it is time for a GA to be sent over the TELNET connection; this, however, is not a requirement on the implementer of a TELNET process.

Note that the symmetry of the TELNET model requires that there is an NVT at each end of the TELNET connection, at least conceptually.

STANDARD REPRESENTATION OF CONTROL FUNCTIONS

As stated in the Introduction to this document, the primary goal of the TELNET protocol is the provision of a standard interfacing of terminal devices and terminal-oriented processes through the network. Early experiences with this type of interconnection have shown that certain functions are implemented by most servers, but that the methods of invoking these functions differ widely. For a human user who interacts with several server systems, these differences are highly frustrating. TELNET, therefore, defines a standard representation for five of these functions, as described

below. These standard representations have standard, but not required, meanings (with the exception that the Interrupt Process (IP) function may be required by other protocols which use TELNET); that is, a system which does not provide the function to local users need not provide it to network users and may treat the standard representation for the function as a No-operation. On the other hand, a system which does provide the function to a local user is obliged to provide the same function to a network user who transmits the standard representation for the function.

Interrupt Process (IP)

Many systems provide a function which suspends, interrupts, aborts, or terminates the operation of a user process. This function is frequently used when a user believes his process is in an unending loop, or when an unwanted process has been inadvertently activated. IP is the standard representation for invoking this function. It should be noted by implementers that IP may be required by other protocols which use TELNET, and therefore should be implemented if these other protocols are to be supported.

Abort Output (AO)

Many systems provide a function which allows a process, which is generating output, to run to completion (or to reach the same stopping point it would reach if running to completion) but without sending the output to the user's terminal. Further, this function typically clears any output already produced but not yet actually printed (or displayed) on the user's terminal. AO is the standard representation for invoking this function. For example, some subsystem might normally accept a user's command, send a long text string to the user's terminal in response, and finally signal readiness to accept the next command by sending a "prompt" character (preceded by <CR><LF>) to the user's terminal. If the AO were received during the transmission of the text string, a reasonable implementation would be to suppress the remainder of the text string, but transmit the prompt character and the preceding <CR><LF>. (This is possibly in distinction to the action which might be taken if an IP were received; the IP might cause suppression of the text string and an exit from the subsystem.)

It should be noted, by server systems which provide this function, that there may be buffers external to the system (in

the network and the user's local host) which should be cleared; the appropriate way to do this is to transmit the "Synch" signal (described below) to the user system.

Are You There (AYT)

Many systems provide a function which provides the user with some visible (e.g., printable) evidence that the system is still up and running. This function may be invoked by the user when the system is unexpectedly "silent" for a long time, because of the unanticipated (by the user) length of a computation, an unusually heavy system load, etc. AYT is the standard representation for invoking this function.

Erase Character (EC)

Many systems provide a function which deletes the last preceding undeleted character or "print position"* from the stream of data being supplied by the user. This function is typically used to edit keyboard input when typing mistakes are made. EC is the standard representation for invoking this function.

*NOTE: A "print position" may contain several characters which are the result of overstrikes, or of sequences such as <char1> BS <char2>...

Erase Line (EL)

Many systems provide a function which deletes all the data in the current "line" of input. This function is typically used to edit keyboard input. EL is the standard representation for invoking this function.

THE TELNET "SYNCH" SIGNAL

Most time-sharing systems provide mechanisms which allow a terminal user to regain control of a "runaway" process; the IP and AO functions described above are examples of these mechanisms. Such systems, when used locally, have access to all of the signals supplied by the user, whether these are normal characters or special "out of band" signals such as those supplied by the teletype "BREAK" key or the IBM 2741 "ATTN" key. This is not necessarily true when terminals are connected to the system through the network; the network's flow control mechanisms may cause such a signal to be buffered elsewhere, for example in the user's host.

By convention the sequence [IP, Synch] is to be used as such a signal. For example, suppose that some other protocol, which uses TELNET, defines the character string STOP analogously to the TELNET command AO. Imagine that a user of this protocol wishes a server to process the STOP string, but the connection is blocked because the server is processing other commands. The user should instruct his system to:

1. Send the TELNET IP character;

2. Send the TELNET SYNC sequence, that is:

   Send the Data Mark (DM) as the only character in a TCP urgent mode send operation.

3. Send the character string STOP; and

4. Send the other protocol's analog of the TELNET DM, if any.

The user (or process acting on his behalf) must transmit the TELNET SYNCH sequence of step 2 above to ensure that the TELNET IP gets through to the server's TELNET interpreter.

The Urgent should wake up the TELNET process; the IP should wake up the next higher level process.

THE NVT PRINTER AND KEYBOARD

The NVT printer has an unspecified carriage width and page length and can produce representations of all 95 USASCII graphics (codes 32 through 126). Of the 33 USASCII control codes (0 through 31 and 127), and the 128 uncovered codes (128 through 255), the following have specified meaning to the NVT printer:

NAME	CODE	MEANING
NULL (NUL)	0	No Operation
Line Feed (LF)	10	Moves the printer to the next print line, keeping the same horizontal position.
Carriage Return (CR)	13	Moves the printer to the left margin of the current line.

To counter this problem, the TELNET "Synch" mechanism is introduced. A Synch signal consists of a TCP Urgent notification, coupled with the TELNET command DATA MARK. The Urgent notification, which is not subject to the flow control pertaining to the TELNET connection, is used to invoke special handling of the data stream by the process which receives it. In this mode, the data stream is immediately scanned for "interesting" signals as defined below, discarding intervening data. The TELNET command DATA MARK (DM) is the synchronizing mark in the data stream which indicates that any special signal has already occurred and the recipient can return to normal processing of the data stream.

The Synch is sent via the TCP send operation with the Urgent flag set and the DM as the last (or only) data octet.

When several Synchs are sent in rapid succession, the Urgent notifications may be merged. It is not possible to count Urgents since the number received will be less than or equal the number sent. When in normal mode, a DM is a no operation; when in urgent mode, it signals the end of the urgent processing.

If TCP indicates the end of Urgent data before the DM is found, TELNET should continue the special handling of the data stream until the DM is found.

If TCP indicates more Urgent data after the DM is found, it can only be because of a subsequent Synch. TELNET should continue the special handling of the data stream until another DM is found.

"Interesting" signals are defined to be: the TELNET standard representations of IP, AO, and AYT (but not EC or EL); the local analogs of these standard representations (if any); all other TELNET commands; other site-defined signals which can be acted on without delaying the scan of the data stream.

Since one effect of the SYNCH mechanism is the discarding of essontially all characters (except TELNET commands) between the sender of the Synch and its recipient, this mechanism is specified as the standard way to clear the data path when that is desired. For example, if a user at a terminal causes an AO to be transmitted, the server which receives the AO (if it provides that function at all) should return a Synch to the user.

Finally, just as the TCP Urgent notification is needed at the TELNET level as an out-of-band signal, so other protocols which make use of TELNET may require a TELNET command which can be viewed as an out-of-band signal at a different level.

In addition, the following codes shall have defined, but not required, effects on the NVT printer. Neither end of a TELNET connection may assume that the other party will take, or will have taken, any particular action upon receipt or transmission of these:

BELL (BEL)	7	Produces an audible or visible signal (which does NOT move the print head).
Back Space (BS)	8	Moves the print head one character position towards the left margin.
Horizontal Tab (HT)	9	Moves the printer to the next horizontal tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Vertical Tab (VT)	11	Moves the printer to the next vertical tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Form Feed (FF)	12	Moves the printer to the top of the next page, keeping the same horizontal position.

All remaining codes do not cause the NVT printer to take any action.

The sequence "CR LF", as defined, will cause the NVT to be positioned at the left margin of the next print line (as would, for example, the sequence "LF CR"). However, many systems and terminals do not treat CR and LF independently, and will have to go to some effort to simulate their effect. (For example, some terminals do not have a CR independent of the LF, but on such terminals it may be possible to simulate a CR by backspacing.) Therefore, the sequence "CR LF" must be treated as a single "new line" character and used whenever their combined action is intended; the sequence "CR NUL" must be used where a carriage return alone is actually desired; and the CR character must be avoided in other contexts. This rule gives assurance to systems which must decide whether to perform a "new line" function or a multiple-backspace that the TELNET stream contains a character following a CR that will allow a rational decision.

Note that "CR LF" or "CR NUL" is required in both directions

(in the default ASCII mode), to preserve the symmetry of the NVT model. Even though it may be known in some situations (e.g., with remote echo and suppress go ahead options in effect) that characters are not being sent to an actual printer, nonetheless, for the sake of consistency, the protocol requires that a NUL be inserted following a CR not followed by a LF in the data stream. The converse of this is that a NUL received in the data stream after a CR (in the absence of options negotiations which explicitly specify otherwise) should be stripped out prior to applying the NVT to local character set mapping.

The NVT keyboard has keys, or key combinations, or key sequences, for generating all 128 USASCII codes. Note that although many have no effect on the NVT printer, the NVT keyboard is capable of generating them.

In addition to these codes, the NVT keyboard shall be capable of generating the following additional codes which, except as noted, have defined, but not required, meanings. The actual code assignments for these "characters" are in the TELNET Command section, because they are viewed as being, in some sense, generic and should be available even when the data stream is interpreted as being some other character set.

Synch

This key allows the user to clear his data path to the other party. The activation of this key causes a DM (see command section) to be sent in the data stream and a TCP Urgent notification is associated with it. The pair DM-Urgent is to have required meaning as defined previously.

Break (BRK)

This code is provided because it is a signal outside the USASCII set which is currently given local meaning within many systems. It is intended to indicate that the Break Key or the Attention Key was hit. Note, however, that this is intended to provide a 129th code for systems which require it, not as a synonym for the IP standard representation.

Interrupt Process (IP)

Suspend, interrupt, abort or terminate the process to which the NVT is connected. Also, part of the out-of-band signal for other protocols which use TELNET.

Abort Output (AO)

Allow the current process to (appear to) run to completion, but do not send its output to the user. Also, send a Synch to the user.

Are You There (AYT)

Send back to the NVT some visible (i.e., printable) evidence that the AYT was received.

Erase Character (EC)

The recipient should delete the last preceding undeleted character or "print position" from the data stream.

Erase Line (EL)

The recipient should delete characters from the data stream back to, but not including, the last "CR LF" sequence sent over the TELNET connection.

The spirit of these "extra" keys, and also the printer format effectors, is that they should represent a natural extension of the mapping that already must be done from "NVT" into "local". Just as the NVT data byte 68 (104 octal) should be mapped into whatever the local code for "uppercase D" is, so the EC character should be mapped into whatever the local "Erase Character" function is. Further, just as the mapping for 124 (174 octal) is somewhat arbitrary in an environment that has no "vertical bar" character, the EL character may have a somewhat arbitrary mapping (or none at all) if there is no local "Erase Line" facility. Similarly for format effectors: if the terminal actually does have a "Vertical Tab", then the mapping for VT is obvious, and only when the terminal does not have a vertical tab should the effect of VT be unpredictable.

TELNET COMMAND STRUCTURE

All TELNET commands consist of at least a two byte sequence: the "Interpret as Command" (IAC) escape character followed by the code for the command. The commands dealing with option negotiation are three byte sequences, the third byte being the code for the option referenced. This format was chosen so that as more comprehensive use of the "data space" is made -- by negotiations from the basic NVT, of course -- collisions of data bytes with reserved command values will be minimized, all such collisions requiring the inconvenience, and

inefficiency, of "escaping" the data bytes into the stream. With the current set-up, only the IAC need be doubled to be sent as data, and the other 255 codes may be passed transparently.

The following are the defined TELNET commands. Note that these codes and code sequences have the indicated meaning only when immediately preceded by an IAC.

NAME	CODE	MEANING
SE	240	End of subnegotiation parameters.
NOP	241	No operation.
Data Mark	242	The data stream portion of a Synch. This should always be accompanied by a TCP Urgent notification.
Break	243	NVT character BRK.
Interrupt Process	244	The function IP.
Abort output	245	The function AO.
Are You There	246	The function AYT.
Erase character	247	The function EC.
Erase Line	248	The function EL.
Go ahead	249	The GA signal.
SB	250	Indicates that what follows is subnegotiation of the indicated option.
WILL (option code)	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option.
WON'T (option code)	252	Indicates the refusal to perform, or continue performing, the indicated option.
DO (option code)	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the indicated option.
DON'T (option code)	254	Indicates the demand that the other party stop performing, or confirmation that you are no longer expecting the other party to perform, the indicated option.
IAC	255	Data Byte 255.

CONNECTION ESTABLISHMENT

The TELNET TCP connection is established between the user's port U and the server's port L. The server listens on its well known port L for such connections. Since a TCP connection is full duplex and identified by the pair of ports, the server can engage in many simultaneous connections involving its port L and different user ports U.

Port Assignment

When used for remote user access to service hosts (i.e., remote terminal access) this protocol is assigned server port 23 (27 octal). That is L=23.

Network Working Group
Request for Comments: 855

J. Postel
J. Reynolds
ISI

Obsoletes: NIC 18640

May 1983

TELNET OPTION SPECIFICATIONS

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

The intent of providing for options in the TELNET Protocol is to permit hosts to obtain more elegant solutions to the problems of communication between dissimilar devices than is possible within the framework provided by the Network Virtual Terminal (NVT). It should be possible for hosts to invent, test, or discard options at will. Nevertheless, it is envisioned that options which prove to be generally useful will eventually be supported by many hosts; therefore it is desirable that options should be carefully documented and well publicized. In addition, it is necessary to insure that a single option code is not used for several different options.

This document specifies a method of option code assignment and standards for documentation of options. The individual responsible for assignment of option codes may waive the requirement for complete documentation for some cases of experimentation, but in general documentation will be required prior to code assignment. Options will be publicized by publishing their documentation as RFCs; inventors of options may, of course, publicize them in other ways as well.

Option codes will be assigned by:

Jonathan B. Postel
University of Southern California
Information Sciences Institute (USC-ISI)
4676 Admiralty Way
Marina Del Rey, California 90291
(213) 822-1511

Mailbox = POSTEL@USC-ISIF

Documentation of options should contain at least the following sections:

Section 1 - Command Name and Option Code

Section 2 - Command Meanings

The meaning of each possible TELNET command relevant to this option should be described. Note that for complex options, where

"subnegotiation" is required, there may be a larger number of possible commands. The concept of "subnegotiation" is described in more detail below.

Section 3 - Default Specification

The default assumptions for hosts which do not implement, or use, the option must be described.

Section 4 - Motivation

A detailed explanation of the motivation for inventing a particular option, or for choosing a particular form for the option, is extremely helpful to those who are not faced (or don't realize that they are faced) by the problem that the option is designed to solve.

Section 5 - Description (or Implementation Rules)

Merely defining the command meanings and providing a statement of motivation are not always sufficient to insure that two implementations of an option will be able to communicate. Therefore, a more complete description should be furnished in most cases. This description might take the form of text, a sample implementation, hints to implementers, etc.

A Note on "Subnegotiation"

Some options will require more information to be passed between hosts than a single option code. For example, any option which requires a parameter is such a case. The strategy to be used consists of two steps: first, both parties agree to "discuss" the parameter(s) and, second, the "discussion" takes place.

The first step, agreeing to discuss the parameters, takes place in the normal manner; one party proposes use of the option by sending a DO (or WILL) followed by the option code, and the other party accepts by returning a WILL (or DO) followed by the option code. Once both parties have agreed to use the option, subnegotiation takes place by using the command SB, followed by the option code, followed by the parameter(s), followed by the command SE. Each party is presumed to be able to parse the parameter(s), since each has indicated that the option is supported (via the initial exchange of WILL and DO). On the other hand, the receiver may locate the end of a parameter string by searching for the SE command (i.e., the string IAC SE), even if the receiver is unable to parse the parameters. Of course, either party may refuse to pursue further subnegotiation at any time by sending a WON'T or DON'T to the other party.

Thus, for option "ABC", which requires subnegotiation, the formats of the TELNET commands are:

IAC WILL ABC

Offer to use option ABC (or favorable acknowledgment of other party's request)

IAC DO ABC

Request for other party to use option ABC (or favorable acknowledgment of other party's offer)

IAC SB ABC <parameters> IAC SE

One step of subnegotiation, used by either party.

Designers of options requiring "subnegotiation" must take great care to avoid unending loops in the subnegotiation process. For example, if each party can accept any value of a parameter, and both parties suggest parameters with different values, then one is likely to have an infinite oscillation of "acknowledgments" (where each receiver believes it is only acknowledging the new proposals of the other). Finally, if parameters in an option "subnegotiation" include a byte with a value of 255, it is necessary to double this byte in accordance the general TELNET rules.

Network Working Group
Request for Comments: 856

J. Postel
J. Reynolds
ISI
May 1983

Obsoletes: NIC 15389

TELNET BINARY TRANSMISSION

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

1. Command Name and Code

TRANSMIT-BINARY 0

2. Command Meanings

IAC WILL TRANSMIT-BINARY

The sender of this command REQUESTS permission to begin transmitting, or confirms that it will now begin transmitting characters which are to be interpreted as 8 bits of binary data by the receiver of the data.

IAC WON'T TRANSMIT-BINARY

If the connection is already being operated in binary transmission mode, the sender of this command DEMANDS to begin transmitting data characters which are to be interpreted as standard NVT ASCII characters by the receiver of the data. If the connection is not already being operated in binary transmission mode, the sender of this command REFUSES to begin transmitting characters which are to be interpreted as binary characters by the receiver of the data (i.e., the sender of the data demands to continue transmitting characters in its present mode).

A connection is being operated in binary transmission mode only when one party has requested it and the other has acknowledged it.

IAC DO TRANSMIT-BINARY

The sender of this command REQUESTS that the sender of the data start transmitting, or confirms that the sender of data is expected to transmit, characters which are to be interpreted as 8 bits of binary data (i.e., by the party sending this command).

IAC DON'T TRANSMIT-BINARY

If the connection is already being operated in binary transmission mode, the sender of this command DEMANDS that the sender of the data start transmitting characters which are to be interpreted as

standard NVT ASCII characters by the receiver of the data (i.e., the party sending this command). If the connection is not already being operated in binary transmission mode, the sender of this command DEMANDS that the sender of data continue transmitting characters which are to be interpreted in the present mode.

A connection is being operated in binary transmission mode only when one party has requested it and the other has acknowledged it.

3. Default

WON'T TRANSMIT-BINARY

DON'T TRANSMIT-BINARY

The connection is not operated in binary mode.

4. Motivation for the Option

It is sometimes useful to have available a binary transmission path within TELNET without having to utilize one of the more efficient, higher level protocols providing binary transmission (such as the File Transfer Protocol). The use of the IAC prefix within the basic TELNET protocol provides the option of binary transmission in a natural way, requiring only the addition of a mechanism by which the parties involved can agree to INTERPRET the characters transmitted over a TELNET connection as binary data.

5. Description of the Option

With the binary transmission option in effect, the receiver should interpret characters received from the transmitter which are not preceded with IAC as 8 bit binary data, with the exception of IAC followed by IAC which stands for the 8 bit binary data with the decimal value 255. IAC followed by an effective TELNET command (plus any additional characters required to complete the command) is still the command even with the binary transmission option in effect. IAC followed by a character which is not a defined TELNET command has the same meaning as IAC followed by NOP, although an IAC followed by an undefined command should not normally be sent in this mode.

6. Implementation Suggestions

It is foreseen that implementations of the binary transmission option will choose to refuse some other options (such as the EBCDIC transmission option) while the binary transmission option is in

effect. However, if a pair of hosts can understand being in binary transmission mode simultaneous with being in, for example, echo mode, then it is all right if they negotiate that combination.

It should be mentioned that the meanings of WON'T and DON'T are dependent upon whether the connection is presently being operated in binary mode or not. Consider a connection operating in, say, EBCDIC mode which involves a system which has chosen not to implement any knowledge of the binary command. If this system were to receive a DO TRANSMIT-BINARY, it would not recognize the TRANSMIT-BINARY option and therefore would return a WON'T TRANSMIT-BINARY. If the default for the WON'T TRANSMIT-BINARY were always NVT ASCII, the sender of the DO TRANSMIT-BINARY would expect the recipient to have switched to NVT ASCII, whereas the receiver of the DO TRANSMIT-BINARY would not make this interpretation.

Thus, we have the rule that when a connection is not presently operating in binary mode, the default (i.e., the interpretation of WON'T and DON'T) is to continue operating in the current mode, whether that is NVT ASCII, EBCDIC, or some other mode. This rule, however, is not applied once a connection is operating in a binary mode (as agreed to by both ends); this would require each end of the connection to maintain a stack, containing all of the encoding-method transitions which had previously occurred on the connection, in order to properly interpret a WON'T or DON'T. Thus, a WON'T or DON'T received after the connection is operating in binary mode causes the encoding method to revert to NVT ASCII.

It should be remembered that a TELNET connection is a two way communication channel. The binary transmission mode must be negotiated separately for each direction of data flow, if that is desired.

Implementation of the binary transmission option, as is the case with implementations of all other TELNET options, must follow the loop preventing rules given in the General Considerations section of the TELNET Protocol Specification.

Consider now some issues of binary transmission both to and from both a process and a terminal:

a. Binary transmission from a terminal.

   The implementer of the binary transmission option should consider how (or whether) a terminal transmitting over a TELNET connection with binary transmission in effect is allowed to generate all eight bit characters, ignoring parity considerations, etc., on input from the terminal.

b. Binary transmission to a process.

The implementer of the binary transmission option should consider how (or whether) all characters are passed to a process receiving over a connection with binary transmission in effect. As an example of the possible problem, TOPS-20 intercepts certain characters (e.g., ETX, the terminal control-C) at monitor level and does not pass them to the process.

c. Binary transmission from a process.

The implementer of the binary transmission option should consider how (or whether) a process transmitting over a connection with binary transmission in effect is allowed to send all eight bit characters with no characters intercepted by the monitor and changed to other characters. An example of such a conversion may be found in the TOPS-20 system where certain non-printing characters are normally converted to a Circumflex (up-arrow) followed by a printing character.

d. Binary transmission to a terminal.

The implementer of the binary transmission option should consider how (or whether) all characters received over a connection with binary transmission in effect are sent to a local terminal. At issue may be the addition of timing characters normally inserted locally, parity calculations, and any normal code conversion.

Network Working Group
Request for Comments: 857

Obsoletes: NIC 15390

J. Postel
J. Reynolds
ISI
May 1983

TELNET ECHO OPTION

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

1. Command Name and Code

   ECHO 1

2. Command Meanings

   IAC WILL ECHO

      The sender of this command REQUESTS to begin, or confirms that it will now begin, echoing data characters it receives over the TELNET connection back to the sender of the data characters.

   IAC WON'T ECHO

      The sender of this command DEMANDS to stop, or refuses to start, echoing the data characters it receives over the TELNET connection back to the sender of the data characters.

   IAC DO ECHO

      The sender of this command REQUESTS that the receiver of this command begin echoing, or confirms that the receiver of this command is expected to echo, data characters it receives over the TELNET connection back to the sender.

   IAC DON'T ECHO

      The sender of this command DEMANDS the receiver of this command stop, or not start, echoing data characters it receives over the TELNET connection.

3. Default

   WON'T ECHO

   DON'T ECHO

      No echoing is done over the TELNET connection.

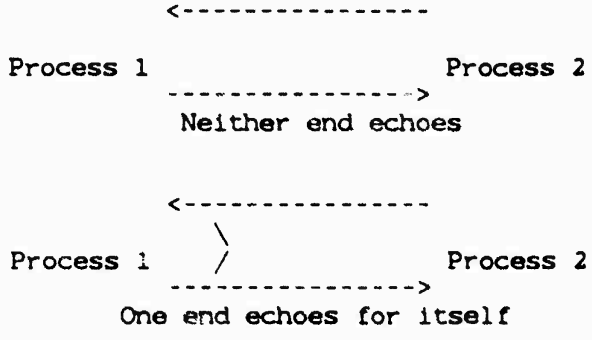
4. Motivation for the Option

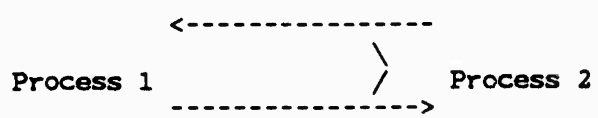
The NVT has a printer and a keyboard which are nominally interconnected so that "echoes" need never traverse the network; that is to say, the NVT nominally operates in a mode where characters typed on the keyboard are (by some means) locally turned around and printed on the printer. In highly interactive situations it is appropriate for the remote process (command language interpreter, etc.) to which the characters are being sent to control the way they are echoed on the printer. In order to support such interactive situations, it is necessary that there be a TELNET option to allow the parties at the two ends of the TELNET connection to agree that characters typed on an NVT keyboard are to be echoed by the party at the other end of the TELNET connection.

5. Description of the Option

When the echoing option is in effect, the party at the end performing the echoing is expected to transmit (echo) data characters it receives back to the sender of the data characters. The option does not require that the characters echoed be exactly the characters received (for example, a number of systems echo the ASCII ESC character with something other than the ESC character). When the echoing option is not in effect, the receiver of data characters should not echo them back to the sender; this, of course, does not prevent the receiver from responding to data characters received.

The normal TELNET connection is two way. That is, data flows in each direction on the connection independently; and neither, either, or both directions may be operating simultaneously in echo mode. There are five reasonable modes of operation for echoing on a connection pair:





One end echoes for the other

Both ends echo for themselves

One end echoes for both ends

This option provides the capability to decide on whether or not either end will echo for the other. It does not, however, provide any control over whether or not an end echoes for itself; this decision must be left to the sole discretion of the systems at each end (although they may use information regarding the state of "remote" echoing negotiations in making this decision).

It should be noted that if BOTH hosts enter the mode of echoing characters transmitted by the other host, then any character transmitted in either direction will be "echoed" back and forth indefinitely. Therefore, care should be taken in each implementation that if one site is echoing, echoing is not permitted to be turned on at the other.

As discussed in the TELNET Protocol Specification, both parties to a full-duplex TELNET connection initially assume each direction of the connection is being operated in the default mode which is non-echo (non-echo is not using this option, and the same as DON'T ECHO, WON'T ECHO).

If either party desires himself to echo characters to the other party or for the other party to echo characters to him, that party gives the appropriate command (WILL ECHO or DO ECHO) and waits (and hopes) for acceptance of the option. If the request to operate the connection in echo mode is refused, then the connection continues to operate in non-echo mode. If the request to operate the connection in echo mode is accepted, the connection is operated in echo mode.

After a connection has been changed to echo mode, either party may demand that it revert to non-echo mode by giving the appropriate DON'T ECHO or WON'T ECHO command (which the other party must confirm thereby allowing the connection to operate in non-echo mode). Just as each direction of the TELNET connection may be put in remote echoing mode independently, each direction of the TELNET connection must be removed from remote echoing mode separately.

Implementations of the echo option, as implementations of all other TELNET options, must follow the loop preventing rules given in the General Considerations section of the TELNET Protocol Specification. Also, so that switches between echo and non-echo mode can be made with minimal confusion (momentary double echoing, etc.), switches in mode of operation should be made at times precisely coordinated with the reception and transmission of echo requests and demands. For instance, if one party responds to a DO ECHO with a WILL ECHO, all data characters received after the DO ECHO should be echoed and the WILL ECHO should immediately precede the first of the echoed characters.

The echoing option alone will normally not be sufficient to effect what is commonly understood to be remote computer echoing of characters typed on a terminal keyboard--the SUPPRESS-GO AHEAD option will normally have to be invoked in conjunction with the ECHO option to effect character-at-a-time remote echoing.

6. A Sample Implementation of the Option

The following is a description of a possible implementation for a simple user system called "UHOST".

A possible implementation could be that for each user terminal, the UHOST would keep three state bits: whether the terminal echoes for itself (UHOST ECHO always) or not (ECHO mode possible), whether the (human) user prefers to operate in ECHO mode or in non-ECHO mode, and whether the connection from this terminal to the server is in ECHO or non-ECHO mode. We will call these three bits P(hysical), D(esired), and A(ctual).

When a terminal dials up the UHOST the P-bit is set appropriately, the D-bit is set equal to it, and the A-bit is set to non-ECHO. The P-bit and D-bit may be manually reset by direct commands if the user so desires. For example, a user in Hawaii on a "full-duplex" terminal, would choose not to operate in ECHO mode, regardless of the preference of a mainland server. He should direct the UHOST to change his D-bit from ECHO to non-ECHO.

When a connection is opened from the UHOST terminal to a server, the

UHOST would send the server a DO ECHO command if the MIN (with non-ECHO less than ECHO) of the P- and D-bits is different from the A-bit. If a WON'T ECHO or WILL ECHO arrives from the server, the UHOST will set the A-bit to the MIN of the received request, the P-bit, and the D-bit. If this changes the state of the A-bit, the UHOST will send off the appropriate acknowledgment; if it does not, then the UHOST will send off the appropriate refusal if not changing meant that it had to deny the request (i.e., the MIN of the P-and D-bits was less than the received A-request).

If while a connection is open, the UHOST terminal user changes either the P-bit or D-bit, the UHOST will repeat the above tests and send off a DO ECHO or DON'T ECHO, if necessary. When the connection is closed, the UHOST would reset the A-bit to indicate UHOST echoing.

While the UHOST's implementation would not involve DO ECHO or DON'T ECHO commands being sent to the server except when the connection is opened or the user explicitly changes his echoing mode, bigger hosts might invoke such mode switches quite frequently. For instance, while a line-at-a-time system were running, the server might attempt to put the user in local echo mode by sending the WON'T ECHO command to the user; but while a character-at-a-time system were running, the server might attempt to invoke remote echoing for the user by sending the WILL ECHO command to the user. Furthermore, while the UHOST will never send a WILL ECHO command and will only send a WON'T ECHO to refuse a server sent DO ECHO command, a server host might often send the WILL and WON'T ECHO commands.

TELNET RECONNECTION OPTION

1. Command name and code

RCP 2 (prepare to reconnect)

2. Command meanings.

IAC DO RCP

The sender of this command requests the receiver of the command to be prepared to break the TELNET connection with the sender of the command and to re-establish the TELNET connection with some other party (to be specified later).

IAC WILL RCP

The receiver of this command agrees to break its TELNET connection to the sender of the DO RCP command and to re-establish the connection with the party to be specified by the sender of the DO RCP command.

IAC WON'T RCP

The receiver of this command refuses to take part in a reconnection.

IAC DON'T RCP

The sender of this command demands the cancellation of its previous DO RCP command.

IAC SB RCP RCS <host> <socket>

The sender of this command instructs the receiver of the command to transfer this TELNET connection to the place specified by <host> <socket>. The code for RCS is 0.

IAC SB RCP RCW <host> <socket>

The sender of this command instructs the receiver of the command to break the TELNET connection and to await a new TELNET connection from the place specified by <host> <socket>. The code for RCW is 1.

TELNET Reconnection Option
NIC 15391 (Aug. 1973)

3. Default.

WON'T RCP

i.e., no reconnection is allowed.

4. Motivation for the option.

There are situations in which it is desirable to move one or both ends of a communication path from one Host to another.

A. Consider the case of an executive program which TIP users could use to get network status information, send messages, link to other users, etc. Due to the TIP's limited resources the executive program would probably not run on the TIP itself but rather would run on one or more larger Hosts who would be willing to share some of their resources with the TIP (see Figure 1).

The TIP user could access the executive by typing a command such as "@EXEC"; the TIP should then ICP to Host1's executive port. After obtaining the latest network news and perhaps sending a few messages, theuser would be ready to log into Host2 (in general not the same as Host1) and do some work. At that point he would like to tell the executive program that he is ready to use Host2 and have the executive hand him offto Host2. To do this the executive program would first interact with Host2, telling it to expect a call from the TIP, and then would instruct the TIP to reconnect to Host2. When the user logs off Host2 he could be passed back to the executive at Host1 preparatory to doing more elsewhere. The reconnection activity would be invisible to the TIP user.

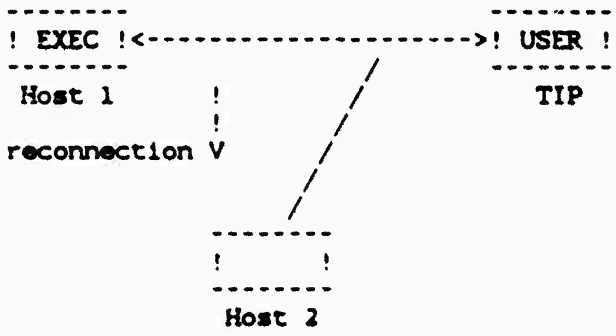


FIGURE 1

B. Imagine a scenario in which a user could use the same name and password (and perhaps account) to log into any server on the network. For reasons of security and economy it would be undersirable to have every name and password stored at every site. A user wanting to use a Host that doesn't have his name or password locally would connect to it and attempt to log in as usual (see Figure 2). The Host, discovering that it doesn't know the user, would hand him off to a network authentication service which can determine whether the user is who he claims to be. If the user passes the authentication test he can be handed back to the Host which can then provide him service.

If the user doesn't trust the Host and is afraid that it might read his password rather than pass him off to the Authenticator he could connect directly to the authentication service. After authentication, the Authenticator can pass him off to the Host.

The idea is that the shuffling of the user back and forth between Host and Authenticator should be invisible to the user.

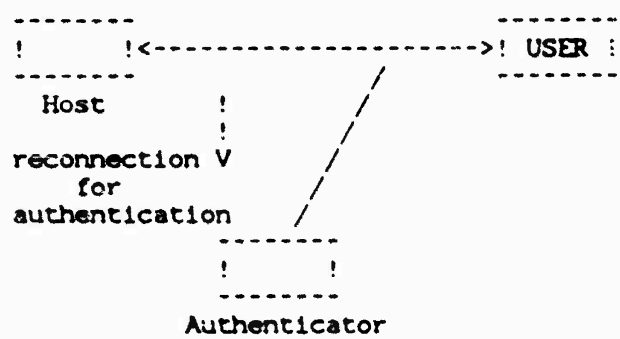


FIGURE 2a

TELNET Reconnection Option
NIC 15391 (Aug. 1973)

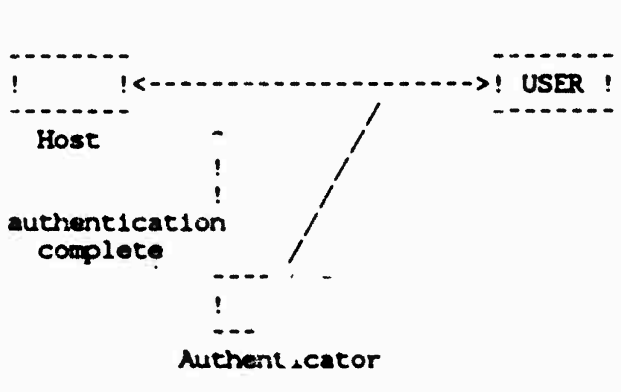


FIGURE 2b

C. The McROSS air traffic simulation system (see 1972 SJCC paper by Thomas) already supports reconnection. It permits an on - going simulation to reconfigure itself by allowing parts to move from computer to computer. For example, in a simulation of air traffic in the Northeast, the program fragment simulating the New York Enroute air spacecould move from Host2 to Host5 (see figure 3). As part of the reconfiguration process the New York Terminal area simulator and Boston Enroute area simulators break their connections with the New York Enroute simulator at Host2 and reconnect to it at Host5.

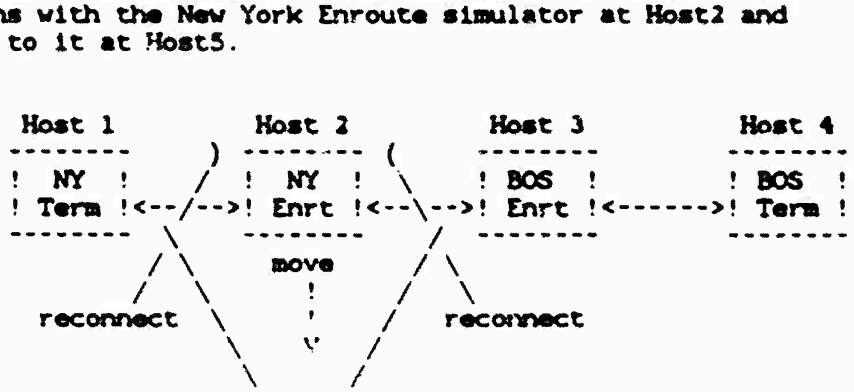


FIGURE 3

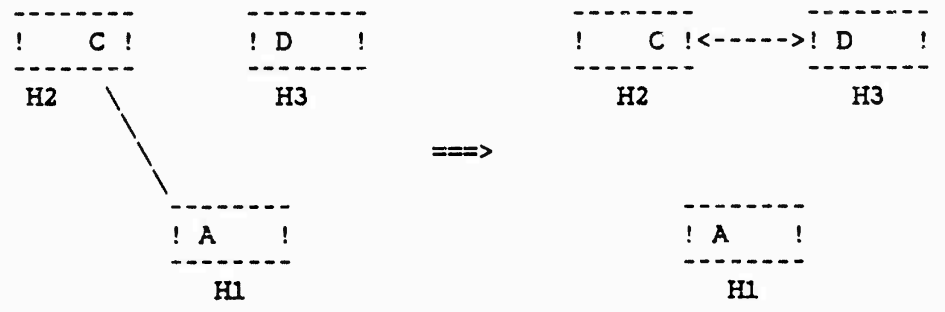
5. An abstract description of a reconnection mechanism.

The reconnection mechanism includes four (abstract) commands:

```
Reconnect Request: RRQ <path>
Reconnect OK:      ROK <path>
Reconnect No:      RNO <path>
Reconnect Do:      RDO <path> <destination>
```

where <path> is a communication path to be redirected to <destination>.

Assume that H1 wants to move its end of communication path A-C from itself to port D at H3 (Figure 4).



(a) situation (b) desired situation

FIGURE 4

The reconnection proceeds by steps:

a. H1 arranges for the reconnection by sending RRQ to H2:

   H1->H2: RRQ (path A-C)

b. H2 agrees to reconnect and acknowledges with ROK:

   H2->H1: ROK (path C-A)

c. H1 notes that H2 has agreed to reconnect and instructs H2 to perform the reconnection;

   H1->H2: RDO (path A-C) (Host H3, PortD)

TELNET Reconnection Option
NIC 15391 (Aug. 1973)

d. H1 breaks paths A-C.

H2 breaks path C-A and initiates path C-D.

In order for the reconnection to succeed H1 must, of course, have arranged for H3's cooperation. One way H1 could do this would be to establish the path B-D and then proceed through the reconnection protocol exchange with H3 concurrently with its exchange with H2 (See Figure 5):

H1->H3: RRQ (path B-D)
H3->H1: ROK (path D-B)
H1->H3: RDO (path B-D) (Host H2, Port C)

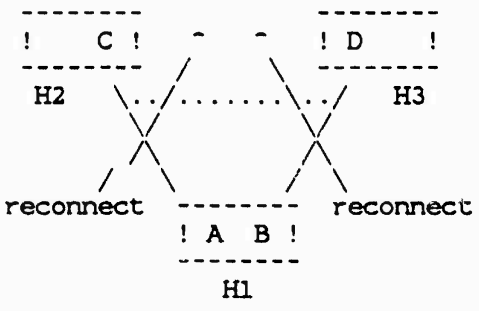
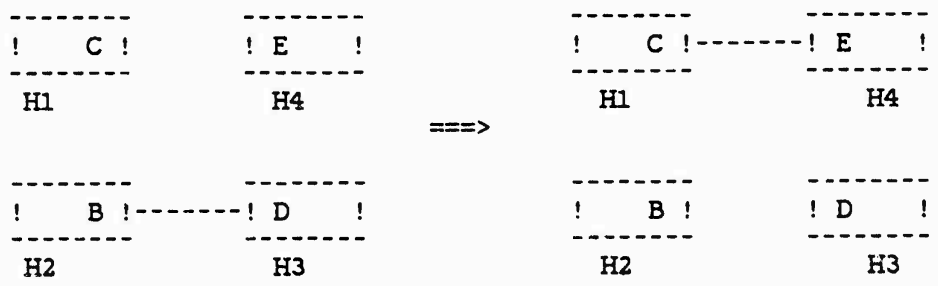


FIGURE 5

Either of the parties may use the RNO command to refuse or abort reconnection. H2 could respond to H1's RRQ with RNO; H1 can abort the reconnection by responding to ROK with RNO rather than RDO.

It is easy to insure that messages in transit are not lost during the reconnection. Receipt of the ROK message by H1 is taken to mean that no further messages are coming from H2; similarly receipt of RDO from H1 by H2 is taken to mean that no further messages are coming from H1.

To complete the specification of the reconnection mechanism consider the situation in which two "adjacent" entities initiate reconnections:



(a) situation (b) desired situation

FIGURE 6

H2 and H3 "simultaneously" try to arrange for reconnection:

H2->H3: RRQ (path B-D)

H3->H2: RRQ (path D-B)

Thus, H2 sees an RRQ from H3 rather than an ROK or RNO in response to its RRQ to H3. This "race" situation can be resolved by having the reconnections proceed in series rather than in parallel: first one entity (say H2) performs its reconnect and then the other (H3) performs its reconnect. There are several means that could be used to decide which gets to go first. Perhaps the simplest is to base the decision on sockets and site addresses: the entity for which the 40 bit number formed by concatenating the 32 bit socket number with the 8 bit site address is largest gets to go first. Using this mechanism the rule is the following:

If H2 receives an RRQ from H3 in response to an RRQ of its own:

(let NH2, NH3 = the 40 bit numbers corresponding to H2 and H3)

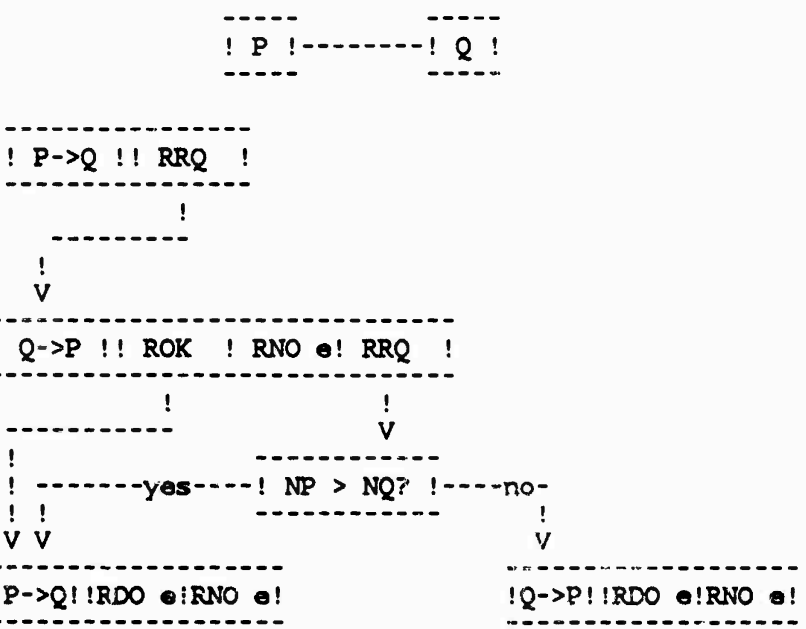
a. if NH2>NH3 then both H2 and H3 interpret H3's RRQ as an ROK in response to H2's RRQ.

b. if NH2<NH3 then both interpret H3's RRQ as an RNO in response to H2's RRQ. This would be the only case in which it makes sense to "ignore" the refusal and try again - of course, waiting until completion of the first reconnect before doing so.

TELNET Reconnection Option
NIC 15391 (Aug. 1973)

Once an ordering has been determined the reconnection proceeds as though there was no conflict.

The following diagram describes the legal protocol command/response exchange sequences for a reconnection initiated by P:



NP and NQ are the 40 bit numbers for P and Q; e indicates end of sequence.

6. A description of the option.

The reconnection mechanism described abstractly in the previous section can be effected as a TELNET option by use of the command RCP. Using this command and the TELNET DO, DON'T, WILL, WON'T, and SB prefixes, the four commands used in the previous abstract description become

RRQ => DO RCP

ROK => WILL RCP

```
RNO =>  WON'T RCP    ;for responses to DO RCP
        DON'T RCP    ;for responses to WILL RCP
                     ;i.e. used to cancel an RCP.
RDO  <host> <socket> => SB RCP RCS <host> <socket>
```

A fifth command is also introduced

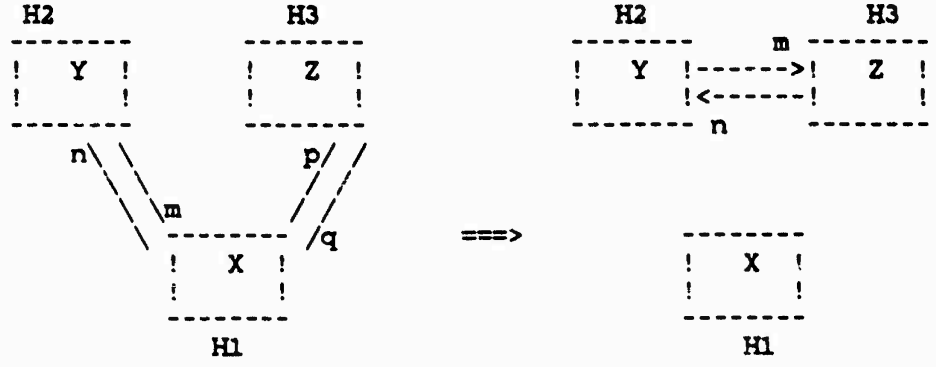
```
RWT  <host> <socket> => SB RCP RCW <host> <socket>
```

The first three commands require no parameters since they refer to the connections they are received on. For RDO and RWT, <host> is an 8 bit (= 1 TELNET character) Host address and <socket> is a 32 bit (= 4 TELNET characters) number that specifies a TELNET receive socket at the specified Host (the associated transmit socket is always one higher than the receive socket.

A pending reconnection can be activiated with either RDO or RWT. The response to either is to first break the TELNET connection with the sender and then reopen the TELNET connection to the Host and sockets specified. For RDO, the connection is to be reopened by sending two RFC's; for RWT, by waiting for two RFC's.

The RWT command is introduced to avoid requiring Hosts to queue RFC's.

As an example, the reconnnection



could be accomplished as follows:

TELNET Reconnection Option
NIC 15391 (Aug. 1973)

```
X->Y:   RRQ         (=IAC DO RCP)
X->Z:   RRQ         (=IAC DO RCP)
Y->X:   ROK         (=IAC WILL RCP)
Z->X:   ROK         (=IAC WILL RCP)
X->Y:   RWT H3 P    (=IAC SB RCP RCW H3 P)
X closes connections to Y
Y closes connections to X
Y waits for STR and RTS from H3
X->Z:   RDO H2 N     (=IAC SB RCP RCS H2 N)
X closes connections to Z
Z closes connections to X
Z sends STR and RTS to H2 which Y answers with matching RTS
and STR to compete reconnection
```

The RCS and RCW sub-commands should never be sent until a DO RCP has been acknowledged by a WILL RCP. Thus a Host not choosing to implement the reconnection option does not have to know what RCP means--all the Host need do in response to DO RCP is to transmit WON'T RCP. The WILL RCP and WON'T RCP commands should never be volunteered. If an unsolicited WILL RCP is ever received, a DON'T RCP should be fired back, which should be answered bya WON'T RCP command. If an unsolicited WON'T RCP command is received, it should be treated as a No-operation.

7. A word about SECURITY.

It should be clear that the decision to accept or reject a particular reconnection request is the responsibility of the entity (person at the terminal or process) using the connection. In many cases the entity may chose to delegate that responsibility to its TELNET (e.g., Example A, Section 4). However, the interface a Host provides to the reconnection mechanism would best include means for local entities to exercise control over response to remotely intitiated reconnection requests. For example, a user-TELNET might support several modes of operation with respect to remotely initiated reconnections:

1. transparent: all requested reconnections are to be performed in a way that is invisible to the user;

2. visible: all requested reconnections are to be performed and the user is to be informed whenever a reconnection occurs;

3. confirmation: the user is to be informed of each reconnection request which he may accept or reject;

4. rejection: all requested reconnects are to be rejected.

Network Working Group
Request for Comments: 858

Obsoletes: NIC 15392

J. Postel
J. Reynolds
ISI
May 1983

TELNET SUPPRESS GO AHEAD OPTION

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

1. Command Name and Code

SUPPRESS-GO-AHEAD 3

2. Command Meanings

IAC WILL SUPPRESS-GO-AHEAD

The sender of this command requests permission to begin suppressing transmission of the TELNET GO AHEAD (GA) character when transmitting data characters, or the sender of this command confirms it will now begin suppressing transmission of GAs with transmitted data characters.

IAC WON'T SUPPRESS-GO-AHEAD

The sender of this command demands to begin transmitting, or to continue transmitting, the GA character when transmitting data characters.

IAC DO SUPPRESS-GO-AHEAD

The sender of this commannd requests that the sender of data start suppressing GA when transmitting data, or the sender of this command confirms that the sender of data is expected to suppress transmission of GAs.

IAC DON'T SUPPRESSS-GO-AHEAD

The sender of this command demands that the receiver of the command start or continue transmitting GAs when transmitting data.

3. Default

WON'T SUPPRESS-GO-AHEAD

DON'T SUPPRESS-GO-AHEAD

Go aheads are transmitted.

4. Motivation for the Option

While the NVT nominally follows a half duplex protocol complete with a GO AHEAD signal, there is no reason why a full duplex connection between a full duplex terminal and a host optimized to handle full duplex terminals should be burdened with the GO AHEAD signal. Therefore, it is desirable to have a TELNET option with which parties involved can agree that one or the other or both should suppress transmission of GO AHEADS.

5. Description of the Option

When the SUPPRESS-GO-AHEAD option is in effect on the connection between a sender of data and the receiver of the data, the sender need not transmit GAs.

It seems probable that the parties to the TELNET connection will suppress GO AHEAD in both directions of the TELNET connection if GO AHEAD is suppressed at all; but, nonetheless, it must be suppressed in both directions independently.

With the SUPPRESS-GO-AHEAD option in effect, the IAC GA command should be treated as a NOP if received, although IAC GA should not normally be sent in this mode.

6. Implementation Considerations

As the SUPRESS-GO-AHEAD option is sort of the opposite of a line at a time mode, the sender of data which is suppressing GO AHEADs should attempt to actually transmit characters as soon as possible (i.e., with minimal buffering) consistent with any other agreements which are in effect.

In many TELNET implementations it will be desirable to couple the SUPPRESS-GO-AHEAD option to the echo option so that when the echo option is in effect, the SUPPRESS-GO-AHEAD option is in effect simultaneously: both of these options will normally have to be in effect simultaneously to effect what is commonly understood to be character at a time echoing by the remote computer.

TELNET Approximate Message Size Negotiation Option
NIC 15393 (Aug. 1973)

TELNET APPROXIMATE MESSAGE SIZE NEGOTIATION OPTION

1. Command name and code.

   NAMS 4

   (Negotiate Approximate Message Size)

2. Command meanings.

   IAC WILL NAMS

   The sender of this command requests, or agrees, to negotiate the approximate size for messages of data characters it sends.

   IAC WON'T NAMS

   The sender of this command refuses to negotiate the approximate size for messages of data characters it sends.

   IAC DO NAMS

   The sender of this command requests the receiver of this command to negotiate the approximate size for messages of data characters transmitted by the command receiver.

   IAC DON'T NAMS

   The sender of this command refuses to negotiate the approximate size for
   messages of data characters transmitted by the command receiver.

   IAC SB NAMS DR <16 bit value>

   The sender of this command requests the receiver of this command to set its approximate message size for data the command receiver transmits to the value specified in the 16 bit parameter, a data character count. Thecode for DR (Data Receiver) is 0.

   IAC SB NAMS DS <16 bit value>

   The sender of this command requests or agrees to set its approximate message size for data it transmits to the value specified in the 16 bit parameter, a data character count. The code for DS (Data Sender) is 1.

TELNET Approximate Message Size Negotiation Option
NIC 15393 (Aug. 1973)

3. Default

WON'T NAMS

DON'T NAMS

i.e., no attempt will be made to agree on a message size.

4. Motivation for the option.

The TELNET protocol does not specify how many characters the transmitter of data should attempt to pack into messages it sends. However, 1) some receivers may prefer received messages to generally have some minimum size, for example, to lessen the burden of processing input interrupts; 2) some receivers may prefer received data messages to generally have some maximum size, for example, because the maximum data message size could be used in conjunction with the Host/Host protocol message and bit allocates to more efficiently utilize input buffer space; 3) some transmitters may have maximum sizes for transmitted data messages, information which could be used in conjunction with the Host/Host protocol message and bit allocates to more efficiently utilize the receiver's input buffer space; and 4) some transmitters may desire to transmit some minimum size message, for example, to lessen the burden of processing output interrupts.

Therefore, it is desirable to have some mechanism whereby the parties involved can attempt to agree on the approximate size of messages transmitted over the connection. (It might be even more powerful to be able to negotiate approximate or even exact upper and lower bounds on message size. However, fixed bounds would sometimes be hard to manage and sometimes even in conflict with Host/Host protocol allocates; and specifying both upper and lower bounds, even approximately, seems overly complicated considering the expected payoff.)

5. Description of the option.

With the option which specifies the approximate size of messages transmitted over the connection, the transmitter attempts to send messages of the specified size unless some other constraint (for instance, an end of line) requires the message to be sent sooner, or characters for transmission arrive so fast that the message has to be bigger than the specified size. The option is to be used strictly to improve the STATISTICS (e.g., timing and buffering) of message reception and transmission -- the option does NOT specify any absolutes.

With this option not in effect, message size is completely (even statistically) undefined as per the NVT specification.

Once the data transmitter and receiver have agreed to negotiate the approximate message size, they must actually do this negotiation. This is done using the DS and DR SB commands. The transmitter of data messages may give the SB NAMS DS command and the receiver may give the SB NAMS DR command. The rules for negotiation of the acutal aproximate message size are as follows:

a. Either party may at any time send a SB command specifying a value less than any previously sent or received and immediately assume that that value has been agreed upon.

b. If either party receives a SB command, the party should assume the value specified in the received command is in effect if the party has not previously sent a SB command specifying a lower value.

c. Before any SB command is sent, the approximate message size is undefined.

d. At any time either party may quit the whole thing by sending a DON'T or WON'T NAMS command which must be acknowledged and the approximate message length becomes undefined.

e. An approximate message size value may not be less than one.

As the receiver and transmitter may have conflicting requirements for the approximate message size, neither should be cavalier about requesting a specified approximate message size, each "bending over backward" to let the other party (who should be presumed to have a greater need) specify the approximate message size.

Host/Host protocol allocate considerations, of course, always dominate negotiated message size considerations.

Network Working Group
Request for Comments: 859

Obsoletes: RFC 651 (NIC 31154)

J. Postel
J. Reynolds
ISI

May 1983

TELNET STATUS OPTION

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

1. Command Name and Code

   STATUS 5

2. Command Meanings

   This option applies separately to each direction of data flow.

   IAC DON'T STATUS

      Sender refuses to carry on any further discussion of the current status of options.

   IAC WON'T STATUS

      Sender refuses to carry on any further discussion of the current status of options.

   IAC SB STATUS SEND IAC SE

      Sender requests receiver to transmit his (the receiver's) perception of the current status of Telnet options. The code for SEND is 1. (See below.)

   IAC SB STATUS IS ... IAC SE

      Sender is stating his perception of the current status of Telnet options. The code for IS is 0. (See below.)

3. Default

   DON'T STATUS, WON'T STATUS

      The current status of options will not be discussed.

4. Motivation for the Option

   This option allows a user/process to verify the current status of TELNET options (e.g., echoing) as viewed by the person/process on the other end of the TELNET connection. Simply renegotiating options

could lead to the nonterminating request loop problem discussed in the General Consideration section of the TELNET Specification. This option fits into the normal structure of TELNET options by deferring the actual transfer of status information to the SB command.

5. Description of the Option

WILL and DO are used only to obtain and grant permission for future discussion. The actual exchange of status information occurs within option subcommands (IAC SB STATUS...).

Once the two hosts have exchanged a WILL and a DO, the sender of the WILL STATUS is free to transmit status information, spontaneously or in response to a request from the sender of the DO. At worst, this may lead to transmitting the information twice. Only the sender of the DO may send requests (IAC SB STATUS SEND IAC SE) and only the sender of the WILL may transmit actual status information (within an IAC SB STATUS IS ... IAC SE command).

IS has the subcommands WILL, DO and SB. They are used EXACTLY as used during the actual negotiation of TELNET options, except that SB is terminated with SE, rather than IAC SE. Transmission of SE, as a regular data byte, is accomplished by doubling the byte (SE SE). Options that are not explicitly described are assumed to be in their default states. A single IAC SB STATUS IS ... IAC SE describes the condition of ALL options.

The following is an example of use of the option:

Host1: IAC DO STATUS

Host2: IAC WILL STATUS

(Host2 is now free to send status information at any time. Solicitations from Host1 are NOT necessary. This should not produce any dangerous race conditions. At worst, two IS's will be sent.)

Host1 (perhaps): IAC SB STATUS SEND IAC SE

Host2 (the following stream is broken into multiple lines only for readability. No carriage returns are implied.):

IAC SB STATUS IS

WILL ECHO

DO SUPPRESS-GO-AHEAD

WILL STATUS

DO STATUS

IAC SE

Explanation of Host2's perceptions: It is responsible for echoing back the data characters it receives over the TELNET connection; it will not send Go-Ahead signals; it will both issue and request Status information.

Suppose that Process A of Figure 1 wishes to synchronize with B. The DO TIMING-MARK is sent from A to B. B can refuse by replying WON'T TIMING-MARK, or agree by permitting the timing mark to flow through his "outgoing" buffer, BUF2. Then, instead of delivering it to the terminal, B will enter the mark into his "incoming" buffer BUF1, to flow through toward A. When the mark has propagated through B's incoming buffer, B returns the WILL TIMING-MARK over the TELNET connection to A.

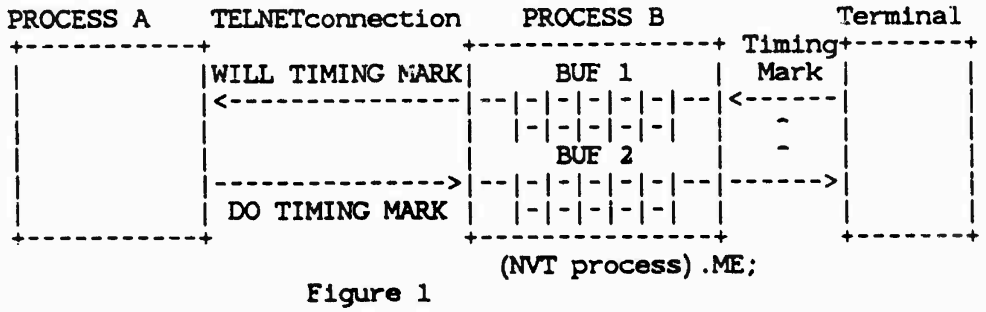


Figure 1

When A receives the WILL TIMING-MARK, he knows that all the information he sent to B before sending the timing mark been delivered, and all the information sent from B to A before turnaround of the timing mark has been delivered.

Three typical applications are:

A. Measure round-trip delay between a process and a terminal or another process.

B. Resynchronizing an interaction as described in section 4 above. A is a process interpreting commands forwarded from a terminal by B. When A sees an illegal command it:

   i. Sends <carriage return>, <line feed>, <question mark>.

   ii. Sends DO TIMING-MARK.

   iii. Sends an error message.

   iv. Starts reading input and throwing it away until it receives a WILL TIMING-MARK.

   v. Resumes interpretation of input.

This achieves the effect of flushing all "type ahead" after the erroneous command, up to the point when the user actually saw the question mark.

C. The dual of B above. The terminal user wants to throw away unwanted output from A.

i. B sends DO TIMING-MARK, followed by some new command.

ii. B starts reading output from A and throwing it away until it receives WILL TIMING-MARK.

iii. B resumes forwarding A's output to the terminal.

This achieves the effect of flushing all output from A, up to the point where A saw the timing mark, but not output generated in response to the following command.

It is sometimes useful for a user or process at one end of a TELNET connection to be sure that previously transmitted data has been completely processed, printed, discarded, or otherwise disposed of. This option provides a mechanism for doing this. In addition, even if the option request (DO TIMING-MARK) is refused (by WON'T TIMING-MARK) the requester is at least assured that the refuser has received (if not processed) all previous data.

As an example of a particular application, imagine a TELNET connection between a physically full duplex terminal and a "full duplex" server system which permits the user to "type ahead" while the server is processing previous user input. Suppose that both sides have agreed to Suppress Go Ahead and that the server has agreed to provide echoes. The server now discovers a command which it cannot parse, perhaps because of a user typing error. It would like to throw away all of the user's "type-ahead" (since failure of the parsing of one command is likely to lead to incorrect results if subsequent commands are executed), send the user an error message, and resume interpretation of commands which the user typed after seeing the error message. If the user were local, the system would be able to discard the buffered input; but input may be buffered in the user's host or elsewhere. Therefore, the server might send a DO TIMING-MARK and hope to receive a WILL TIMING-MARK from the user at the "appropriate place" in the data stream.

The "appropriate place", therefore (in absence of other information) is clearly just before the first character which the user typed after seeing the error message. That is, it should appear that the timing mark was "printed" on the user's terminal and that, in response, the user typed an answering timing mark.

Next, suppose that the user in the example above realized that he had misspelled a command, realized that the server would send a DO TIMING-MARK, and wanted to start "typing ahead" again without waiting for this to occur. He might then instruct his own system to send a WILL TIMING-MARK to the server and then begin "typing ahead" again. (Implementers should remember that the user's own system must remember that it sent the WILL TIMING-MARK so as to discard the DO/DON'T TIMING-MARK when it eventually arrives.) Thus, in this case the "appropriate place" for the insertion of the WILL TIMING-MARK is the place defined by the user.

It should be noted, in both of the examples above, that it is the responsibility of the system which transmits the DO TIMING-MARK to discard any unwanted characters; the WILL TIMING-MARK only provides help in deciding which characters are "unwanted".

5. Description of the Option

Network Working Group
Request for Comments: 860

J. Postel
J. Reynolds
ISI
May 1983

Obsoletes: NIC 16238

TELNET TIMING MARK OPTION

This RFC specifies a standard for the ARPA community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

1. Command Name and Code

TIMING-MARK 6

2. Command Meanings

IAC DO TIMING-MARK

The sender of this command REQUESTS that the receiver of this command return a WILL TIMING-MARK in the data stream at the "appropriate place" as defined in section 4 below.

IAC WILL TIMING-MARK

The sender of this command ASSURES the receiver of this command that it is inserted in the data stream at the "appropriate place" to insure synchronization with a DO TIMING-MARK transmitted by the receiver of this command.

IAC WON'T TIMING-MARK

The sender of this command REFUSES to insure that this command is inserted in the data stream at the "appropriate place" to insure synchronization.

IAC DON'T TIMING-MARK

The sender of this command notifies the receiver of this command that a WILL TIMING-MARK (previously transmitted by the receiver of this command) has been IGNORED.

3. Default

WON'T TIMING-MARK, DON'T TIMING-MARK

i.e., No explicit attempt is made to synchronize the activities at the two ends of the TELNET connection.

4. Motivation for the Option

NWG/RFC# 726 JBP DHC 8-MAR-77 08:29 39237
Remote Controlled Transmission & Echoing Telnet Option

Network Working Group Jon Postel & Dave Crocker
Request for Comments: 726 SRI-ARC UC Irvine
NIC: 39237 8 March 1977

Remote Controlled Transmssion and Echoing Telnet Option

1

1. Command name and code: 2

RCTE 7 2a

2. Command meanings: 3

IAC WILL RCTE 3a

The sender of this command REQUESTS or AGREES to use the RCTE option, and will send instructions for controlling the other side's terminal printer. 3a1

IAC WON'T RCTE 3b

The sender of this option REFUSES to send instructions for controlling the other side's terminal printer. 3b1

IAC DO RCTE 3c

The sender REQUEST or AGREES to have the other side (sender of WILL RCTE) issue commands which will control his (sender of the DO) output to the terminal printer. 3c1

IAC DON'T RCTE 3d

The sender of this command REFUSES to allow the other side to control his (sender of DON'T) terminal printer. 3d1

IAC SB RCTE <cmd> [BC1 BC2] [TC1 TC2] IAC SE 3e

where: 3e1

<cmd> is one 8-bit byte having the following flags (bits are counted from the right): 3e1a

NWG/RFC# 726 JBP DHC 8-MAR-77 08:29 39237
Remote Controlled Transmission & Echoing Telnet Option

Bit Meaning 3elb

0 0 = Ignore all other bits in this byte and repeat the last <cmd> that was sent. Equals a 'continue what you have been doing'.
1 = Perform actions as indicated by other bits in this byte. 3elc

1 0 = Print (echo) break character
1 = Skip (don't echo) break character 3eld

2 0 = Print (echo) text up to break character
1 = Skip (don't echo) text up to break character 3ele

3 0 = Continue using same classes of break characters.
1 = The two 8-bit bytes following this byte contain flags for the new break classes. 3elf

4 0 = Continue using same classes of transmit characters.
1 = Reset transmit classes according to the two bytes following 1) the break classes bytes, if the break classes are also being reset, or 2) this byte, if the break classes are NOT also being reset. 3elg

Value (decimal) of the <cmd> byte and its meaning: 3elh

0 = Continue what you have been doing 3eli

Even numbers greater than zero (i.e. numbers with the right most bit off) are in error and should be interpreted as equal to zero. When the <cmd> is an even number greater than zero, classes bytes TC1 & TC2 and/or BC1 & BC2 must not be sent. 3elj

1 = Print (echo) up to AND INCLUDING break character 3elk

3 = Print up to break character and SKIP (don't echo) break character 3ell

5 = Skip text (don't echo) up to break character, but PRINT break character 3elm

7 = Skip up to and including break character 3eln

Add one of the previous non-zero values to one of the following values, to get the total decimal value for

the byte (Note that classes may not be reset without also resetting the printing action; so an odd number is guaranteed): 3e1o

8 = Set break classes (using the next two bytes [BC1 BC2]) 3e1p

16 = Set transmission classes (using the next two bytes [TC1 TC2]) 3e1q

24 = Set break classes (using the next two bytes [BC1 BC2]) and the transmission classes (using the two bytes after that [TC1 TC2]). 3e1r

Sub-commands (IAC SB RCTE...) are only sent by the controlling host and, in addition to other functions, functionally replace the Go-Ahead (IAC GA) Telnet feature. RCTE also functionally replaces the Echo (IAC ECHO) Telnet option. That is the Suppress Go-Ahead option should be in force and the Echo option should not be in force while the RCTE option is in use. The echo mode on terminating use of the RCTE option should be the default state, that is DON'T ECHO, WON'T ECHO. 3e2

Classes for break and transmission (the right-most bit of the second byte (TC2 or BC2) represents class 1; the left-most bit of the first byte (TC1 or BC1) represents the currently undefined class 16: 3e3

1: Upper-Case Letter (A-Z) 3e3a

2: Lower-case Letters (a-z) 3e3b

3: Numbers (0-9) 3e3c

4: Format Effectors (<BS> <CR> <LF> <FF> <HT> <VT>) 3e3d

The sequence <cr><lf> counts as one character when processed as the Telnet end of line, and is a single break character when class 4 is set. The sequence <cr><nul> counts as one character and is a break character if and only if <cr> is a break character (i.e. class 4 is set).

5: Non-format Effector Control Characters including <DEL> and <ESC> 3e3e

6: . , ; : ? ! 3e3f

NWG/RFC# 726 JBP DHC 8-MAR-77 08:29 39237
Remote Controlled Transmission & Echoing Telnet Option

7: { [(< >)] } 3e3g

8: ' " / \ % @ $ & # + - * = ^ _ | ~ 3e3h

9: <Space> 3e3i

And Telnet commands (IAC . . .) sent by the user are always to have the effect of a break character. That is, every instance of an IAC is to be treated as a break character, except the sequence IAC IAC. 3e3j

The representation to be displayed when printing is called for is the obvious one for the visible characters (classes 1, 2, 3, 6, 7, and 8). Space (class 9) is represented by a blank space. The format effectors (class 4) by their format effect. The non-format effector controls (class 5) print nothing (no space). 3e4

Initially no break classes or transmission classes are in effect. 3e5

Please note that if all the bits are set in a Telnet subcommand argument byte such as TC2 or BC2 then that byte must be preceeded by an <IAC> flag byte. This is the common convention of doubling the escape character to use its value as data. 3e6

Sub-commands (IAC SB RCTE...) are refered to as "break reset commands". 3e7

3. Default: 4

WON'T RCTE -- DON'T RCTE 4a

Neither host asserts special control over the other host's terminal printer. 4a1

4. Motivation for the option: 5

RFC's 1, 5 and 51 discuss Network and process efficiency and smoothness. 5a

RFC 357, by John Davidson, introduces the problem of echoing delay that occurs when a remote user accesses a full-duplex host, thru a satellite link. In order to save the many thousands of miles of transit time for each echoed character, while still permitting full server responsiveness and clean terminal output, an echo control

similar to that used by some time-sharing systems is suggested for the entire Network. 5b

In effect, the option described in this document involves making a using host carefully regulate the local terminal printer according to explicit instructions from the remote (serving) host. 5b1

An important additional issue is efficient Network transmission. Implementation of the Davidson Echoing Scheme will eliminate almost all server-to-user echoing. 5c

The option described in this document also requests using hosts to buffer a terminal's input to the serving host until it forms a useful unit (with "useful unit" delimited by break or transmission characters as described below). Therefore, fewer messages are sent on the user-to-server path. 5c1

N.B.: This option is only intended for use with full-duplex hosts. The Go-Ahead Telnet feature is completely adequate for half-duplex server hosts. Also, RCTE should be used in place of the ECHO Telnet option. That is the Suppress Go-Ahead option should be in force and the Echo option should not be in force while the RCTE option is in use. 5d

NWG/RFC# 726 JBP DHC 8-MAR-77 08:29 39237
Remote Controlled Transmission & Echoing Telnet Option

5. Explicit description of control mechanism: 6

User Terminal Printing Action & Control Procedure 6a

Negotiate the use of the RCTE option. Once the option is in force the user Telnet follows the following procedure. 6a1

1) Read an item from the network. 6a2

If the item is data, then print it and go to 1. 6a2a

If the item is a command, then set the classes and go to 2. 6a2b

2) If the terminal input buffer is empty, then go to 3, else go to 4. 6a3

3) Wait for an item to appear either from the terminal or from the network. 6a4

If an item appears from the terminal, then go to 4. 6a4a

If a data item appears from the network, then print it and go to 3. 6a4b

If a command appears from the network, then an error has occured. 6a4c

4) Read an item from the terminal input buffer. 6a5

If the item is not a break, then print/skip it and go to 2. 6a5a

If the item is a break, then print/skip it and go to 1. 6a5b

Note: Output from the server host may occur at any time, such "spontaneous output" is printed in step 3. 6a6

NWG/RFC# 726 JBP DHC 8-MAR-77 08:29 39237
Remote Controlled Transmission & Echoing Telnet Option

Explanation: 6b

Both Hosts agree to use the RCTE option. After that, the using host (IAC DO RCTE) merely acts upon the controlling (serving) host's commands and does not issue any RCTE commands unless and until it (using host) decides to stop allowing use of the option (by sending IAC DON'T RCTE). 6b1

1) The using host is synchronized with the server by initially and when ever it returns to step 1 suspending terminal echo printing until it receives a command from the server. 6b2

The server may send either output to the terminal printer or a command, and usually sends a both. 6b3

The server may send output to the terminal printer either in response to user input or spontaneously. In the former case, the output is processed in step 1. In the latter case, the output is processed in step 3. 6b4

Server sends an RCTE command. The command may redefine break and transmission classes, action to be performed on break characters, and action to be performed on text. Each of these independent functions is controlled by separate bits in the <cmd> byte. 6b5

A transmission character is one which RECOMMENDS that the using host transmit all text accumulated up to and including its occurrence. (For network efficiency, using hosts are DISCOURAGED (but not prohibited) from sending before the occurrence of a transmission character, as defined at the moment the character is typed). 6b5a

If the transmission classes bit (bit 4) is on, the two bytes following the two break classes bytes (or immediately following the <cmd> byte, if the break classes bit is not on) will indicate what classes are to be enabled.

If the bit is OFF, the transmission classes remain unchanged. When the RCTE option is first initiated, NO CLASSES are in effect. That is, no character will be considered a transmission character. (As if both TC1 and TC2 are zero.)

A break character REQUIRES that the using host

transmit all text accumulated up to and including its occurrence and also causes the using host to stop its print/discard action upon the user's input text, until directed to do otherwise by another IAC SB RCTE <cmd> IAC SE command from the serving host. Break characters therefore define printing units. "Break character" as used in this document does NOT mean Telnet Break character. 6b5b

If the break classes bit (bit 3) is on, the two bytes following <cmd> will indicate what classes are to be enabled. There are currently nine (9) classes defined, with room for expansion.

If the bit is OFF, the break classes remain unchanged. When the RCTE option is initiated, NO CLASSES are to be in effect. That is, no transmission will take place in the user to server direction until the first break reset command is received by the user from the server.

The list of character classes, used to define break and transmission classes are listed at the end of this document, in the Tables Section. 6b5c

Because break characters are special, the print/discard action that should be performed upon them is not always the same as should be performed upon the rest of the input text. 6b5d

For example, while typing a filename to TENEX, I want the text of the filename to be printed (echoed); but I do not want the <escape> (if I use the name completion feature) to be printed.

If bit 1 is ON the break character is NOT to be printed.

A separate bit (bit 2) signals whether or not the text itself should be printed (echoed) to the terminal. If bit 2 = 0, then the text IS to be printed. 6b5e

Yet another bit (bit 0 - right-most bit) signals whether or not any of the other bits of the command should be checked. If this bit is OFF, then the command should be interpreted to mean "continue whatever echoing strategy you have been following, using the same break and transmission classes." 6b5f

2) The user Telnet now checks the terminal input buffer, if it contains data it is processed in step 4, otherwise the user Telnet waits in step 3 for further developments. 6b6

3) The user Telnet waits until either the human user enters some data in which case Telnet proceeeds to step 4, or an item is received from the network. If the item from the network is data it is spontaneous output and is printed, Telnet then continues to wait. If the item from the network is a command then an error has occured. In this case the user Telnet may attempt to resynchronize the use of RCTE as indicated below. 6b7

4) Items from the terminal are processed with printing controlled by the settings of the latest break reset command. When a break character is processed, the cycle of control is complete and action re-commences at step 1. 6b8

Input from the terminal is (hopefully) buffered into units ending with a transmission or break character; and echoing of input text is suspended after the occurrence of a break character and until receipt of a break reset command from the serving host. The most recent break reset command determines the break actions. 6b9

In summary, what is required is that for every break character sent in the user to server direction there be a break reset command sent in the server to user direction. The user host initially has no knowledge of which characters are break characters and so starts in a state that assumes that there are no break characters and also that no echoing is to be provided. The server host is expected to send a break reset command to establish the break classes and the echoing mode before it receives any data from the user. 6b10

Synchronization and Resynchronization: 6c

The serving and using hosts must carefully synchronize break reset commands with the transmission of break characters. Except at the beginning of an interaction, the serving host may only send a break reset command in response to the Using host's having sent a break character as defined at that time. This should establish a one-to-one correspondence between them. (A <cmd> value of zero, in this context, is interpreted as

a break classes reset to the same class(es) as before.) The break reset command may be preceded by terminal output. 6c1

The re-synchronization of the break characters and the break reset commands is done via the exchange of the Telnet signal Abort Output (AO) in the server to user direction and the SYNCH in the user to server direction. 6c2

Suppose the server wants to resynchronize the break characters and the break reset commands. 6c3

a. The server should be sure all output to the terminal has been printed by using, for example, the Timing Mark Option. 6c3a

b. The server sends the AO signal. 6c3b

c. The user receives the AO signal. The user flushes all user to server data wheather it has been echoed or not. The user sends a SYNCH to the server. [The SYNCH consists of the Telnet Data Mark (DM) and the host-to-host interrupt (INS).] The user now enters the initial state at step 1. 6c3c

d. The server receives the SYNCH and flushes any data preceeding the DM (as always). The server now sends a break reset command. (Actually the break reset command could be sent at any time following the AO.) 6c3d

Suppose the user wants to resynchronize the break characters and the break reset commands. 6c4

a. The user should discard all user to server data wheather it has been echoed or not. 6c4a

b. The user sends the AO signal. The user now enters the algorithm at step 1. 6c4b

c. The server receives the AO signal. The server discards all data buffered but not yet sent to the user. The server sends a SYNCH to the user. The server sends a break reset command to the user. 6c4c

NWG/RFC# 726 JBP DHC 8-MAR-77 08:29 39237
Remote Controlled Transmission & Echoing Telnet Option

Notes and Comments: 6d

Even-numbered commands, greater than zero, are in error, since they will have the low-order bit off. The command should be interpreted as equal to zero, which means that any classes reset bytes ([TC1 TC2] [BC1 BC2]) will be in error. (The IAC SE, at the end of the command, eliminates any parsing problems due to this error.) 6d1

Serving hosts will generally instruct using hosts not to echo break characters, even though it might be alright to echo most break characters. For example, <cr> is usually a safe character to echo but <esc> is not. TENEX Exec is willing to accept either, during filename specification. Therefore, the using host must be instructed not to echo any break characters. 6d2

This is generally a tolerable problem, since the serving host has to send an RCTE command at this point, anyhow. Adding an echo for the break character to the message will not cause any extra network traffic. 6d2a

The RCTE Option entails a rather large overhead. In a true character-at-a-time situation, this overhead is not justified. But on the average, it should result in significant savings, both in network traffic and host wake-ups. 6d3

Buffering Problems and Transmission vs. Printing Constraints: 6d4

There are NO mandatory transmission constraints. The using host is allowed to send a character a time, though this would be a waste of RCTE. The transmission classes commands are GUIDELINES, so deviating from them, as when the user's buffer gets full, is allowed. 6d4a

Additionally, the using host may send a break class character, without knowing that it is one (as with type-ahead). 6d4b

If the user implementation is clever it may send the user entered data to the server before it is actually needed. This type ahead data may contain break characters.

Assume that only space is a break character (that is the last break reset command specified print up to and including the break characters and set the break classes to class 9). Suppose the user had typed "abc<space>def<esc>ghi<cr>". The user side RCTE could send it all to the server, but it could print only "abc<space>", and would have to buffer "def<esc>ghi<cr>" at least until a break reset command was received from the server. That break reset command could change the break ckasses requiring rescanning the buffered string.

For example suppose the break reset command set the break characters to class 5 and the action to print up to but not including the break character. The user RCTE could then print "def" and discard the <esc>, but whould have to continue to buffer the "ghi<cr>".

The problem with buffering occurs when printing on the user's terminal must be suspended, after the user has typed a currently valid break character and until a break reset command is received from the serving host. During this time, the user may be typing merrily along. The text being typed may be SENT, but may not yet be PRINTED. 6d4c

The more common problem of filling the transmission buffer, while awaiting a host to host allocate from the serving host, may also occur, but this problem is well known to implementors and in no way special to RCTE. 6d4d

In any case, when the buffer does fill and further text typed by the user will be lost, the user should be notified (perhaps by ringing the terminal bell). 6d4e

Text should be buffered by the using host until the user types a character which belongs to the transmission class in force at the moment the character is typed. 6d5

Transmission class reset commands may be sent by the serving host at any time. If they are frequently sent separate from break class reset commands, it will probably be better to exit from RCTE and enter regular character at a time transmission. 6d6

It is not immediately clear what the using host should

do with currently buffered text, when a transmission classes reset command is received. The buffering is according to the previous transmission classes scheme. 6d7

The using host clearly should not simply wait until a transmission character (according to the new scheme) is typed. 6d7a

Either the buffered text should be rescanned, under the new scheme; 6d7b

Or the buffered text should simply be sent as a group. This is the simpler approach, and probably quite adequate. 6d7c

It is possible to define NO BREAK CHARACTERS except Telnet commands (IAC ...). This seems undesirable and should not be done. 6d8

If this situation were to occur the using host should send a Telnet command to allow the server to know when he may reset the break classes, but the mechanism is awkward and this case should be avoided. 6d8a

6. Sample Interaction: 7

"S:" is sent from serving (WILL RCTE) host to using host.
"U:" is sent from using (DO RCTE) host to serving host.
"T:" is entered by the terminal user.
"P:" is printed on the terminal.

Text surrounded by square brackets ([]) is commentary. Text surrounded by angle brackets (<>) is to be taken as a single unit. E.g., carriage return is <cr>, and the decimal value 27 is represented <27>. 7a

The following interaction shows a logon to a Tenex, initiation of the DED editor, insertion of some text and the return to the Exec level. 7b

An attempt has been made to give some flavor of the asynchrony of network I/O and the user's terminal input. Many other possible combinations, using the same set of actions listed below, could be devised. The actual order of events will depend upon network and hosts' load and the user's typing speed. 7b1

We assume that the user's Telnet is also in an "insert linefeed" mode. That is, whenever the user types carriage

return <cr> the user Telnet sends both carriage return and linefeed <cr><lf> (the Telnet end of line signal). When space character occurs at the end of a line in the example description it is shown explicitly by <sp> to avoid confusion. Other uses of the space character are not so marked to avoid destroying the readability of the example. 7c

A Telnet connection has already been opened, but the TENEX prompt has not yet been issued. The hosts first discuss using the RCTE option: 7d

S: <IAC><WILL><RCTE> 7d1

U: <IAC><DO><RCTE> 7d2

S: TENEX 1.31.18, TENEX EXEC 1.50.2<cr><lf>@
<IAC><SB><RCTE><11><1><24><IAC><SE> 7d3

[Print the herald and echo input text up to a break character, but do not echo the break character. Classes 4 (Format Effectors), 5 (Non-format Effector Controls and <DEL>), and 9 (<sp>) act as break characters.] 7d3a

P: TENEX 1.31.18, TENEX EXEC 1.50.2<cr><lf>@ 7d4

T: LOGIN ARPA<cr> 7d5

P: LOGIN 7d6

U: LOGIN<sp> 7d7

U: ARPA<cr><lf> 7d8

S: <sp><IAC><SB><RCTE><0><IAC>SE> 7d9

P: <sp>ARPA 7d10

S: <cr><lf>(PASSWORD): <IAC><SB><RCTE><7><IAC><SE> 7d11

P: <cr><lf>(PASSWORD):<sp> 7d12

T: WASHINGTON 1000<cr> 7d13

[The password "WASHINGTON" is not echoed. Printing of "1000<cr>" is withheld] 7d13a

U: WASHINGTON<sp> 7d14

NWG/RFC# 726 JBP DHC 8-MAR-77 08:29 39237
Remote Controlled Transmission & Echoing Telnet Option

1000<cr><lf> 7d15

S: <sp><IAC><SB><RCTE><3><IAC><SE> 7d16

S: <cr><lf>JOB 17 ON TTY41 7-JUN-73 14:13<cr><lf>@
<IAC><SB><RCTE><0><IAC><SE> 7d17

P: <sp>1000 7d18

[Printing is slow at this point; so the account number is not printed as soon as the server's command for it is received.] 7d18a

P: <cr><lf>JOB 17 ON TTY41 7-JUN-73 14:13<cr><lf>@ 7d19

T: DED<esc><cr> 7d20

P: DED 7d21

U: DED<esc> 7d22

S: .SAV;1<IAC><SB><RCTE><0><IAC><SE> 7d23

P: .SAV;1 7d24

U: <cr><lf> 7d25

S: <cr><lf><lf>DED 3/14/73 DRO,KRK<cr><lf>:
<IAC><SB><RCTE><15><1><IAC><255><IAC><SE> 7d26

[The program is started and the DED prompt ":" is sent. At the command level, DED responds to every character. The server sets the break classes to all classes.] 7d26a

P: <cr><lf><lf>DED 3/14/73 DRO,KRK<cr><lf>: 7d27

T: IThis is a test line.<cr>This is another test line.<^Z>Q 7d28

["I" means Insert Text. The text follows, terminated by a Control-Z. The "Q" instructs DED to Quit.] 7d28a

U: I 7d29

U: This is a test line.<cr><lf> 7d30

S: I<cr><lf>*<IAC><SB><RCTE><11><0><24><IAC><SE> 7d31

NWG/RFC# 726 JBP DHC 8-MAR-77 08:29 39237
Remote Controlled Transmission & Echoing Telnet Option

[DED prompts the user, during text input, with an asterisk at the beginning of every line. The server sets the break classes to classes 4 and 5, the format effectors and the non-format effector controls.] 7d31a

P: I<cr><lf>*This is a test line. 7d32

S: <cr><lf>*<IAC><SB><RCTE><0><IAC><SE> 7d33

P: <cr><lf>*This is another test line. 7d34

U: This is another test line.<^Z> 7d35

U: Q 7d36

[Note that the "Q" will not immediately be printed on the terminal, since it must wait for authorization.] 7d36a

S: ^Z<cr><lf>:<IAC><SB><RCTE><15><1><IAC><255><IAC><SE> 7d37

[The returned "^Z" is two characters, not the ASCII Control-Z or <sub>.] 7d37a

S: Q<cr><lf>@<IAC><SB><RCTE><11><1><24><IAC><SE> 7d38

P: Q<cr><lf>@ 7d39

And the user is returned to the Exec level. 7d40

TELNET Output Line Width Option

NIC 20196 (Nov. 13, 1973)

TELNET OUTPUT LINE WIDTH OPTION

1. Command name and code.

   NAOL 8 (Negotiate About Output Line-width)

2. Command meanings

   In the following, we are discussing a simplex connection, one half of a full duplex TELNET connection. On the simplex connection under discussion, by definition data passes from the data sender to the data receiver. If we consider the example of a computer transmitting data over a connection to a terminal where the data is printed, then the computer is the data sender and the terminal is the data receiver. Continuing to use this example, the NAOL option could be used to negotiate the line width to be used when printing lines from the computer on the terminal. To negotiate line width on the other half of the TELNET connection the parties involved reverse their data sender and data receiver roles; this can be done unambiguously as the sender of a DO or DON'T NAOL command can only be the data sender, thus defining the half of the TELNET connection under discussion, and the sender of a WILL or WON'T NAOL command can only be the data receiver.

   IAC DO NAOL

   The data sender requests or agrees to negotiate about output line width with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output line width considerations.

   IAC DON'T NAOL

   The data sender refused to negotiate about output line width with the data receiver, or demands a return to the unnegotiated default mode.

   IAC WILL NAOL

   The data receiver requests or agrees to negotiate about output line width with the data sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output line width considerations.

IAC WON'T NAOL

The data receiver refuses to negotiate about output line width, or demands a return to the unnegotiated default mode.

IAC SB NAOL DS <8 bit value> IAC SE

The data sender specifies, with the 8 bit value, which party should handle output line width considerations and how. The code for DS is 1.

IAC SB NAOL DR <8 bit value> IAC SE

The data receiver specifies, with the 8 bit value, which party should handle output line width considerations and how. The code for DR is 0.

3. Default

DON'T NAOL

In the default absence of

WON'T NAOL

negotiation concerning which party, data sender or data receiver, is handling output line width considerations, neither party is required to handle line width consideration and neither party is prohibited from handling line width consideration but it is appropriate if at least the data receiver handles line width considerations albeit primitively.

4. Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about the output line width:

a) The sender may wish the receiver to use its local knowledge of the printer width to properly handle the line width;

b) The receiver may wish the sender to use its local knowledge of the data being sent to properly handle the line width;

c) The sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of the line width; and

d) The receiver may wish to use its local knowledge of the printer width to instruct the sender in the proper handling of the line width.

An example of proper handling of the line length is for the receiver to "fold" lines sent by the sender so that the lines fit on the printer page. Another example of proper handling of the line length might be notfolding lines even though they overflow the printer page, as that is whatthe user desires.

5. Description of the Option

The data sender and data receiver use the 8 bit value along with the DS and DR SB commands as follows:

8 bit value	Meaning
0	command sender suggests he alone will handle output line-width considerations for the connection.
1 to 253	command sender suggests other party alone should handle output line-width considerations but suggests line width should be value given, in characters.
254	command sender suggests other party alone should handle output line-width considerations but suggests line width should be considered infinity.
255	command sender suggests other party alone should handle output line-width considerations and suggests nothing about how it should be done.

The guiding rules are that

1) if neither data receiver or data sender wants to handle output line-width considerations, the data receiver must do it, and

2) if both data receiver or data sender want to handle output line-width considerations, the data sender gets to do it.

The reasoning for the former rule is that if neither want to do it, then the default in the NAOL option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver makes.

6. Some sample negotiations are:

no subnegotiations	data receiver handles output line-width considerations
IAC SB NAOL DS 132 IAC SE IAC SB NAOL DR 0 IAC SE	data sender suggests data receiver handle output line-width consideration data with suggested line width of 132; receiver agrees.
IAC SB NAOL DR 255 IAC SE IAC SB NAOL DS 0 IAC SE	data receiver suggests data sender handle line-width considerations; sender refuses.
IAC SB NAOL DS 0 IAC SE IAC SB NAOL DR 72 IAC SE	data sender wants to handle line-width considerations; receiver agrees but notifies the sender the line printer only has 72 columns.

As with all option negotiation, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time either party can disable further negotiation by giving the appropriate WON'T NAOL or DON'T NAOL command.

TELNET Output Page Size Option

NIC 20197 (Nov. 13, 1973)

1. Command name and code.

NAOP 9 (Negotiate About Output Page-size)

(By page size we mean number of lines per page.)

2. Command meanings.

In the following, we are discussing a simplex connection, one half of a full duplex TELNET connection. On the simplex connection under discussion, by definition data passes fromthe data sender to the data receiver. If we consider the example of a computer transmitting data over a connection to a terminal where the data is printed, then the computer is the data sender and the the terminal is the data receiver. Continuing to use this example, the NAOP option could be used to negotiate the page size to be used when printing pages from the computer on the terminal. To negotiate page size on the other half of the TELNET connection the parties involved reverse their data sender and data receiver roles; this can be done unambiguously as the sender of a DO or DON'T NAOP command can only be the data sender, thus defining the half of the TELNET connection under discussion, and the sender of a WILL or WON'T NAOP command can only be the data receiver.

IAC DO NAOP	The data sender requests or agrees to negotiate about output page size with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output page size considerations.
IAC DON'T NAOP	The data sender refuses to negotiate about output page size with the data receiver, or demands a return to the unnegotiated default mode.
IAC WILL NAOP	The data receiver requests or agrees to negotiate about output page size with the data sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output page size considerations.

IAC WON'T NAOP — The data receiver refuses to negotiate about output page size, or demands a return to the unnegotiated default mode.

IAC SB NAOP DS <8 bit value> IAC SE — The data sender specifies, with the 8 bit value, which party should handle output page size considerations and how. The code for DS is 1.

IAC SB NAOP DR <8 bit value> IAC SE — The data receiver specifies, with the 8 bit value, which party should handle output page size considerations and how. The code for DR is 0.

3. Default

DON'T NAOP
WON'T NAOP — In the default absence of negotiation concerning which party, data sender or data receiver, is handling output page size considerations, neither party is required to handle page size consideration and neither party is prohibited from handling page size consideration, but it is appropriate if at least the data receiver handles page size considerations albeit primitively.

4. Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about the output page size:

(a) the sender may wish the receiver to use its local knowledge of the printer page size to properly handle the page size;

(b) the receiver may wish the sender to use its local knowledge to the data being sent to properly handle the page size;

(c) the sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of the page size; and

(d) the receiver may wish to use its local knowledge of the printer size to instruct the sender in the proper handling of the page size.

An example of proper handling of the page size is for the receiver to hold off further output until instructed to continue when the lines being printed are about to overflow the scope face.

5. Description of the Option.

The data sender and data receiver use the 8 bit value along with the DS and DR SB commands as follows.

8 bit value	Meaning
0	command sender suggests he alone will handle output page size considerations for the connection.
1 to 253	command sender suggests other party alone should handle output page-size considerations but suggests page size should be value given, in lines.
254	command sender suggests other party alone should handle output page size considerations but suggests page size should be considered infinity.
255	command sender suggests other party alone should handle output page size considerations and suggests nothing about how it should be done.

The guiding rules are that

(1) if neither data receiver or data sender wants to handle output page size considerations, the data receiver must do it, and

(2) if both data receiver or data sender want to handle output page size, the data sender gets to do it.

The reasoning for the former rule is that if neither want to do it, then the default in the NAOP option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver makes.

Some sample negotiations are:

no subnegotiations	data receiver handles output page size considerations
IAC SB NAOP DS 66 IAC SE IAC SB NAOP DR 0 IAC SE	data sender suggests data receiver handle output page size consideration data with suggested page size of 66 lines; receiver agrees.
IAC SB NAOP DR 255 IAC SE IAC SB NAOP DS 0 IAC SE	data receiver suggests data sender handle page size condiseration; sender refuses.
IAC SB NAOP DS 0 IAC SE IAC SB NAOP DR 30 IAC SE	data sender wants to handle page size considerations; receiver agrees but notifies the sender the scope only has 30 IAC SE lines.

As with all option negotiation, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time either party can disable further negotiation by giving the appropriate WON'T NAOP or DON'T NAOP command.

Request for Comments: 652 D. Crocker (UCLA-NMC)
25 Oct. 74

NIC #31155
Online file: [ISI]<DCROCKER>NAOCRD.TXT

Telnet Output Carriage-Return Disposition Option

1. Command name and code

NAOCRD 10 (Negotiate About Output Carriage-Return Disposition)

2. Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet options.

IAC DO NAOCRD — The data sender requests or agrees to negotiate about output carriage-return character disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output carriage-returns.

IAC DON'T NAOCRD — The data sender refuses to negotiate about output carriage-return disposition with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOCRD — The data receiver requests or agrees to negotiate about output carriage-return disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output carriage-returns.

IAC WON'T NAOCRD — The data receiver refuses to negotiate about output carriage-return disposition, or demands a return to the unnegotiated default mode.

IAC SB NAOCRD DS <8-bit value> IAC SE
The data sender specifies, with the 8-bit value, which party should handle carriage-returns and what their disposition should be. The code for DS is 1

IAC SB NAOCRD DR <8-bit value> IAC SE The data receiver specifies, with the 8-bit value, which party should handle carriage-returns and what their disposition should be. The code for DR is 0.

3. Default

DON'T NAOCRD/WON'T NAOCRD. In the default absence of negotiations concerning which party, data sender or data receiver, is handling output carriage-returns, neither party is required to handle carriage-returns and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles carriage-returns, albeit primitively.

4. Motivation for the Option

Please refer to section 4 of the NAOL and of the NAOP Telnet option descriptions.

5. Description of the Option

The data sender and the data receiver use the 8-bit value along with the NAOCRD SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle carriage-returns, for the connection.
1 to 250	Command sender suggests that the other party alone should handle carriage-returns, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character. (See qualification, below.)
251	Not allowed, in orde. to be compatible with related Telnet options.
252	Command sender suggests that the other party alone handle carriage-returns, but suggests that they be discarded.
253	Not allowed, in order to be compatible with related Telnet options.

254 Command sender suggests that the other party alone should handle carriage-returns but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. (See qualification, below.) Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.

255 Command sender suggests that the other party alone should handle carriage-returns and suggests nothing about how it should be done.

The guiding rules are that:

(1) if neither data receiver nor data sender wants to handle carriage-returns, the data receiver must do it, and

(2) if both data receiver and data sender want to handle carriage-returns, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOCRD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make.

Note that carriage-return delays, controlled by the data sender, must consist of NUL characters inserted immediately after the character in question. This is necessary due to the assynchrony of network transmissions. Due to the Telnet end-of-line convention, with carriage-returns followed by a linefeed, any NULs that would otherwise be placed after the carriage-return must be placed after the linefeed, regardless of any modifications that may additionally be made to the line feed (see NAOLFD Telnet option).

As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time, either party can disable further negotiation by giving the appropriate WON'T NAOCRD or DON'T NAOCRD command.

TELNET OUTPUT HORIZONTAL TABSTOPS OPTION
RFC 653, NIC 31156 (Oct. 25, 1974)
D. Crocker (UCLA-NMC)
Online file: [ISI]<DCROCKER>NAOHTS.TXT

TELNET OUTPUT HORIZONTAL TABSTOPS OPTION

1. Command name and code
   NAOHTS 11 (Negotiate About Output Horizontal Tabstops)

2. Command meanings
   In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet options.

   IAC DO NAOHTS
   The data sender requests or agrees to negotiate about output horizontal tabstops with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output horizontal tabstops.

   IAC DON'T NAOHTS
   The data sender refuses to negotiate about output horizontal tabstops with the data receiver, or demands a return to the unnegotiated default mode.

   IAC WILL NAOHTS
   The data receiver requests or agrees to negotiate about output horizontal tabstops with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output horizontal tabstops.

   IAC WON'T NAOHTS
   The data receiver refuses to negotiate about output horizontal tabstops, or demands a return to the unnegotiated default mode.

   IAC SB NAOHTS DS <8-bit value> ... <8-bit value> IAC SE
   The data sender specifies, with the 8-bit value(s), which party should handle output horizontal tabstop considerations and what the stops should be. The code for DS is 1.

   IAC SB NAOHTS DR <8-bit value> ... <8-bit value> IAC SE
   The data receiver specifies, with the 8-bit value(s), which party should handle output horizontal tabstop considerations and what the stops should be. The code for DR is 0.

3. Default
   DON'T NAOHTS/WON'T NAOHTS.
   In the default absence of negotiations concerning which party, data sender or data receiver, is handling output horizontal tabstops, neither party is required to handle them and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles horizontal tabstops, albeit primitively.

4. Motivation for the Option
   Please refer to section 4 of the NAOL and of the NAOP Telnet option descriptions.

5. Description of the Option
The data sender and the data receiver use the 8-bit value(s) along with th DS and DR SB subcommands as follows (multiple 8-bit values are allowed onl if each is greater than zero and less than 251):

8-bit value :	Meaning :
0	Command sender suggests that he alone will handle tabstops, for the connection.
1 to 250	Command sender suggests that the other party alone should handle tabstop considerations, but suggests that the indicated value(s) be used. The value(s) are the column numbers, relative to the physical left side of the printer page or terminal screen, that are to be set.
251 to 254	Not allowed, in order to be compatible with related Telnet options.
255	Command sender suggests that the other party alone should handle output tabstops and suggests nothing about how it should be done.

The guiding rules are that:
(1) if neither data receiver nor data sender wants to handle output horizontal tabstops, the data receiver must do it, and
(2) if both data receiver and data sender want to handle output horizontal tabstops, the data sender gets to do it.
The reasoning for the former rule is that if neither wants to do it, then the default in the NAOHTS option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should b allowed to do it, taking into account any suggestions the receiver may mak As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).
At any time, either party can disable further negotiation by giving the appropriate WON'T NAOHTS or DON'T NAOHTS command.

TELNET OUTPUT HORIZONTAL TAB DISPOSITION OPTION
RFC 654, NIC 31157 (Oct. 25, 1974)
D. Crocker (UCLA-NMC)
Online file: [ISI]<DCROCKER>NAOHTD.TXT

TELNET OUTPUT HORIZONTAL TAB DISPOSITION OPTION

1. Command name and code
   NAOHTD 12
   (Negotiate About Output Horizontal Tab Disposition)

2. Command meanings
   In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet options.
   IAC DO NAOHTD
   The data sender requests or agrees to negotiate about output horizontal tab character disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output horizontal tab character considerations.
   IAC DON'T NAOHTD
   The data sender refuses to negotiate about output horizontal tab characters with the data receiver, or demands a return to the unnegotiated default mode.
   IAC WILL NAOHTD
   The data receiver requests or agrees to negotiate about output horizontal tab characters with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output horizontal tab character considerations.
   IAC WON'T NAOHTD
   The data receiver refuses to negotiate about output horizontal tab characters, or demands a return to the unnegotiated default mode.
   IAC SB NAOHTD DS <8-bit value> IAC SE
   The data sender specifies, with the 8-bit value, which party should handle output horizontal tab characters and what their disposition should be. The code for DS is 1.
   IAC SB NAOHTD DR <8-bit value> IAC SE
   The data receiver specifies, with the 8-bit value, which party should handle output horizontal tab characters and what their disposition should be. The code for DR is 0.

3. Default
   DON'T NAOHTD/WON'T NAOHTD.
   In the default absence of negotiations concerning which party, data sender or data receiver, is handling output horizontal tab character considerations, neither party is required to handle horizontal tab characters and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles horizontal tab character considerations, albeit primitively.

4. Motivation for the Option
   Please refer to section 4 of the NAOL and of the NAOP Telnet option descriptions.

5. Description of the Option
The data sender and the data receiver use the 8-bit value along with the DS and DR SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle horizontal tab characters, for the connection.
1 to 250	Command sender suggests that the other party alone should handle horizontal tab characters, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character.
251	Command sender suggests that the other party alone handle horizontal tabs, but suggests that each occurrence of the character be replaced by a space.
252	Command sender suggests that the other party alone handle horizontal tabs, but suggests that they be discarded.
253	Command sender suggests that the other party alone should handle horizontal tab characters, but suggests that tabbing be simulated.
254	Command sender suggests that the other party alone should handle horizontal tab characters, but suggests that waiting for a character to be transmitted (on the other simplex connection) before sending more data. Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
255	Command sender suggests that the other party alone should handle output horizontal tabs and suggests nothing about how it should be done.

The guiding rules are that:

1) if neither data receiver nor data sender wants to handle output horizontal tab characters, the data receiver must do it, and
2) if both data receiver and data sender wants to handle output horizontal tab characters, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOHTD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. Simulation is defined as the replacement of the horizontal tab character by enough spaces to move the printer head (or line-pointer) to the next horizontal tab stop
Note that delays, controlled by the data sender, must consist of NUL characters inserted immediately after the horizontal tab character. This is necessary due to the assynchrony of network transmissions. As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOHTD or DON'T NAOHTD command.

TELNET OUTPUT FORMFEED DISPOSITION OPTION
RFC 655, NIC 31158 (Oct. 25, 1974)
D. Crocker (UCLA-NMC)
Online file: [ISI]<DCROCKER>NAOFFD.TXT

TELNET OUTPUT FORMFEED DISPOSITION OPTION

1. Command name and code
   NAOFFD - 13
      (Negotiate About Output Formfeed Disposition)

2. Command meanings
   In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet Options specifications.
      IAC DO NAOFFD
         The data sender requests or agrees to negotiate about output formfeed disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output formfeeds.
      IAC DON'T NAOFFD
         The data sender refuses to negotiate about output formfeed disposition with the data receiver, or demands a return to the unnegotiated default mode.
      IAC WILL NAOFFD
         The data receiver requests or agrees to negotiate about output formfeed disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output formfeeds.
      IAC WON'T NAOFFD
         The data receiver refuses to negotiate about output formfeed disposition, or demands a return to the unnegotiated default mode.
      IAC SB NAOFFD DS <8-bit value> IAC SE
         The data sender specifies, with the 8-bit value, which party should handle formfeeds and what their disposition should be. The code for DS is 1.
      IAC SB NAOFFD DR <8-bit value> IAC SE
         The data receiver specifies, with the 8-bit value, which party should handle formfeeds and what their disposition should be. The code for DR is 0.

3. Default
   DON'T NAOFFD/WON'T NAOFFD
      In the default absence of negotiations concerning which party, data sender or data receiver, is handling output formfeeds, neither party is required to handle formfeeds and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles formfeed considerations, albeit primitively.

4. Motivation for the Option
   Please refer to section 4 of the NAOL and of the NAOFFD Telnet option descriptions.

5. Description of the Option
The data sender and the data receiver use the 8-bit value along with the DS and DR SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle formfeeds, for the connection.
1 to 250	Command sender suggests that the other party alone should handle formfeeds, but suggests that the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character.
251	Command sender suggests that the other party alone handle formfeeds, but suggests that each occurrence of the character be replaced by carriage-return/line-feed.
252	Command sender suggests that the other party alone handle formfeeds, but suggests that they be discarded.
253	Command sender suggests that the other party alone should handle formfeeds, but suggests that formfeeds be simulated.
254	Command sender suggests that the other party alone should handle output formfeeds but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
255	Command sender suggests that the other party alone should handle output formfeeds and suggests nothing about how it should be done.

The guiding rules are that:

1) if neither data receiver nor data sender wants to handle output formfeeds, the data receiver must do it, and
2) if both data receiver and data sender want to handle output formfeeds, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOFFD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. Simulation is defined as the replacement of the formfeed character by enough line-feeds (only) to advance the paper (or line-pointer) to the top of the next page (or to the top of the terminal screen). Note that delays, controlled by the data sender, must consist of NUL characters inserted immediately after the formfeed character. This is necessary due to the assynchrony of network transmission. As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOFFD or DON'T NAOFFD command.

TELNET OUTPUT VERTICAL TABSTOPS OPTION
RFC 656, NIC 31159 (Oct. 25, 1974)"
D. Crocker (UCLA-NMC)
Online file: [ISI]<DCROCKER>NAOVTS.TXT

TELNET OUTPUT VERTICAL TABSTOPS OPTION

1. Command name and code
   NAOVTS 14
      (Negotiate About Vertcial Tabstops)

2. Command meanings
   In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet Options specifications.
      IAC DO NAOVTS
         The data sender requests or agrees to negotiate about output vertical tabstops with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output vertical tabstop considerations.
      IAC DON'T NAOVTS
         The data sender refuses to negotiate about output vertical tabstops with the data receiver, or demands a return to the unnegotiated default mode.
      IAC WILL NAOVTS
         The data receiver requests or agrees to negotiate about output vertical tabstops with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output vertical tabstop considerations.
      IAC WON'T NAOVTS
         The data receiver refuses to negotiate about output vertical tabstops, or demands a return to the unnegotiated default mode.
      IAC SB NAOVTS DS <8-bit value> ... <8-bit value> IAC SE
         The data sender specifies, with the 8-bit value(s), which party should handle output vertical tabstop considerations and what the stops should be. The code for DS is 1.
      IAC SB NAOVTS DR <8-bit value> ... <8-bit value> IAC SE
         The data receiver specifies, with the 8-bit value(s), which party should handle output vertical tabstop considerations and what the stops should be. The code for DR is 0.

3. Default
   DON'T NAOVTS/WON'T NAOVTS.
      In the default absence of negotiations concerning which party, data sender or data receiver, is handling output vertical tabstop considerations, neither party is required to handle vertical tabstops and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles vertical tabstop considerations, albeit primitively.

4. Motivation for the Option
   Please refer to section 4 of the NAOL and of the NAOVTS Telnet option descriptions.

5. Description of the Option
The data sender and the data receiver use the 8-bit value(s) along with the DS and DR SB commands as follows (multiple 8-bit values are allowed only if each is greater than zero and less than 251):

8-bit value	Meaning
0	Command sender suggests that he alone will handle the vertical tabstops, for the connection.
1 to 250	Command sender suggests that the other party alone should handle the stops, but suggests that the indicated value(s) be used. Each value is the line number, relative to the top of the printer page or terminal screen, that is to be set as a vertical tabstop.
251 to 254	Not allowed, in order to be compatible with related Telnet options.
255	Command sender suggests that the other party alone should handle output vertical tabstops and suggests nothing about how it should be done.

The guiding rules are that:

1) if neither data receiver nor data sender wants to handle output vertical tabstops, the data receiver must do it, and
2) if both data receiver and data sender want to handle output vertical tabstops, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOVTS option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. This is necessary due to the assynchrony of network transmissions.
As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).
At any time, either party can disable further negotiation by giving the appropriate WON'T NAOVTS or DON'T NAOVTS command.

D. Crocker (UCLA-NMC)
RFC 657, NIC 31160 (Oct. 25, 1974)
Online file: [ISI]<DCROCKER>NAOVTD.TXT

TELNET OUTPUT VERTICAL TAB DISPOSITION OPTION

1. Command name and code
   NAOVTD 15
   (Negotiate About Output Vertcial Tab Disposition)

2. Command meanings
   In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet Options specifications.
   IAC DO NAOVTD
   The data sender requests or agrees to negotiate about output vertical tab character disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output vertical tab character considerations.
   IAC DON'T NAOVTD
   The data sender refuses to negotiate about output vertical tab character disposition with the data receiver, or demands a return to the unnegotiated default mode.
   IAC WILL NAOVTD
   The data receiver requests or agrees to negotiate about output vertical tab character disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output vertical tab character considerations.
   IAC WON'T NAOVTD
   The data receiver refuses to negotiate about output vertical tab character disposition, or demands a return to the unnegotiated default mode.
   IAC SB NAOVTD DS <8-bit value> IAC SE
   The data sender specifies, with the 8-bit value, which party should handle output vertical tab characters and what their disposition should be. The code for DS is 1.
   IAC SB NAOVTD DR <8-bit value> IAC SE
   The data receiver specifies, with the 8-bit value, which party should handle output vertical tab characters and what their disposition should be. The code for DR is 0.

3. Default
   DON'T NAOVTD/WON'T NAOVTD
   In the default absence of negotiations concerning which party, data sender or data receiver, is handling output vertical tab character considerations, neither party is required to handle vertical tab characters and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles vertical tab character considerations, albeit primitively.

4. Motivation for the Option
   Please refer to section 4 of the NAOL and of the NAOVTD Telnet option descriptions.

5. Description of the Option
The data sender and the data receiver use the 8-bit value along with the DS and DR SB commands as follows:

8 bit value	Meaning
0	Command sender suggests that he alone will handle vertical tab characters, for the connection.
1 to 250	Command sender suggests that the other party alone should handle tab characters, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character.
251	Command sender suggests that the other party alone handle vertical tabs, but suggests that each occurrence of the character be replaced by carriage-return/linefeed.
252	Command sender suggests that the other party alone handle vertical tabs, but suggests that they be discarded
253	Command sender suggests that the other party alone should handle tab characters, but suggests that tabbing be simulated.
254	Command sender suggests that the other party alone should handle the output disposition but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
255	Command sender suggests that the other party alone should handle the output disposition and suggests nothing about how it should be done.

The guiding rules are that:

1. if neither data receiver nor data sender wants to handle the output vertical tab characters, the data receiver must do it, and
2. if both data receiver and data sender want to handle the output vertical tab characters, the data sender gets to do it.

The reasoning for the former rule is that if neither want to do it, then the default in the NAOVTD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. Simulation is defined as the replacement of the character by enough line-feeds (only) to advance the paper (or line-pointer) to the next vertical tab stop.
Note that delays, controlled by the data sender, must consist of NUL characters, inserted immediately after the line-feed character. This is necessary due to the assynchrony of network transmissions. As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOVTD or DON'T NAOVTD command.

D. Crocker (UCLA-NMC)
RFC 658, NIC 31161 (Oct. 25, 1974)
Online file: [ISI]<DCROCKER>NAOLFD.TXT

TELNET OUTPUT LINEFEED DISPOSITION

1. Command name and code
   NAOLFD 16
      (Negotiate About Output Linefeed Disposition)

2. Command meanings
   In the following, we are discussing a simplex connection, as described in the NAOL and NAOP Telnet Options.
      IAC DO NAOLFD
         The data sender requests or agrees to negotiate about output linefeed disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output linefeed considerations.
      IAC DON'T NAOLFD
         The data sender refuses to negotiate about output linefeed disposition with the data receiver, or demands a return to the unnegotiated default mode.
      IAC WILL NAOLFD
         The data receiver requests or agrees to negotiate about output linefeed disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output linefeed considerations.
      IAC WON'T NAOLFD
         The data receiver refuses to negotiate about output linefeed disposition, or demands a return to the unnegotiated default mode.
      IAC SB NAOLFD DS <8-bit value> IAC SE
         The data sender specifies, with the 8-bit value, which party should handle output linefeeds and what their disposition should be. The code for DS is 1.
      IAC SB NAOLFD DR <8-bit value> IAC SE
         The data receiver specifies, with the 8-bit value, which party should handle output linefeeds and what their disposition should be. The code for DR is 0.

3. Default
   DON'T NAOLFD/WON'T NAOLFD.
      In the default absence of negotiations concerning which party, data under or data receiver, is handling output linefeed considerations, neither party is required nor prohibited from handling linefeeds; but it is appropriate if at least the data receiver handles them, albeit primitively.

4. Motivation for the Option
   Please refer to section 4 of the NAOL and of the NAOLFD Telnet option descriptions.

5. Description of the Option
The data sender and the data receiver use the 8-bit value along with DS and DR SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle linefeeds, for the connection.
1 to 250	Command sender suggests that the other party alone should handle linefeeds, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character. (See qualifications, below.)
251	Not allowed, in order to be compatible with related Telnet options.
252	Command sender suggests that the other party alone handle linefeeds, but suggests that they be discarded.
253	Command sender suggests that the other party alone should handle linefeeds, but suggests that linefeeds be simulated.
254	Command sender suggests that the other party alone should handle output linefeeds but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. (See qualifications, below.) Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
255	Command sender suggests that the other party alone should handle output linefeeds and suggests nothing about how it should be done.

The guiding rules are that:

1) if neither data receiver nor data sender wants to handle output linefeeds, the data receiver must do it, and
2) if both data receiver and data sender want to handle output linefeed disposition, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOLFD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. Simulation is defined as the replacement of the linefeed character by new-line (see following) and enough blanks to move the print head (or line pointer) to the same lateral position it occupied just prior to receiving the linefeed. To avoid infinite recursion, such simulation is allowed only for linefeed characters that are not immediately preceded by carriage-returns (that is, part of a Telnet new-line combination). It is assumed that linefeed simulation will be necessary for printers that do not have a separate linefeed (like the IBM 2741); in this case, end-of-line character padding can be specified through NAOCRD. Any padding (0 < <8-bit-value> < 251) of linefeed characters is to be done for ALL linefeed characters.

NOTE: Delays, controlled by the data sender, must consist of NUL characters inserted immediately after the character. This is necessary due to the assynchrony of network transmissions. Additionally, due to the presence of the Telnet end-of-line convention, it may be necessary to add carriage-return padding or delay after the associated linefeed (see NAOCRD Telnet option). As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts). At any time, either party can disable further negotiation by giving the appropriate WON'T NAOLFD or DON'T NAOLFD command.

TELNET EXTENDED ASCII OPTION
RFC 698, NIC 32964 (July 23, 1975)

TELNET EXTENDED ASCII OPTION

1. Command Name and code.

   EXTEND-ASCII 17

2. Command Meanings.

   IAC WILL EXTEND-ASCII

      The sender of this command requests permission to begin transmitting, or confirms that it may now begin transmitting extended ASCII, where additional 'control' bits are added to normal ASCII, which are treated specially by certain programs on the host computer.

   IAC WON'T EXTEND-ASCII

      If the connection is already being operated in extended ASCII mode, the sender of this command demands that the reciever begin transmitting data characters in standard NVT ASCII. If the connection is not already being operated in extended ASCII mode, The sender of this command refuses to begin transmitting extended ASCII.

   IAC DO EXTEND-ASCII

      The sender of this command requests that the receiver begin transmitting,or confirms that the receiver of this command is allowed to begin transmitting extended ASCII.

   IAC DON'T EXTEND-ASCII

      The sender of this command demands that the receiver of this command stop or not start transmitting data in extended ASCII mode.

   IAC SB EXTASC

      <high order bits (bits 15-8)><low order bits (bits 7-0)> IAC SE

      This command transmits an extended ASCII character in the form of two 8-bit bytes. Each 8-bit byte contains 8 data bits.

TELNET EXTENDED ASCII OPTION
RFC 698, NIC 32964 (July 23, 1975)

3. Default

DON'T EXTEND-ASCII

WON'T EXTEND-ASCII

i.e., only use standard NVT ASCII

4. Motivation.

Several sites on the net, for example, SU-AI and MIT-AI, use keyboards which use almost all 128 characters as printable characters, and use one or more additional bits as 'control' bits as command modifiers or to separate textual input from command input to programs. Without these additional bits, several characters cannot be entered as text because they are used for control purposes, such as the greek letter 'beta' which on a TELNET connection is CONTROL-C and is used for stopping ones job. In addition there are several commonly used programs at these sites which require these additional bits to be run effectively. Hence it is necessary to provide some means of sending characters larger than 8 bits wide.

5. Description of the option.

This option is to allow the transmission of extended ASCII.

Experience has shown that most of the time, 7-bit ASCII is typed, with an occasional 'control' character used. Hence, it is expected normal NVT ASCII would be used for 7-bit ASCII and that extended-ASCII be sent as an escape character sequence.

The exact meaning of these additional bits depends on the user program. At SU-AI and at MIT-AI, the first two bits beyond the normal 7-bit ASCII are passed on to the user program and are denoted as follows.

Bit 8 (or 200 octal) is the CONTROL bit
Bit 9 (or 400 octal) is the META bit

(NOTE: 'CONTROL' is used in a non-standard way here; that is, it usually refers to codes 0-37 in NVT ASCII. CONTROL and META are echoed by prefixing the normal character with 013 (integral symbol) for CONTROL and 014 (plus-minus) for META. If both are present, it is known as CONTROL-META and echoed as 013 014 7-bit character.)

6. Description of Stanford Extended ASCII Characters

In this section, the extended graphic character set used at SU-AI is described for reference, although this specific character set is not required as part of the extended ASCII Telnet option. Characters described as "hidden" are alternate graphic interpretations of codes normally used as format effectors, used by certain typesetting programs.

Code	Graphic represented
000	null (hidden vertically centered dot)
001	downward arrow
002	alpha (all Greek letters are lowercase)
003	beta
004	logical and (caret)
005	logical not (dash with downward extension)
006	epsilon
007	pi
010	lambda
011	tab (hidden gamma)
012	linefeed (hidden delta)
013	vertical tab (hidden integral)
014	formfeed (hidden plus-minus)
015	carriage return (hidden circled-plus)
016	infinity
017	del (partial differential)
020	proper subset (right-opening horseshoe)
021	proper superset (left-opening horseshoe)
022	intersection (down-opening horseshoe)
023	union (up-opening horseshoe)
024	universal quantifier (upside-down A)
025	existential quantifier (backwards E)
026	circled-times
027	left-right double headed arrow
030	underbar
031	right pointing arrow
032	tilde
033	not-equal
034	less-than-or-equal
035	greater-than-or-equal
036	equivalence (column of 3 horizontal bars)
037	logical or (V shape)
040-135	as in standard ASCII

TELNET EXTENDED ASCII OPTION
RFC 698, NIC 32964 (July 23, 1975)

136	upward pointing arrow
137	left pointing arrow
140-174	as in standard ASCII
175	altmode (prints as lozenge)
176	right brace
177	rubout (hidden circumflex)

NWG/RFC# 727 MRC 26-APR-77 18:24 40025
Telnet Logout Option

Network Working Group Mark Crispin
Request for Comments 727 MIT-AI
NIC 40025 27 April 1977

TELNET Logout Option

1. Command name and code.

LOGOUT 18

2. Command meanings.

IAC WILL LOGOUT

The sender of this command REQUESTS permission to, or confirms that it will, forcibly log off the user process at its end.

IAC WON'T LOGOUT

The sender of this command REFUSES to forcibly log off the user process at its end.

IAC DO LOGOUT

The sender of this command REQUESTS that the receiver forcibly log off the user process at the receiver's end, or confirms that the receiver has its permission to do so.

IAC DON'T LOGOUT

The sender of this command DEMANDS that the receiver not forcibly log off the user process at the receiver's end.

3. Default.

WON'T LOGOUT

DON'T LOGOUT

i.e., no forcible logging off of the server's user process.

4. Motivation for the option.

Often, a runaway user process could be hung in such a state that it cannot be interrupted by normal means. Conversely, the system itself could be bottlenecked so that response delays are intolerable. A user (human or otherwise) eventually will time out out of frustration

and take the drastic means of closing the connection to free itself from the hung process. In some situations, even the simple operation of logging out can take a long time.

Some systems treat a close to mean that it should log out its user process under it. However, many hosts merely "detach" the process so that an accidental close due to a user or temporary hardware error will not cause all work done on that job to be lost; when the connection is re-established, the user may "attach" back to its process. While this protection is often valuable, if the user is giving up completely on the host, it can cause this hung job to continue to load the system.

This option allows a process to instruct the server that the user process at the server's end should be forcibly logged out instead of detached. A secondary usage of this option might be for a server to warn of impending auto-logout of its user process due to inactivity.

5. Description of the option.

When a user decides that it no longer wants its process on the server host and decides that it does not want to wait until the host's normal log out protocol has been gone through, it sends IAC DO LOGOUT. The receiver of the command may respond with IAC WILL LOGOUT, in which case it will then forcibly log off the user process at its end. If it responds with IAC WON'T LOGOUT, then it indicates that it has not logged off the user process at its end, and if the connection is broken, the process very possibly will be detached.

A truly impatient user that feels that it must break away from the server immediately could even send IAC DO LOGOUT and then close. At the worst, the server would only ignore the request and detach the user process. A server that implements the LOGOUT option should know to log out the user process despite the sudden close and even an inability to confirm the LOGOUT request!

6. A sample implementation of the option.

The server implements the LOGOUT option both for accepting LOGOUT requests and for auto-logout warning.

Case 1:

The user connects to the server, and starts interacting with the server. For some reason, the user wishes to terminate interaction with the server, and is reluctant to go through the normal log out procedure, or perhaps the user is unable to go through the normal

log out procedure. It does not want the process at the server any more, so it sends IAC DO LOGOUT. The server verifies the request with IAC WILL LOGOUT, and then forcibly logs off the user process (perhaps by using a system call that causes another process to be logged out). It does not have to close the connection unless the user closes or it wants to close. Neither does it wait until the user has received its confirmation--it starts the log out immediately so if the user has in the mean time closed the connection without waiting for confirmation, its logout request still is performed.

Case 2:

The user connects to the server, and after logging in, is idle for a while, long enough to approach the server's autologout time. The server shortly before the autologout sends IAC WILL LOGOUT; the user sees this and sends IAC DON'T LOGOUT, and continues work on the host. Nothing prevents the server from logging out the user process if inactivity continues; this can be used to prevent a malicious user from locking up a process on the server host by the simple expedient of sending IAC DON'T LOGOUT every time it sees IAC WILL LOGOUT but doing nothing else.

RFC 735 DHC RHG 3 Nov 77 42083
Telnet Byte Macro Option

Network Working Group
RFC: #735
NIC: #42083

David H. Crocker
Rand-ISD
(Dcrocker at Rand-Unix)
Richard H. Gumpertz
Carnegie-Mellon University
(Gumpertz at CMU-10A)

Obsoletes: RFC #729 (NIC #40306) 3 November 1977

Revised TELNET Byte Macro Option

1. Command name and code:

   BM 19

2. Command Meanings:

   IAC WILL BM

      The sender of this command REQUESTS or AGREES to use the BM option, and will send single data characters which are to be interpreted as if replacement data strings had been sent.

   IAC WON'T BM

      The sender of this option REFUSES to send single data characters which are to be interpreted as if replacement data strings had been sent. Any existing BM <macro byte> definitions are discarded (i.e., reset to their original data interpretations).

   IAC DO BM

      The sender REQUESTS or AGREES to have the other side (sender of WILL BM) send single data characters which are to be interpreted as if replacement data strings had been sent.

   IAC DON'T BM

      The sender REFUSES to allow the other side to send single data characters which are to be interpreted as if replacement data strings had been sent. Any existing BM <macro byte> definitions are to be discarded.

RFC 735 DHC RHG 3 Nov 77 42083
Telnet Byte Macro Option

IAC SB BM <DEFINE> <macro byte> <count>
<replacement string> IAC SE

where:

<macro byte> is the data byte actually to be sent across the network; it may NOT be Telnet IAC (decimal 255, but may be any other 8-bit character.

<count> is one 8-bit byte binary number, indicating how many <replacement string> characters follow, up to the ending IAC SE, but not including it. Note that doubled IACs in the definition should only be counted as one character per pair.

<replacement string> is a string of zero or more Telnet ASCII characters and/or commands, which the <macro byte> is to represent; any character may occur within a <replacement string>. Note, however, that an IAC in the string must be doubled, to be interpreted later as an IAC; to be interpreted later as data byte 255, it must be quadrupled in the original <replacement string> specification.

The indicated <macro byte> will be sent instead of the indicated <replacement string>. The receiver of the <macro byte> (the sender of the DO BM) is to behave EXACTLY as if the <replacement string> string of bytes had instead been received from the network. This interpretation is to occur before any other Telnet interpretations, unless the <macro byte> occurs as part of a Telnet command; in this case no special interpretation is to be made. In particular, an entire Telnet subnegotiation (i.e. from IAC SB through IAC SE) is to be considered a Telnet command in which NO replacement should be done.

The effect of a particular <macro byte> may be negated by reseting it to "expand" into itself.

IAC SB BM <DEFINE> X <0> IAC SE may be used to cause X to be ignored in the data stream.

<DEFINE> is decimal 1.

IAC SB BM <ACCEPT> <macro byte> IAC SE

The receiver of the <DEFINE> for <macro byte> accepts the requested definition and will perform the indicated replacement whenever a <macro byte> is received and is not part of any IAC Telnet command sequence.

<ACCEPT> is decimal 2.

IAC SB BM <REFUSE> <macro byte> <REASON> IAC SE

The receiver of the <DEFINE> for <macro byte> refuses to perform the indicated translation from <macro byte> to <replacement string> because the particular <macro byte> is not an acceptable choice, the length of the <replacement string> exceeds available storage, the length of the actual <replacement string> did not match the length predicted in the <count>, or for some unspecified reason.

<REFUSE> is decimal 3.

<REASON> may be

<BAD-CHOICE>	which is decimal 1;
<TOO-LONG>	(for receiver's storage) which is decimal 2;
<WRONG-LENGTH>	(of actual string compared with promised length in <count>) which is decimal 3; or
<OTHER-REASON>	(intended for use only until this document can be updated to include reasons not anticipated by the authors) which is decimal zero (0).

IAC SB BM <LITERAL> <macro byte> IAC SE

The <macro byte> is to be treated as real data, rather than as representative of the <replacement string>

Note that this subcommand cannot be used during Telnet subcommands, since subcommands are defined to end with the next occurrence of "IAC SE". Including this BM subcommand within any Telnet subcommand would therefore prematurely terminate the containing subcommand.

<LITERAL> is decimal 4.

IAC SB BM <PLEASE CANCEL> <macro byte> <REASON> IAC SE

The RECEIVER of the defined <macro byte> (i.e., the sender of IAC DO BM) requests the sender of <macro byte> to cancel its definition. <REASON> is the same as for the <REFUSE> subcommand.

The <macro byte> sender should (but is not required to) respond by resetting <macro byte> (i.e., sending an IAC SB BM <DEFINE> <macro byte> <1> <macro byte> IAC SE).

If the receiver absolutely insists on cancelling a given macro, the best it can do is to turn off the entire option, with IAC DONT BM, wait for an acknowledging IAC WONT BM and then restart the option, with IAC DO BM. This will reset all other macroes as well but it will allow the receiver to REFUSE with code BAD CHOICE if/when the foreign site attempts to redefine the macro in question.

3. Default:

WON'T BM -- DON'T BM

No reinterpretation of data bytes is done.

4. Motivation for the option:

Subcommands for Telnet options currently require a minimum of five characters to be sent over the network (i.e., IAC SB <Option name> IAC SE). For subcommands which are employed infrequently, in absolute numbers and in relation to normal data, this overhead is tolerable. In other cases, however, it is not. For example, data which is sent in a block- oriented fashion may need a "block separator" mark. If blocks are commonly as small as five or ten bytes, then most of the cross-net data will be control information. The BM option is intended as a simple data compression technique, to remove this overhead from the communication channel.

5. Description of the option

The option is enabled through the standard Telnet Option negotiation process. Afterwards, the SENDER of data (the side which sends the IAC WILL BM) is free to define and use mappings between single and replacement NVT characters. Except for the ability to refuse particular definitions, the receiver of data has no control over the definition and use of mappings.

The sender (of the WILL BM) is prohibited from using or redefining a <macro byte> until it has received an <ACCEPT> <REFUSE>, or DONT BM, in reply to a <DEFINE>.

NOTE: The Telnet command character IAC (decimal 255) may be a member of a <replacement string> but is the ONLY character which may NOT be defined as a <macro byte>.

Within any Telnet command (i.e., any sequence beginning with IAC) macro replacement may NOT take place. Data are to be interpreted only as their normal character values. This avoids the problem of distinguishing between a character which is to be taken as a <macro byte>, and interpreted as its corresponding <replacement string>, and one which is to be taken as its usual Telnet NVT value. In all other cases, however, <macro byte>s are to be interpreted immediately, as if their corresponding <replacement string>s had actually been sent across the network. Expanded strings are not subject to reinterpretation, so that recursive definitions cannot be made. Telnet commands may be included in <replacement strings>; however, they must be totally contained within the macro or must begin within the macro and terminate outside of it. In particular, they may NOT begin outside a macro and continue or terminate inside one, since no macro replacement takes place while processing any Telnet command.

Note that when skipping data due to Telnet SYNCH (INS/DM) processing, BM macro replacement should still take place, since (for example) "IAC DM" would be a valid <replacement string>.

The <count> in the <DEFINE> subcommand is intended to allow the receiver to allocate storage. IAC interpretation is not over-ridden during BM subcommands so that IAC SE will continue to safely terminate malformed subcommands.

The BM option is notably inefficient with regard to problems during <macro byte> definition and use of <macro byte>s as real data. It is expected that relatively few <macro byte>s will be defined and that they will represent relatively short strings. Since the Telnet data space between decimal 128 and decimal 254 is not normally used, except by implementations employing the original (obsolete) Telnet protocol, it is recommended that <macro byte>s normally be drawn from that pool.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

Network Working Group John Day
Request for Comments: 732
NIC: 41762 12 September 1977

Obsoletes: 731

Telnet Data Entry Terminal Option

1. Command Name and Code:

DET 20

2. Command Meanings

IAC WILL DET

The sender of this command REQUESTS or AGREES to send and receive subcommands to control the Data Entry Terminal.

IAC WONT DET

The sender of this command REFUSES to send and receive subcommands to control the Data Entry Terminal.

IAC DO DET

The sender of this command REQUESTS or AGREES to send and receive subcommands to control the Data Entry Terminal.

IAC DONT DET

The sender of this command REFUSES to send and receive subcommands to control the Data Entry Terminal.

The DET option uses five classes of subcommands 1) to establish the requirements and capabilities of the application and the terminal, 2) to format the screen, and to control the 3) edit, 4) erasure, and 5) transmission functions. The subcommands that perform these functions are described below.

The Network Virtual Data Entry Terminal (NVDET)

The NVDET consists of a keyboard and a rectangular display. The keyboard is capable of generating all of the characters of the ASCII character set. In addition, the keyboard may possess a number of function keys which when pressed cause a FN subcommand to be sent.

—

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

(Although most DET's will support one or more peripheral devices such as a paper tape reader or a printer, this option does not consider their support. Support of peripheral devices should be treated by a is a separate option).

The screen of the data entry terminal is a rectangle M characters by N lines. The values of M and N are set by negotiating the Output Line Width and Output Page Size options, respectively. The next writing position (x,y) on the screen (where x is the character position and y is the position of the line on the screen) is indicated by a special display character called the cursor. The cursor may be moved to any position on the screen without disturbing any characters already on the screen. Cursor addressing in existing terminals utilizes several topologies and addressing methods. In order to make the burden of implementaton as easy as possible this protocol supports two topologies (the finite plane and the helical torus) and three addressing methods ((x,y); x and y, and relative increments). Since the finite plane with absolute addressing is the least ambiguous and the easiest to translate to and from the others, it is the default scheme used by the NVDET. The torodial form with either relative or absolute addressing is provided for convience.

Also the NVDET provides a mechanism for defining on the screen fields with special attributes. For example, characters entered into these fields may be displayed with brighter intensity, highlighted by reverse video or blinking, or protected from modification by the user. This latter feature is one of the most heavily used for applications where the DET displays a form to be filled out by the user.

The definition of the NVDET uses Telnet option subnegotiations to accomplish all of its functions. Since none of the ASCII characters sent in the data stream have been used to define these functions, the DET option can be used in a "raw" or even "rare" mode. In circumstances where the application program knows what kind of terminal is on the other end, it can send the ASCII characters required to control functions not supported by the option or an implementation. In general keeping all NVDET functions out of the data stream provides better flexibility.

Facility Functions (for detailed semantics see Section 5.)

IAC SB DET <DET facility subcommand><facility map> IAC SE

where <DET facility subcommand> is one 8-bit byte indicating the class of the facilities to be described, and <facility map> is a field of one or two 8-bit bytes containing flags describing the

facilities required or desired by the sender. The bits of the facility maps are numbered from the right starting at zero. Thus, if bit 2 is set the field will have a decimal value of 4. The values of the field are as follows:

facility cmd: EDIT FACILITIES subcommand code: 1

facility map:	bit numbers
Toroidal Cursor Addressing	6
Incremental Cursor Addressing	5
Read Cursor Address	4
Line Insert/Delete	3
Char Insert/Delete	2
Back Tab	1
Positive Addressing only	0

where:

If the Toroidal Cursor Addressing bit is set, the sender requests or provides that the SKIP TO LINE and SKIP TO CHAR subcommands be supported.

If the Incremental Cursor Addressing bit is set, the sender requests or provides that the UP, DOWN, LEFT, and RIGHT subcommands be supported.

If the Read Cursor bit is set, the sender requests or provides the READ CURSOR subcommand.

If the Line Insert/Delete bit is set, the sender requests or provides that the LINE INSERT and LINE DELETE subcommands be supported.

If the Char Insert/Delete bit is set, the sender requests or provides that the CHAR INSERT and CHAR DELETE subcommands be supported.

If the Back Tab bit is set, the sender requests or provides that the BACK TAB subcommand be supported.

If the Positive Addressing bit is set, then the sender is informing the receiver that it can only move the cursor in the positive direction. (Note: Terminals that have this property also have a Home function to get back to the beginning.)

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

facility cmd: ERASE FACILITIES subcommand code: 2

facility map:	bit numbers
Erase Field	4
Erase Line	3
Erase Rest of Screen	2
Erase Rest of Line	1
Erase Rest of Field	0

where:

If a bit of the facility map for this facility command is set, the sender requests or provides the facility indicated by the bit. For a more complete description of each of these functions see the Erase Functions section below.

facility cmd: TRANSMIT FACILITIES subcommand code: 3

facility map:	bit numbers
Data Transmit	5
Transmit Line	4
Transmit Field	3
Transmit Rest of Screen	2
Transmit Rest of Line	1
Transmit Rest of Field	0

where:

If a bit of the facility map for this facility command is set, the sender requests or provides the facility indicated by the bit. For a more complete description of each of these functions see the Transmit Functions section below.

facility cmd: FORMAT FACILITIES subcommand code: 4

facility map:		bit numbers
EN	byte 0	7
Modified		6
Light Pen		5
Repeat		4
Blinking		3
Reverse Video		2
Right Justification		1
Overstrike		0

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

Protection On/Off	byte 1	6
Protection		5
Alphabetic-only Protection		4
Numeric-only Protection		3
Intensity		0-2

where:

If the FN bit is set, the sender requests or provides the FN subcommand.

If the Modified bit is set, the sender requests or provides the ability to indicate fields that are modified and supports the TRANSMIT MODIFIED subcommand.

If the Light Pen bit is set, the sender requests or provides the support of a light pen, including the Pen Selectable attribute of the DATA FORMAT subcommand.

If the Repeat bit is set the sender requests or provides the REPEAT subcommand.

If the Blinking bit is set, the sender requests or provides the ability to highlight a string of characters by causing them to blink.

If the Reverse Video bit is set, the sender requests or provides the ability to highlight a string of characters by "reversing the video image," i.e., if the characters are normally displayed as black characters on a white background, this is reversed to be white characters on a black background, or vice versa.

If the Right Justification bit is set, the sender requests or provides the ability to cause entries of data to be right justified in the field.

If the Overstrike bit is set, the sender requests or provides the ability to superimpose one character over another on the screen much like a hard copy terminal would do if the print mechanism struck the same position on the paper with different characters.

If the Protection On/Off bit is set, the sender requests or provides the ability to turn on and off field protection.

If the Protection bit is set, the sender requests or provides the ability to protect certain strings of

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

characters displayed on the screen from being altered by the user of the terminal. Setting this bit also implies that ERASE UNPROTECTED, DATA TRANSMIT, FIELD SEPARATOR, and TRANSMIT UNPROTECTED subcommands (see below) are supported.

If the Alphabetic-only Protection bit is set, the sender requests or provides the ability to constrain the user of the terminal such that he may only enter alphabetic data into certain areas of the screen.

If the Numeric-only Protection bit is set, the sender requests or provides the ability to constrain the user of the terminal such that he may only enter numerical data into certain areas of the screen.

The three bits of the Intensity field will contain a positive binary integer indicating the number of levels of intensity that the sender requests or provides for displaying the data. The value of the 3 bit field should be interpreted in the following way:

```
1       one visible intensity
2       two intensities; normal and bright
3       three intensities; off, normal, and bright
>3       >3 intensities; off, and the remaining levels
proportioned from dimmest to brightest intensity.
```

For the all of the above commands, if the appropriate bit in <facility map> is not set, then the sender does not request or provide that facility.

Editing Functions

IAC SB DET MOVE CURSOR <x><y> IAC SE subcommand code: 5

where <x> is an 8-bit byte containing a positive binary integer representing the character position of the cursor, <y> is an 8-bit byte containing a positive binary integer representing the line position of the cursor.

This subcommand moves the cursor to the absolute screen address (x,y) with the following boundary conditions:

if x>M-1, set x=M-1 and send an ERROR subcommand

if y>N-1, set y=N-1 and send an ERROR subcommand

This describes a finite plane topology on the screen.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

IAC SB DET SKIP TO LINE <y> IAC SE subcommand code: 6

where <y> is a positive 8-bit binary number.

This subcommand moves the cursor to the absolute screen line y. x remains constant. For values of y>N-1

y = y mod N.

IAC SB DET SKIP TO CHAR <x> IAC SE subcommand code: 7

where <x> is a positive 8-bit binary number.

This subcommand moves the cursor to the absolute character position x. y remains constant, unless x>M-1 in which case:

x' = (x mod M)
y' = (y+(x DIV N))
where x' and y' are the new values of the cursor.

These last two subcommands define a toroidal topology on the screen.

IAC SB DET UP IAC SE subcommand code: 8

IAC SB DET DOWN IAC SE subcommand code: 9

IAC SB DET LEFT IAC SE subcommand code: 10

IAC SB DET RIGHT IAC SE subcommand code: 11

These subcommands are provided as a convenience for some terminals. The commands UP, DOWN, LEFT, and RIGHT are defined as

UP: (x,y)=(x, y-1 mod N)
DOWN: (x,y)=(x, y+1 mod N)
LEFT: (x,y)=(x-1, y); if x=0 then x-1 = 0

RIGHT: (x,y)=(x+1 mod M, y) and y = y+1 if x+1>M-1

Note: DOWN, LEFT, and RIGHT cannot always be replaced by the ASCII codes for linefeed, backspace, and space respectively. The latter are format effectors while the former are cursor controls.

IAC SB DET HOME IAC SE subcommand code: 12

This subcommand positions the cursor to (0,0). This is equivalent to a MOVE CURSOR 0,0 or the sequence SKIP TO LINE 0, SKIP TO CHAR 0.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

This subcommand is provided for convenience, since most terminals have it as a separate control.

IAC SB DET LINE INSERT IAC SE subcommand code: 13

This subcommand inserts a line of spaces between lines y (the current line, determined by the position of the cursor) and line y-1. Lines y through N-2 move down one line, i.e. line y becomes line y+1; y+1 becomes y+2, ...; N-2 becomes N-1. Line N-1 is lost off the bottom of the screen. The position of the cursor remains unchanged.

IAC SB DET LINE DELETE IAC SE subcommand code: 14

This subcommand deletes line y where y is the current line position of the cursor. Lines y+1 through N-1 move up one line, i.e. line y+1 becomes line y; y+2 becomes y+1; ...; N-1 becomes N-2. The N-1st line position is set to all spaces. The cursor position remains unchanged.

IAC SB DET CHAR INSERT IAC SE subcommand code: 15

This subcommand inserts the next character in the data stream between the xth and x-1st characters, where x is the current character position of the cursor. The xth through M-2nd characters on the line are shifted one character positon to the right. The new character is inserted at the vacated xth position. The M-1st character is lost. The position of the cursor remains unchanged.

IAC SB DET CHAR DELETE IAC SE subcommand code: 16

This subcommand deletes the character on the screen at the x-th position. The x-th character is removed and the characters x+1 through M-1 are shifted one character position to the left to become the x-th through M-2nd characters. The M-1st character position is left empty. (For most terminals it will be set to a NUL or space.) The cursor position remains unchanged.

IAC SB DET READ CURSOR IAC SE subcommand code: 17

This subcommand requests the receiver to send the present position of the cursor to the sender.

IAC SB DET CURSOR POSITION <x><y> IAC SE subcommand code: 18

where <x> and <y> are positive 8-bit binary integers.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

This subcommand is sent by a Telnet implementation in response to a READ CURSOR subcommand to convey the coordinates of the cursor to the other side. Note: x is less than M and y is less than N.

IAC SB DET REVERSE TAB IAC SE subcommand code: 19

This subcommand causes the cursor to move to the previous tab position. If none exists on the present line, the cursor moves to the previous line and so on until a tab is found or the address (0,0) is encountered. When field protection is in effect the cursor moves to the beginning of the preceding unprotected field.

Transmit Functions (For detailed semantics see Section 5.)

IAC SB DET TRANSMIT SCREEN IAC SE subcommand code: 20

This subcommand causes the terminal to transmit all characters on the screen from position (0,0) to (M-1,N-1). The cursor will be at (0,0) after the operation is complete.

IAC SB DET TRANSMIT UNPROTECTED IAC SE subcommand code: 21

This subcommand causes the terminal to transmit all characters in unprotected fields from position (0,0) to (M-1,N-1). The unprotected fields are separated by the field separator subcommand. The cursor will be at (0,0) or at the beginning of the first unprotected field after the operation is complete.

IAC SB DET TRANSMIT LINE IAC SE subcommand code: 22

This subcommand causes the terminal to transmit all data on the yth line where y is determined by the present position of the cursor. Data is sent from character position (0,y) to the end-of-line or position (M-1,y) whichever comes first. The cursor position after the transmission is one character position after the end of line condition or the beginning of the next line, (0,y+1).

IAC SB DET TRANSMIT FIELD IAC SE subcommand code: 23

This subcommand causes the terminal to transmit all data in the field presently occupied by the cursor. The cursor position after the operation is complete is one character position after the end of the field or, if that

position is protected, at the beginning of the next unprotected field.

IAC SB DET TRANSMIT REST OF SCREEN IAC SE subcommand code: 24

This subcommand causes the terminal to transmit all characters on the screen from position (x,y) to (M-1,N-1) or until the end of text. (x,y) is the current cursor position. The cursor position after the operation is one character position after the last text character, or (0,0) if the last filled character position is (M-1,N-1).

IAC SB DET TRANSMIT REST OF LINE IAC SE subcommand code: 25

This subcommand causes the terminal to transmit all characters on the yth line from position (x,y) to the end of line or (M-1,y) whichever comes first. (x,y) is the current cursor position. The cursor position after the operation is one character position after the last character of the line or the first character of the next line.

IAC SB DET TRANSMIT REST OF FIELD IAC SE subcommand code: 26

This subcommand causes the receiver to transmit the rest of the characters in the field currently occupied by the cursor. The cursor position after the operation is at the beginning of the next field.

IAC SB DET TRANSMIT MODIFIED IAC SE subcommand code: 27

This subcommand causes the receiver to transmit only those fields which have the modified attribute set. The cursor position after the operation is unchanged.

IAC SB DET DATA TRANSMIT <x><y> IAC SE subcommand code: 28

This subcommand is used to preface data sent from the terminal in response to a user action or a TRANSMIT command. The parameters <x> and <y> indicate the initial position of the cursor. See the Transmit Subcommands subsection in Section 5 for more details. A DATA TRANSMIT subcommand may precede an entire transmission with each field being delineated by the FIELD SEPARATOR subcommand as would be the case in a response toa

TRANSMIT UNPROTECTED. Or, it may precede each field as would be the case in a response to a TRANSMIT MODIFIED.

Erase Functions

IAC SB DET ERASE SCREEN IAC SE subcommand code: 29

This subcommand causes all characters to be removed from the screen. All fields regardless of their attributes are deleted. The cursor position after the operation will be (0,0). Most terminals set the erased characters to either NUL or space characters.

IAC SB DET ERASE LINE IAC SE subcommand code: 30

This subcommand causes all characters on the yth line to be removed from the screen, where y is the line of the current cursor position. All fields regardless of their attributes are deleted. The cursor position after this operation will be (0,y). Note: This operation can be easily simulated by the sequence: LINE DELETE, LINE INSERT. However, the order is important to insure that no data is lost off the bottom of the screen.

IAC SB DET ERASE FIELD IAC SE subcommand code: 31

This subcommand causes all characters in the field occupied by the cursor to be removed. The cursor position after the operation is at the beginning of the field.

IAC SB DET ERASE REST OF SCREEN IAC SE subcommand code: 32

This subcommand causes all characters from position (x,y) to (M-1,N-1) to be removed from the screen. All fields regardless of their attributes are deleted. The cursor position after the operation is unchanged. This is equivalent to doing an ERASE REST OF LINE plus a LINE DELETE for lines greater than y.

IAC SB DET ERASE REST OF LINE IAC SE subcommand code: 33

This subcommand causes all characters from position (x,y) to (M-1,y) to be removed from the screen All fields regardless of their attributes are deleted. The cursor position after the operation is unchanged.

IAC SB DET ERASE REST OF FIELD IAC SE subcommand code: 34

This subcommand causes all characters from position (x,y) to the end of the current field to be removed from the screen. The cursor position after the operation is unchanged.

IAC SB DET ERASE UNPROTECTED IAC SE subcommand code: 35

This subcommand causes all characters on the screen in unprotected fields to be removed from the screen. The cursor position after the

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

operation is at (0,0) or, if that position is protected, at the beginning of the first unprotected field.

Format Functions

IAC SB DET FORMAT DATA <format map><count> IAC SE
subcommand code: 36

where <format map> is a two byte field containing the following flags:

Byte 0	
Blinking	7
Reverse Video	6
Right Justification	5
Protection	3-4
Intensity	0-2
Byte 1	
Modified	1
Pen Selectable	0

where:

If the Blinking bit is set, the following field of <count> characters should have the Blinking attribute applied to it by the receiver.

If the Reverse Video bit is set, the following field of <count> characters should be displayed by the receiver with video reversed.

If the Right Justification bit is set, the input entered into the field of <count> characters should be right justified.

The Protection field is two bits wide and may take on the

following values:

0	no protection
1	protected
2	alphabetic only
3	numeric only

The protection attribute specifies that the other side may modify any character (no protection), modify no characters (protected), enter only alphabetical characters (A-Z, and a-z) (alphabetic only), or enter only numerical characters (0-9,+,.,and -) (numeric only) in the following field of <count> bytes.

The Intensity field is 3 bits wide and should be interpreted in the following way:

The values 0-6 should be used as an indication of the relative brightness to be used when displaying the characters in or entered into the following field <count> characters wide. The number of levels of brightness available should have been obtained previously by the Format Facility subcommand. The exact algorithm for mapping these values to the available levels of intensity is left to the implementors. A value of 7 in the intensity field indicates that the brightness should be off, and any characters in or entered into the field should not be displayed.

If the Modified bit is set, the field is considered to have been modified and will be transmitted in response to a TRANSMIT MODIFIED subcommand.

If the Pen Selectable bit is set, the field can be selected with the light pen. Note: Use of the light pen should be the subject of another Telnet option.

<count> is 2 bytes that should be interpreted as a positive 16-bit binary integer representing the number of characters following this command which are affected by it.

Data sent to the terminal or the Using Host for unwritten areas of the screen not in the scope of the count should be displayed with the default values of the format map. The default values are No Blinking, Normal Video, No Justification, No Protection and Normal Intensity. For example, suppose a FORMAT DATA subcommand was sent to the terminal with attributes Blinking and Protected and a

count of 5 followed by the string "Name: John Doe". The string "Name:" would be protected and blinking, but the string "John Doe" would not be.

This subcommand is used to format data to be displayed on the screen of the terminal. The <format map> describes the attributes that the field <count> bytes wide should have. This field is to start at the position of the cursor when the command is acted upon. The next <count> displayable characters in the data stream are used to fill the field. Subsequent REPEAT subcommands may be used to specify the contents of this field. If the sender specifies attributes that have not been agreed upon by the use of the Format Facility subcommand, the Telnet process should send an Error Subcommand to the sender, but format the screen as if the bit had not been set.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

IAC SB DET REPEAT <count><char> IAC SE subcommand code: 37

where <count> is a positive 8-bit binary integer. <char> is an 8-bit byte containing an ASCII character.

This subcommand is used to perform data compression on data being transferred to the terminal by encoding strings of identical characters as the character and a count. The repeated characters may be part of a field specified

IAC SB DET SUPPRESS PROTECTION <negotiation> IAC SE
subcommand code: 38

where <negotiation> may have the values of the Telnet option negotiation:

251	WILL
252	WONT
253	DO
254	DONT

This subcommand is used to suppress the field protection in a non-destructive manner. Many data entry terminals provide the means by which protection may be turned on and off without modifying the contents of the screen or the terminal's memory. Thus, the protection may be turned off and back on without retransmitting the form.

The default setting of the option is that protection is on, in other words

IAC SB DET SUPPRESS PROTECTION WONT IAC SE
IAC SB DET SUPPRESS PROTECTION DONT IAC SE

Negotiation of this subcommand follows the same rules as negotiations of the Telnet options.

IAC SB DET FIELD SEPARATOR IAC SE subcommand code: 39

It is necessary when transmitting only the unprotected portion of the screen to provide a means for delimiting the fields. Existing DET's use a variety of ASCII characters such as Tab, Group Separator, Unit Separator, etc. In order to maintain transparency of the NVDET this subcommand is used to separate the fields. Clearly, this incurs rather high overhead. This overhead can be avoided by using the Byte Macro Option (see Appendix 3).

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

Miscellaneous Commands

IAC SB DET FN <code> IAC SE subcommand code: 40

where: <code> is one byte.

Many data-entry terminals provide a set of "function" keys which when pressed send a one-character command to the server. This subcommand describes such a facility. The values of the <code> field are defined by the user and server. The option merely provides the means to transfer the information.

IAC SB DET ERROR <cmd> <error code> IAC SE subcommand code: 41

where:

<cmd> is a byte containing the subcommand code of the subcommand in error.

<error code> is a byte containing an error code.

(For a list of the defined error codes see Appendix 2.)

This subcommand is provided to allow DET option implementations to report errors they detect to the corresponding Telnet process. At this point it is worth reiterating that the philosophy of this option is that when an error is detected it should be reported; however, the implementation should attempt its best effort to carry out the intent of the subcommand or data in error.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

3. Default and Minimal Implementation Specifications

Default

WON'T DET -- DON'T DET

Neither host wishes to use the Data Entry Terminal option.

Minimal Implementation

DET EDIT FACILITIES
DET ERASE FACILITIES
DET TRANSMIT FACILITIES
DET FORMAT FACILITIES
DET MOVE CURSOR <x><y>
DET HOME
DET ERASE SCREEN
DET TRANSMIT SCREEN
DET FORMAT DATA
DET ERROR <cmd> <error code>

In the case of formatting the data, the minimal implementation should be able to support a low and high level of intensity and protection for all or no characters in a field. These functions, however, are not required.

The minimal implementation also requires that the Output Line Width and Output Page Size Telnet options be supported.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

4. Motivation

The Telnet protocol was originally designed to provide a means for scroll-mode terminals, such as the standard teletype, to communicate with processes through the network. This was suitable for the vast majority of terminals and users at that time. However, as use of the network has increased into other areas, especially areas where the network is considered to provide a production environment for other work, the desires and requirements of the user community have changed. Therefore, it is necessary to consider supporting facilities that were not initially supported. This Telnet option attempts to do that for applications that require data entry terminals.

This option in effect defines the Network Virtual Data Entry Terminal. Although the description of this option is quite long, this does not imply that the Telnet protocol is a poor vehicle for this facility. Data Entry Terminals are rather complex and varied in their abilities. This option attempts to support both the minimal set of useful functions that are either common to all or can be easily simulated and the more sophisticated functions supplied in some terminals.

Unlike most real data entry terminals where the terminal functions are encoded into one or more characters of the native character set, this option performs all such controls within the Telnet subnegotiation mechanism. This allows programs that are intimately familiar with the kind of terminal they are communicating with to send commands that may not be supported by either the option or the implementation. In other words, it is possible to operate in a "raw" or at least "rare" mode using as much of the option as necessary.

Although many data entry terminals support a variety of peripheral devices such as printers, cassettes, etc. it is beyond the scope of this option to entertain such considerations. A separate option should be defined to handle this aspect of these devices.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

5. Description

General Notes

All implementations of this option are required to support a certain minimal set of the subcommands for this option. Section 3 contains a complete list of the subcommands in this minimal set. In keeping with the Telnet protocol philosophy that an implementation should not have to be able to parse commands it does not implement, every subcommand of this option is either in the minimal set or is covered by one of the facility subcommands. An implementation must "negotiate" with its correspondent for permission to use subcommands not in the minimal set before using them. For details of this negotiation process see the section below on facility subcommands.

Most data entry terminals are used in a half duplex mode. (Although most DET's on the market can be used either as data entry terminals or as standard interactive terminals, we are only concerned here with their use as DET's.) When this option is used, it is suggested that the following Telnet options be refused: Echo, Remote Controlled Transmission and Echoing, and Suppress Go-Ahead. However, this option could be used to support a simple full duplex CRT based application using the basic cursor control functions provided here. For these cases, one or more of the above list of options might be required. (Support of sophisticated interactive calligraphic applications is beyond the scope of this option and should be done by another option or the Network Graphics Protocol.)

In RFC 728, it was noted that a synch sequence can cause undesired interactions between Telnet Control functions and the data stream. A synch sequence causes data but not control functions to be flushed. If a control function which has an effect on the data immediately following it is present in the data stream when a synch sequence occurs, the control function will have its effect not on the intended data but on the data immediately following the Data Mark. The following DET subcommands are susceptible to this pitfall:

CHAR INSERT
DATA TRANSMIT
FORMAT DATA

The undesired interactions are best avoided by the receiver

of the synch sequence deleting these subcommands and all data associated with them before continuing to process the control functions. This implies that the Data Mark should not occur in the middle of the data associated with these subcommands.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

Facility Subcommands

These four subcommands are used by the User and Server implementations to negotiate the subcommands and attributes of the terminal that may be utilized. This negotiation can be viewed as the terminal (User Host) indicating what facilities are provided and the Server Host (or application program) indicating what facilities are desired.

When Sent: A Server Telnet implementation using the DET option must send a facility subcommand requesting the use of a particular subcommand or terminal attribute not in the minimal implementation before the first use of that subcommand or attribute. The User Telnet implementation should respond as quickly as possible with its reply. Neither the User nor Server are required to negotiate one subcommand at a time. Also, a Telnet implementation responding to a facility subcommand is not required to give permission only for that subcommand. It may send a format map indicating all facilities of that class which it supports. However, a Telnet implementation requesting facilities must send a facility subcommand before its first use of the subcommand regardless of whether earlier negotiations have indicated the facility is provided. The facility cannot be used until a corresponding facility subcommand has been received. There are no other constraints on when the facility subcommands may be sent. In particular, it is not necessary for an application to know at the beginning of a session all facilities that it will use.

Action When Recieved: There are two possible actions that may be taken when a facility subcommand is received depending on whether the receiver is a requestor or a provider (User).

Requestor: When a facility subcommand is received by a requestor and it is in the state of Waiting for a Reply, it should go into the state of Not Waiting. It should then take the facility map it had sent and form the logical intersection with the facility map received. (For the Intensity attribute, one should take the minimum of the number received and the number requested.) The result indicates the facilities successfully negotiated. Note: if

the receiver is not in the Waiting for Reply state, then this is the provider case described next.

Provider: When a facility subcommand is received, it should send a facility subcommand with a facility map of the facilities it provides as soon as possible. It should then determine what new

facilities it is providing for the Requestor by forming the logical intersection of the facility map received and the one sent.

Note: Although in most cases the requestor will be the Server Host and the provider will be the User Host supporting the terminal, this distinction may not always be true.

Transmit Subcommands

There are two kinds of transmit subcommands: those used to request that data be sent to the requestor, and one to preface data sent to the requestor. The first kind allow the requestor to control when, from where and to some degree how much data is transmitted from the terminal. Their explanation is straightforward and may be found in Section 2.

Data may be sent from the terminal as a result of two events: the user of the terminal caused the transmission or in response to a transmit subcommand. Some programs may wish to know from where on the screen the transmission began. (This is reasonable, since the terminal user may move the cursor around considerably before transmitting.) Other programs may not need such information. The DATA TRANSMIT subcommand is provided in case this function is needed. When used this subcommand prefaces data coming from the terminal. The parameters <x> and <y> give the screen coordinates of the beginning of the transmission. <x> must be less than or equal to M-1 and <y> must be less than or equal to N-1. It is assumed that all data between this DATA TRANSMIT and the next one starts at the coordinates given by the first subcommand and continues filling each line thereafter according to the constraints of the screen and the format effectors in the data. Thus an intelligent or sloppy user-host DET implementation (depending on your point of view) need only include a DATA TRANSMIT subcommand when the new starting point is different from the last ending point.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

6. Sample Interaction

The nomenclature of RFC 726 will be used to describe this example. To quote that RFC:

"S:" is sent from serving host to using host.
"U:" is sent from using host to serving host.
"T:" is entered by the terminal user.
"P:" is printed on the terminal.

Text surrounded by square brackets([]) is commentary. Text surrounded by angle brackets (<>) is to be taken as a single unit. E.g, carriage return is <cr>, and the decimal value 27 is represented <27>.

We assume that the user has established the Telnet connection, logged on, and an application program has just been started either by the user directly or through a canned start up procedure. The presentation on the page is meant to merely group entities together and does not imply the position of message boundaries. One should assume that any part of the dialogue may be sent as one or many messages. The first action of the program or Telnet is to negotiate the DET option:

S: <IAC><DO><DET>

U: <IAC><WILL><DET>

S:<IAC><DO><OUTPUT PAGE SIZE> [First negotiate the screen size. In this case we are asking the user the size of the terminal. This could have been done before the DET option was negotiated.]

U:<IAC><WILL><NAOP>

U:<IAC><SB><NAOP><DR><25><IAC><SE>

S:<IAC><SB><NAOP><DS><0><IAC><SE>

S:<IAC><DO><OUTPUT LINE WIDTH>

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

U:<IAC><SB><NAOL><DR><80><IAC><SE> [Defines the screen to be 25 lines by 80 characters. The server may use this information when formatting the screen.]

S:<IAC><SB><NAOL><DS><0><IAC><SE>

S:<IAC><SB><DET><FORMAT FACILITIES>
<Repeat><Protection, 3 Levels
Intensity><IAC><SE> [Now set the terminal attributes.]

U:<IAC><SB><DET><FORMAT FACILITIES>
<Repeat, Blinking><Protection, 3
Levels Intensity><IAC><SE>

S:<IAC><SB><DET><ERASE SCREEN><IAC><SE> [Erase the screen and start sending the form.]

<IAC><SB><DET><FORMAT DATA>
<Protection=1, Intensity=1><0>
<5><IAC><SE>Name:

<IAC><SB><DET><MOVE CURSOR><0><1><IAC><SE>

<IAC><SB><DET><FORMAT DATA>
<Protection=1, Intensity=1><0>
<8><IAC><SE>Address:

<IAC><SB><MOVE CURSOR><0><4><IAC><SE>

<IAC><SB><DET><FORMAT DATA>
<Protection=1, Intensity=1><0>
<17><IAC><SE>Telephone number:

<IAC><SB><DET><MOVE CURSOR><32><4><IAC><SE>

<IAC><SB><DET><FORMAT DATA>
<Protection=1, Intensity=1><0>
<24><IAC><SE>Social Security Number:

<IAC><SB><DET><FORMAT DATA>
<Protection=1, Intensity=7>
<0><11><IAC><SE> [Establish a field that doesn't display what is typed into it.]

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

<IAC><SB><DET><MOVE CURSOR><32><5><IAC><SE>

<IAC><SB><DET><FORMAT FACILITIES>
<Blinking><0><IAC><SE> [Get permission to use Blinking Attribute.]

U:<IAC><SB><DET><FORMAT FACILITIES>
<Repeat, Blinking><Protection,
3 Levels Intensity><IAC><SE>

S:<IAC><SB><DET><FORMAT DATA>
<Blinking=1, Protection=1,
Intensity=1><0><29><IAC><SE>

Your SSN will not be printed.

<IAC><SB><DET><HOME><IAC><SE>
<IAC><GA>

The previous exchange has placed a form on the screen that looks like:

Name:
Address:
Telephone Number: Social Security Number:
"Your SSN will not be printed."

where the quoted string is blinking.

The terminal user is now free to fill in the form provided. He positions the cursor at the beginning of the first field (this usually is done by hitting the tab key) and begins typing. We do not show this interaction since it does not generate any interaction with the User Telnet program or the network. After the terminal user has completed filling in the form, he strikes the transmit key to send the unprotected part of the form, but first the User Telnet program negotiates the Byte Macro Option to condense the Field Separator subcommand:

U:<IAC><DO><BM> [Negotiate Byte Macro Option.]

S:<IAC><WILL><BM> [Define decimal 166 to be the Field Separator subcommand (see Appendix 3)]

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

U:<IAC><SB><BM><DEFINE>
<166><6><IAC SB DET FIELD
SEPARATOR IAC SE><IAC><SE>

S:<IAC><SB><BM><ACCEPT><166><IAC><SE> [The server accepts the macro.]

U:<IAC><SB><DET><DATA TRANSMIT><0><6><IAC><SE>
John Doe <166> 1515 Elm St., Urbana, Il 61801
<166> 217-333-9999 <166> 123-45-6789 <166>

S:<IAC><SB><DET><ERASE SCREEN><IAC><SE>
Thank you.

And so on.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

Appendix 1 - Subcommands, opcodes and syntax

1	EDIT FACILITIES	<Facilty map>
2	ERASE FACILITIES	<Facility map>
3	TRANSMIT FACILITIES	<Facility map>
4	FORMAT FACILITIES	<Facility map 1> <Facility map 2>
5	MOVE CURSOR	<x> <y>
6	SKIP TO LINE	<y>
7	SKIP TO CHAR	<x>
8	UP	
9	DOWN	
10	LEFT	
11	RIGHT	
12	HOME	
13	LINE INSERT	
14	LINE DELETE	
15	CHAR INSERT	
16	CHAR DELETE	
17	READ CURSOR	
18	CURSOR POSITION	<x><y>
19	REVERSE TAB	
20	TRANSMIT SCREEN	
21	TRANSMIT UNPROTECTED	
22	TRANSMIT LINE	
23	TRANSMIT FIELD	
24	TRANSMIT REST OF SCREEN	
25	TRANSMIT REST OF LINE	
26	TRANSMIT REST OF FIELD	
27	TRANSMIT MODIFIED	
28	DATA TRANSMIT <x><y>	
29	ERASE SCREEN	
30	ERASE LINE	
31	ERASE FIELD	
32	ERASE REST OF SCREEN	
33	ERASE REST OF LINE	
34	ERASE REST OF FIELD	
35	ERASE UNPROTECTED	
36	FORMAT DATA <format map>	
37	REPEAT <count><char>	
38	SUPPRESS PROTECTION <negotiation>	
39	FIELD SEPARATOR	
40	FN <code>	
41	ERROR <cmd><error code>	

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

Appendix 2 - Error Codes

1	Facility not previously negotiated.
2	Illegal subcommand code.
3	Cursor Address Out of Bounds.
4	Undefined EN value.
5	Can't negotiate acceptable line width.
6	Can't negotiate acceptable page length.
7	Illegal parameter in subcommand.
8	Syntax error in parsing subcommand.
9	Too many parameters in subcommand.
10	Too few parameters in subcommand.
11	Undefined parameter value
12	Unsupported combination of Format Attributes

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

Appendix 3 - Use of the Byte Macro Option

One of the major drawbacks of the DET option is that because the functions are encoded as Telnet option subnegotiations a fairly high overhead is incurred. A function like Character Insert which is encoded as a single byte in most terminals requires six bytes in the DET option. Originally the only other solution that would have accomplished the same transparency that the use of subcommands provides would have been to define additional Telnet control functions. However, since this would entail modification of the Telnet protocol itself, it was felt that this was not a wise solution. Since then the Telnet Byte Macro Option (RFC 729) has been defined. This option allows the user and server Telnets to map an arbitrary character string into a single byte which is then transferred over the net. Thus the Byte Macro Option provides the means for implementations to avoid the overhead for heavily used subcommands. The rest of this appendix suggests how the Byte Macro Option should be applied to the DET option.

In keeping with the specification of the Byte Macro Option, macro bytes will be chosen from the range 128 to 239. For the DET option, it is suggested that macro bytes be chosen by adding the subcommand code to 128. In addition, an unofficial DET subcommand might be defined indicating that each side was willing to support macro bytes for all subcommands (but not necessarily support all of the subcommands themselves) according to this algorithm. This subcommand would be:

IAC SB DET DET-MACRO <negotiation> IAC SE subcommand code: 254

where <negotiation> may have the values of the Telnet option negotiation:

251	WILL
252	WONT
253	DO
254	DONT

This subcommand is sent by a Telnet implementation to indicate its willingness to adopt byte macros for all of the DET subcommands according to the following algorithm:

The macro byte for subcommand i will be i+128 and will represent the following string for parameterless subcommands:

IAC SB DET <subcommand code> IAC SE

and the following string for subcommands with parameters:

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

IAC SB DET <subcommand code>

The default setting for this subcommand is that the macros are not in effect, in other words,

IAC SB DET DET-MACRO WONT IAC SE
IAC SB DET DET-MACRO DONT IAC SE

Negotiation of this subcommand follows the same rules as negotiations of the Telnet options.

NWG/RFC# 732 DAY 13-Sep-77 18:38 41762
Data Entry Terminal Option

References

1. ADM-1 Interactive Display Terminal Operator's Handbook Lear-Siegler, Inc. 7410-31.

2. ADM-Interactive Display Terminal Operator's Handbook Lear-Siegler, Inc. EID, 1974.

3. Burroughs TD 700/800 Reference Manual, Burroughs Corp., 1973

4. Burroughs TD 820 Reference Manual, Burroughs Corp. 1975.

5. CC-40 Communications Station: General Information Manual. Computer Communication, Inc. Pub. No. MI-1100. 1974.

6. Crocker, David. "Telnet Byte Macro Option," RFC 729, 1977.

7. Data Entry Virtual Terminal Protocol for Euronet, DRAFT, 1977.

8. Day, John. "A Minor Pitfall in the Telnet Protocol," RFC 728, 1977.

9. Hazeltine 2000 Desk Top Display Operating Instructions. Hazeltine IB-1866A, 1870.

10. How to Use the Consul 980: A Terminal Operator's Guide and Interface Manual. Applied Digital Data Systems, Inc. 98-3000.

11. How to Use the Consul 520: A Terminal Operator's Guide and Interface Manual. Applied Digital Data Systems, Inc. 52-3000.

12. Honeywell 7700 Series Visual Information Projection (VIP) Systems: Preliminary Edition. 1973.

13. An Introduction to the IBM 3270 Information Display System. IBM GA27-2739-4. 1973.

14. Naffah, N. "Protocole Appareil Virtuel type Ecran" Reseau Cyclades. TER 536. 1976.

15. Postel, Jon and Crocker, David. "Remote Controlled Transmission and Echoing Telnet Option", RFC 726 NIC 39237, Mar. 1977.

16. Schicker, Peter. "Virtual Terminal Protocol (Proposal 2). INWG Protocol Note #32., 1976.

17. UNISCOPE Display Terminal : Programmer Reference . Sperry- Univac UP-7807 Rev. 2, 1975.

18. Universal Terminal System 400: System Description. Sperry- Univac UP-8357, 1976.

19. Walden, David C. "Telnet Output Line Width Option." NIC # 20196, 1973, also in ARPANET Protocol Handbook, 1976.

20. Walden, David C. "Telnet Output Page Size" NIC # 20197, 1973, also in ARPANET Protocol Handbook, 1976.

NWG/RFC# 736 MRC 31-OCT-77 23:28 42213
Telnet SUPDUP Option

Network Working Group
Request for Comments 736
NIC 42213

Mark Crispin
SU-AI
31 October 1977

TELNET SUPDUP Option

1. Command name and code.

SUPDUP 21

2. Command meanings.

IAC WILL SUPDUP

The sender of this command REQUESTS permission to, or confirms that it will, use the SUPDUP display protocol

IAC WON'T SUPDUP

The sender of this command REFUSES to use the SUPDUP protocol.

IAC DO SUPDUP

The sender of this command REQUESTS that the receiver use, or grants the receiver permission to use, the SUPDUP protocol.

IAC DON'T

The sender of this command DEMANDS that the receiver not use the SUPDUP protocol.

3. Default.

WON'T SUPDUP

DON'T SUPDUP

i.e., the SUPDUP display protocol is not in use.

NWG/RFC# 736 MRC 31-OCT-77 23:28 42213
Telnet SUPDUP Option

4. Motivation for the option.

Since the publication of RFC 734, I have been requested to design an option to the TELNET protocol to provide for SUPDUP service. This option allows a host to provide SUPDUP service on the normal TELNET socket (27 octal) instead of 137 (octal) which is the normal SUPDUP ICP socket.

5. Description of the option.

A user TELNET program which wishes to use the SUPDUP display protocol instead of the NVT terminal service should send an IAC DO SUPDUP. If the server is willing to use the SUPDUP display protocol, it should respond with IAC WILL SUPDUP; otherwise it should refuse with IAC WONT SUPDUP.

For hosts which normally provide SUPDUP terminal services, the server can send IAC WILL SUPDUP upon ICP which the user may then accept or refuse.

If the SUPDUP option is in effect, no further TELNET negotiations are allowed. They are meaningless, since SUPDUP has its own facilities to perform the functions that are needed. Hence, octal 377 will become an ordinary transmitted character (in this case an invalid %TD code) instead of an IAC.

Following the mutual acceptance of the SUPDUP option, the SUPDUP negotiation proceeds as described in RFC 734.

NWG/RFC 749
Network Working Group
Request for Comments 749
NIC 45499

BSG 26-Sep-78 13:13 45499
Bernard Greenberg
MIT-Multics
18 September 1978

Telnet SUPDUP-OUTPUT Option

1. Command name and code.

SUPDUP-OUTPUT 22

2. Command meanings.

IAC WILL SUPDUP-OUTPUT

The sender of this command REQUESTS permission to transmit SUPDUP-OUTPUT format messages over the TELNET connection.

IAC WON'T SUPDUP-OUTPUT

The sender of this command STATES that he will no longer send SUPDUP-OUTPUT format messages over the TELNET connection.

IAC DO SUPDUP-OUTPUT

The sender of this command grants the receiver permission to send SUPDUP-OUTPUT format messages over the TELNET connection.

IAC DON'T SUPDUP-OUTPUT

The sender of this command DEMANDS that the receiver not send SUPDUP-OUTPUT format messages over the TELNET connection.

IAC SB SUPDUP-OUTPUT 1 <terminal-parameters> IAC SE

The sender of this command (which must be the TELNET user process) is supplying information describing the capabilities of the user process' terminal.

IAC SB SUPDUP-OUTPUT 2 n TD1 TD2 .. TDn SCx SCy IAC SE

The sender of this command, which must be the TELNET server process, is sending explicit screen control information to be carried out by the user TELNET process.

3. Default.

WON'T SUPDUP-OUTPUT

DON'T SUPDUP-OUTPUT

i.e., the SUPDUP-OUTPUT format messages may not be transmitted.

4. Motivation for the option.

The SUPDUP-OUTPUT protocol provides a means to access the virtual display support provided by the SUPDUP protocol (see RFC 734) within the context of a standard TELNET connection. This allows occasional display-oriented programs at non-display-oriented servers to take advantage of the standardized display support provided by SUPDUP. This cannot be done with the standard SUPDUP protocol or the TELNET SUPDUP option (RFC 736), for they both require that all communication after the negotiation to use SUPDUP has been completed proceed according to the protocol of RFC 734. This places upon the server total responsibility for screen management for the duration of the connection, which, by hypothesis, the non-display oriented server is not willing to accept.

User TELNET programs at display-oriented user hosts provide local screen management by mapping the NVT commands of TELNET into local screen management commands; often, this involves scrolling, end-of-page processing, line clearing etc. The SUPDUP-OUTPUT option allows a display-oriented application program at the server side to take over screen management explicitly, via the SUPDUP display control repertoire. TELNET remains in effect throughout. The IAC IP and other TELNET commands are still valid.

By means of the SUPDUP-OUTPUT option, display-oriented programs can run on the server host, and control the user host's screen explicitly. The user TELNET process sends a description of the user terminal (as specified in RFC 734) to the server TELNET process as a subnegiotiation block when the SUPDUP-OUTPUT negotiation has been successfully completed. The server TELNET process sends explicit screen control commands via subnegotiation blocks to the user TELNET process.

5. Description of the option.

The SUPDUP-OUTPUT protocol may only be initiated by the server TELNET process. A server TELNET process wishing to take advantage of the SUPDUP-OUTPUT protocol will initiate a negotiation for it by sending IAC WILL SUPDUP-OUTPUT. The user TELNET process must accept or refuse the offer by sending IAC DO SUPDUP-OUTPUT or IAC DON'T SUPDUP-OUTPUT.

If the user TELNET process agrees to support the SUPDUP-OUTPUT option, it must follow the sending of IAC DO SUPDUP-OUTPUT immediately with a description of the user's terminal. This information is described in RFC 734 as the "terminal parameters." It is to be sent as a series of six-bit bytes, one byte per eight bit

TELNET data byte. These words may or may not contain the optional line speed and graphics capabilities parameters described by RFC 747; the first six bytes specify the count of 36-bit words to follow as described by RFC 734.

The terminal parameter block will be sent as a subnegotiation of the SUPDUP-OUTPUT option:

IAC SB SUPDUP-OUTPUT 1 byte1 byte2 ... byteN IAC SE

The byte of "1" is a command code, for compatibility with future extensions. Upon receipt of the terminal parameter block from the user TELNET process, the server TELNET process may send SUPDUP-OUTPUT blocks as described below.

The server TELNET process can specify explicit control of the user host's screen by the sending of subnegotiation blocks of the SUPDUP-OUTPUT option. The format of such a block, as seen in eight-bit TELNET data bytes, is:

IAC SB SUPDUP-OUTPUT 2 N TD1 TD2 TD3 ... TDn SCx SCy IAC SE

The byte of "2" is a command code, for compatibility with future extensions. The TDm bytes are the "%TDCODEs" and printing characters of SUPDUP output of RFC 734. N is a byte containing a count of the number of TDm's in this transmission. N may be zero, and may not be greater than 254 (decimal). SCx and SCy are two bytes specifying the anticipated horizontal and vertical (respectively) coordinates of the cursor of the user host's screen after the latter has interpreted all the %TDCODEs in this transmission.

The motivation for the SCx SCy screen position specification is to allow hosts running the ITS operating system, which will transmit the TDCODEs directly into the local output system, to assert the "main program level" screen position without any interpretation of the transmitted TDCODE sequence by the user TELNET program.

The user TELNET process must manage the position of the local cursor with respect to standard TELNET NVT commands and output, and SUPDUP OUTPUT transmissions. The user TELNET process may assume that the server TELNET process is managing both NVT and SUPDUP-OUTPUT output in an integrated way.

The SUPDUP-OUTPUT option makes no statement about how input is sent; this may be negotiated via other options. By default, NVT input will be used. The user-to-server screen management commands of RFC 734 are NOT implicitly handled by IAC WILL SUPDUP-OUTPUT.

In the absence of the transmission of SUPDUP-OUTPUT subnegotiation blocks, a TELNET connection operating with the SUPDUP-OUTPUT option in effect is indistinguishable from a normal TELNET connection. Thus IAC WON'T SUPDUP-OUTPUT is highly optional, and if received by the user TELNET process, should only be used to cause a diagnostic if SUPDUP-OUTPUT subnegotiation blocks are subsequently received. If received, the user TELNET process should respond with IAC DON'T SUPDUP OUTPUT.

Because of the optional nature of IAC WON'T SUPDUP-OUTPUT, the user TELNET process should be prepared to send the terminal parameter subnegotiation block each time IAC WILL SUPDUP-OUTPUT is received, i.e., even if the user TELNET process believes SUPDUP-OUTPUT to be in effect.

The %TDORS (output reset) code may not be sent in a SUPDUP-OUTPUT transmission. The user TELNET program may assume that no byte in a subnegotiation block will be 255 (decimal).

No multi-byte TDCODE sequence (e.g., %TDMOV, %TDILP) may be split across SUPDUP-OUTPUT subnegotiation blocks.

References:

Crispin, Mark:

"SUPDUP Display Protocol", RFC 734, 7 October 1977, NIC 44213.

Crispin, Mark:

"TELNET SUPDUP Option", RFC 736, 31 October 1977, NIC 44213.

Crispin, Mark:

"Recent Extensions to the SUPDUP Protocol", RFC 747, 21 March 1978, NIC 44015.

Network Working Group
Request for Comments: 779

E. Killian
LLL
April 1981

TELNET SEND-LOCATION Option

1. Command name and code.

SEND-LOCATION 23

2. Command meanings.

IAC WILL SEND-LOCATION

The sender REQUESTS or AGREES to use the SEND-LOCATION option to send the user's location.

IAC WON'T SEND-LOCATION

The sender REFUSES to use the SEND-LOCATION option.

IAC DO SEND-LOCATION

The sender REQUESTS that, or AGREES to have, the other side use SEND-LOCATION commands send the user's location.

IAC DON'T SEND-LOCATION

The sender DEMANDS the other side not use the SEND-LOCATION option.

IAC SB SEND-LOCATION <location> IAC SE

The sender specifies the user's location to the other side via a SEND-LOCATION subnegotiation. <location> is a sequence of ASCII printable characters; it is terminated by the IAC SE.

3. Default.

WON'T SEND-LOCATION

DON'T SEND-LOCATION

4. Motivation for the option.

Many network sites now provide a listing of the users currently logged in giving their names and locations (see the NAME/FINGER protocol, RFC 742). The location is useful for physically locating the user if he or she is nearby, or for calling them (a nearby phone number is often included). However, for users logged in via the network, the location printed is often no more than the originating site name. This TELNET option allows the user's TELNET program to send the user's location to the server TELNET so that it can be displayed in addition to the site name. This functionality is already present in the SUPDUP protocol (RFC 734).

5. Description of the option.

When the user TELNET program knows the user's location, it should offer to transmit this information to the server TELNET by sending IAC WILL SEND-LOCATION. If the server's system is able to make use of this information (as can the ITS sites), then the server will reply with IAC DO SEND-LOCATION. The user TELNET is then free to send the location in a subnegotiation at any time.

Network Working Group
Request for Comments: 930
Supersedes: RFC 884

Marvin Solomon
Edward Wimmers
University of Wisconsin - Madison
January 1985

TELNET TERMINAL TYPE OPTION

Status of This Memo

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that exchange terminal type information within the Telnet protocol are expected to adopt and implement this standard. Distribution of this memo is unlimited.

This standard supersedes RFC 884. The only change is to specify that the TERMINAL-TYPE IS sub-negotiation should be sent only in response to the TERMINAL-TYPE SEND sub-negotiation. See below for further explanation.

1. Command Name and Code

   TERMINAL-TYPE 24

2. Command Meanings

   IAC WILL TERMINAL-TYPE

   Sender is willing to send terminal type information in a subsequent sub-negotiation

   IAC WON'T TERMINAL-TYPE

   Sender refuses to send terminal type information

   IAC DO TERMINAL-TYPE

   Sender is willing to receive terminal type information in a subsequent sub-negotiation

   IAC DON'T TERMINAL-TYPE

   Sender refuses to accept terminal type information

   IAC SB TERMINAL-TYPE SEND IAC SE

   Sender requests receiver to transmit his (the receiver's) terminal type. The code for SEND is 1. (See below.)

IAC SB TERMINAL-TYPE IS ... IAC SE

Sender is stating the name of his terminal type. The code for IS is 0. (See below.)

3. Default

WON'T TERMINAL-TYPE

Terminal type information will not be exchanged.

DON'T TERMINAL-TYPE

Terminal type information will not be exchanged.

4. Motivation for the Option

This option allows a telnet server to determine the type of terminal connected to a user telnet program. The transmission of such information does not immediately imply any change of processing. However, the information may be passed to a process, which may alter the data it sends to suit the particular characteristics of the terminal. For example, some operating systems have a terminal driver that accepts a code indicating the type of terminal being driven. Using the TERMINAL TYPE and BINARY options, a telnet server program on such a system could arrange to have terminals driven as if they were directly connected, including such special functions as cursor addressing, multiple colors, etc., not included in the Network Virtual Terminal specification. This option fits into the normal structure of TELNET options by deferring the actual transfer of status information to the SB command.

5. Description of the Option

WILL and DO are used only to obtain and grant permission for future discussion. The actual exchange of status information occurs within option subcommands (IAC SB TERMINAL-TYPE...).

Once the two hosts have exchanged a WILL and a DO, the sender of the DO TERMINAL-TYPE is free to request type information. Only the sender of the DO may send requests (IAC SB TERMINAL-TYPE SEND IAC SE) and only the sender of the WILL may transmit actual type information (within an IAC SB TERMINAL-TYPE IS ... IAC SE command). Terminal type information may not be sent spontaneously, but only in response to a request.

The terminal type information is an NVT ASCII string. Within this

string, upper and lower case are considered equivalent. The complete list of valid terminal type names can be found in the latest "Assigned Numbers" RFC.

The following is an example of use of the option:

Host1: IAC DO TERMINAL-TYPE

Host2: IAC WILL TERMINAL-TYPE

(Host1 is now free to request status information at any time.)

Host1: IAC SB TERMINAL-TYPE SEND IAC SE

Host2: IAC SB TERMINAL-TYPE IS IBM-3278-2 IAC SE

6. Implementation Suggestions

The "terminal type" information may be any NVT ASCII string meaningful to both ends of the negotiation. The list of terminal type names in "Assigned Numbers" is intended to minimize confusion caused by alternative "spellings" of the terminal type. For example, confusion would arise if one party were to call a terminal "IBM3278-2" while the other called it "IBM-3278/2". There is no negative acknowledgement for a terminal type that is not understood, but certain other options (such as switching to BINARY mode) may be refused if a valid terminal type name has not been specified. In some cases, a particular terminal may be known by more than one name, for example a specific type and a more generic type. In such cases, the sender of the TERMINAL-TYPE IS command should reply to successive TERMINAL-TYPE SEND commands with the various names, from most to least specific. In this way, a telnet server that does not understand the first response can prompt for alternatives. However, it should cease sending TERMINAL-TYPE SEND commands after receiving the same response two consecutive times. Similarly, a sender should indicate it has sent all available names by repeating the last one sent. Note that TERMINAL-TYPE IS must only be sent in response to a request (TERMINAL-TYPE SEND), because a host that sent TERMINAL-TYPE IS and then received TERMINAL-TYPE SEND couldn't determine whether the other host was requesting a second option or the TERMINAL-TYPE SEND and the TERMINAL-TYPE IS crossed in midstream.

The type "UNKNOWN" should be used if the type of the terminal is unknown or unlikely to be recognized by the other party.

The complete and up-to-date list of terminal type names will be maintained in the "Assigned Numbers". The maximum length of a terminal type name is 40 characters.

Network Working Group
Request for Comments: 885

J. Postel
ISI
December 1983

TELNET END OF RECORD OPTION

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that need to mark record boundaries within Telnet protocol data are expected to adopt and implement this standard.

1. Command Name and Code

END-OF-RECORD 25

2. Command Meanings

IAC WILL END-OF-RECORD

The sender of this command requests permission to begin transmission of the Telnet END-OF-RECORD (EOR) code when transmitting data characters, or the sender of this command confirms it will now begin transmission of EORs with transmitted data characters.

IAC WON'T END-OF-RECORD

The sender of this command demands to stop transmitting, or to refuses to begin transmitting, the EOR code when transmitting data characters.

IAC DO END-OF-RECORD

The sender of this command requests that the sender of data start transmitting the EOR code when transmitting data, or the sender of this command confirms that the sender of data is expected to transmit EORs.

IAC DON'T END-OF-RECORD

The sender of this command demands that the receiver of the command stop or not start transmitting EORs when transmitting data.

3. Default

WON'T END-OF-RECORD

DON'T END-OF-RECORD

END-OF-RECORD is not transmitted.

4. Motivation for the Option

Many interactive systems use one (or more) of the normal data characters to indicate the end of an effective unit of data (i.e., a record), for example, carriage-return (or line-feed, or escape). Some systems, however, have some special means of indicating the end of an effective data unit, for example, a special key. This Telnet option provides a means of communicating the end of data unit in a standard way.

5. Description of the Option

When the END-OF-RECORD option is in effect on the connection between a sender of data and the receiver of the data, the sender transmits EORs.

It seems probable that the parties to the Telnet connection will transmit EORs in both directions of the Telnet connection if EORs are used at all; however, the use of EORs must be negotiated independently for each direction.

When the END-OF-RECORD option is not in effect, the IAC EOR command should be treated as a NOP if received, although IAC EOR should not normally be sent in this mode.

6. Implementation Considerations

As the EOR code indicates the end of an effective data unit, Telnet should attempt to send the data up to and including the EOR code together to promote communication efficiency.

The end of record is indicated by the IAC EOR 2-octet sequence. The code for EOR is 239 (decimal).

Network Working Group
Request for Comments: 927

Brian A. Anderson
BBN
December 1984

TACACS User Identification Telnet Option

Status of this Memo

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Introduction

The following is the description of a TELNET option designed to facilitate double login avoidance. It is intended primarily for TAC connections to target hosts on behalf of TAC users, but it can be used between any two consenting hosts. For example, all hosts at one site (e.g., BBN) can use this option to avoid double login when TELNETing to one another.

1. Command name and code

   TUID 26

2. Command Meanings

   IAC WILL TUID

   The sender (the TELNET user) proposes to authenticate the user and send the identifing UUID; or, the sender (the TELNET user) agrees to authenticate the user on whose behalf the connection is initiated.

   IAC WON'T TUID

   The sender (the TELNET user) refuses to authenticate the user on whose behalf the connection is initiated.

   IAC DO TUID

   The sender (the TELNET server) proposes that the recipient (the TELNET user) authenticate the user and send the identifing UUID; or, the sender (the TELNET server) agrees to accept the recipient's (the TELNET user's) authentication of the user identified by his UUID.

IAC DON'T TUID

The sender (the TELNET server) refuses to accept the recipient's (the TELNET user) authentication of the user.

IAC SB TUID <uuid> IAC SE

The sender (the TELNET user) sends the UUID <uuid> of the user on whose behalf the connection is established to the host to which he is connected. The <uuid> is a 32 bit binary number.

3. Default

WON'T TUID

A TELNET user host (the initiator of a TELNET connection) not implementing or using the TUID option will reply WON'T TUID to a DO TUID.

DON'T TUID

A TELNET server host (the recipient of a TELNET connection) not implementing or using the TUID option reply DON'T TUID to a WILL TUID.

4. Motivation for the Option

Under TACACS (the TAC Access Control System) a user must be authenticated (give a correct name/password pair) to a TAC before he can connect to a host via the TAC. To avoid a second authentication by the target host, the TAC can pass along the user's proven identity (his UUID) to the that host. Hosts may accept the TAC's authentication of the user or not, at their option.

The same option can be used between any pair of cooperating hosts for the purpose of double login avoidance.

5. Description for the Option

At the time that a host establishes a TELNET connection for a user to another host, if the latter supports the TUID option and wants to receive the user's UUID, it sends an IAC DO TUID to the the user's host. If the user's host supports the TUID option and wants to authenticate the user by sending the user's UUID, it responds IAC WILL TUID; otherwise it responds with IAC WON'T TUID. If both the user and server TELNETs agree, the user TELNET will then send the UUID to the server TELNET by sub-negotiation.

6. Examples

There are two possible negotiations that result in the double login avoidance authentication of a user. Both the server and the user TELNET support the TUID option.

S = Server, U = User

Case 1:

```
S-> IAC DO TUID
U-> IAC WILL TUID
U-> IAC SB TUID <32-bit UUID> IAC SE
```

Case 2:

```
U-> IAC WILL TUID
S-> IAC DO TUID
U-> IAC SB TUID <32-bit UUID> IAC SE
```

There are also two possible negoitiations that do not result in the authentication of a user. In the first example the server supports TUID and the user TELNET doesn't. In the second example the user TELNET supports TUID but the server TELNET doesn't.

S = Server, U = User

Case 3:

```
S-> IAC DO TUID
U-> IAC WONT TUID
```

Case 4:

```
U-> IAC WILL TUID
S-> IAC DONT TUID
```

The TUID is transmitted with the subnegotiation command. For example, if the UUID had the value 1 the following string of octets would be transmitted:

```
IAC SB TUID 0 0 0 1 IAC SE
```

If the UUID had the value 255 the following string of octets would be transmitted:

```
IAC SB TUID 0 0 0 IAC IAC IAC SE
```

If the UUID had the value of all ones the following string of octets would be transmitted:

IAC SB TUID IAC IAC IAC IAC IAC IAC IAC IAC IAC SE

Network Working Group
Request for Comments: 933

S. Silverman
MITRE-Washington
January 1985

OUTPUT MARKING TELNET OPTION

Status of this Memo

This RFC proposes a new option for Telnet for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Overview

This proposed option would allow a Server-Telnet to send a banner to a User-Telnet so that this banner would be displayed on the workstation screen independently of the application software running in the Server-Telnet.

1. Command Name and Code

OUTMRK 27

2. Command Meanings

IAC WILL OUTMRK

Sender is willing to send output marking information in a subsequent sub-negotiation.

IAC WON'T OUTMRK

Sender refuses to send output marking information.

IAC DO OUTMRK

Sender is willing to receive output marking information in a subsequent sub-negotiation.

IAC DON'T OUTMRK

Sender refuses to accept output marking information.

IAC SB OUTMRK CNTL data IAC SE

The sender requests receiver to use the data in this subnegotiation as a marking for the normally transmitted Telnet data until further notice. The CNTL octet indicates the position of the marking (see below).

IAC SB OUTMRK ACK IAC SE

The sender acknowledges the data and agrees to use it to perform output marking (see below).

IAC SB OUTMRK NAK IAC SE

The sender objects to using the data to perform output marking (see below).

3. Default

WON'T OUTMRK

Output marking information will not be exchanged.

DON'T OUTMRK

Output marking information will not be exchanged.

4. Motivation for the Option

The security architecture of some military systems identifies a security level with each Telnet connection. There is a corresponding need to display a security banner on visual display devices. (Reference: Department of Defense Trusted Computer System Evaluation Criteria, Section 3.1.1.3.2.3, Labeling Human-Readable Output.)

The output marking is currently done by transmitting the banner as data within each screen of data. It would be more efficient to transmit the data once with instructions and have User-Telnet maintain the banner automatically without any additional Server-Telnet action. This frees Server-Telnet from needing to know the output device page size.

Under this proposal Server-Telnet would send an option sequence with the command, a control flag, and the banner to be used. While current systems use the top of the screen, it is conceivable other systems would want to put the banner at the bottom or perhaps even the side of the screen. This is the reason for the control flag.

5. Description of the Option

Either side of the session can initiate the option; however, normally it will be the server side that initiates the request to perform output marking. Either the Server-Telnet sends "WILL OUTMRK" or the User-Telnet sends a "DO OUTMRK". The party receiving the initial

"WILL" (or "DO") would respond with "DO" (or "WILL") to accept the option. Then Server-Telnet responds with the marking data. The format of this is:

"IAC SB OUTMRK CNTL data IAC SE"

CNTL is the Control Flag described below, the data is in ASCII.

If this is satisfactory, User-Telnet responds:

"IAC SB OUTMRK ACK IAC SE"

ACK is the ASCII ACK (6).

From this point, User-Telnet will have to translate any command which uses cursor controls so that the application data is mapped to the application part of the screen.

If the data passed in the subnegotiation field is unacceptable to User-Telnet, then it responds with:

"IAC SB OUTMRK NAK IAC SE"

NAK is the ASCII NAK (21).

It is now up to Server-Telnet to start the sequence over again and use "more acceptable" data (or possibly take other action such as connection termination).

To terminate output marking, Server-Telnet transmits "WON'T OUTMRK".

If necessary, User-Telnet would notify Server-Telnet about the new effective page size. User-Telnet would then map the output data to the allowed usable space on the screen.

User-Telnet may request OUTMRK data or initiate setup of this convention at anytime by transmitting "DO OUTMRK". If a WILL, DO OUTMRK exchange is not followed by the OUTMRK subnegotiation of the marking data, the User-Telnet may terminate the output marking option by sending a "DON'T OUTMRK".

Control Flag

The CNTL flag is defined as:

D = Default, the placement of the markings is up to User-Telnet. This is the expected mode for most interactions.

T = Top, this banner is to be used as the top of the screen. If multiple output markings are desired, then T and B (or R & L) are to be used.

B = Bottom, this banner is to be used at the bottom of the screen.

L = Left, markings on the left. (The precise meaning of this is to be defined.)

R = Right, marking on right. (The precise meaning of this is to be defined.)

Banner Data

The use of Carriage Return and Line Feed (CRLF) will be interpreted as a end of line in the marking banner text. If the user wants a multiline banner, CRLF will be used between each line. No CRLF is needed at the end of the marking data.

To use multiple banners, all of the banners will be included in one subnegotiation command of the form:

"IAC SB OUTMRK CNTL data GS CNTL data IAC SE"

where GS is the ASCII Group Separator (29) character.

User-Telnet will be responsible for positioning the marking banner data on the screen.

Network Working Group
Request for Comments: 861

J. Postel
J. Reynolds
ISI
May 1983

Obsoletes: NIC 16239

TELNET EXTENDED OPTIONS - LIST OPTION

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

1. Command Name and Code

EXTENDED-OPTIONS-LIST (EXOPL) 255

2. Command Meanings

IAC DO EXOPL

The sender of this command REQUESTS that the receiver of this command begin negotiating, or confirms that the receiver of this command is expected to begin negotiating, TELNET options which are on the "Extended Options List".

IAC WILL EXOPL

The sender of this command requests permission to begin negotiating, or confirms that it will begin negotiating, TELNET options which are on the "Extended Options List".

IAC WON'T EXOPL

The sender of this command REFUSES to negotiate, or to continue negotiating, options on the "Extended Options List".

IAC DON'T EXOPL

The sender of this command DEMANDS that the receiver conduct no further negotiation of options on the "Extended Options List".

IAC SB EXOPL <subcommand>

The subcommand contains information required for the negotiation of an option of the "Extended Options List". The format of the subcommand is discussed in section 5 below.

3. Default

WON'T EXOPL, DON'T EXOPL

Negotiation of options on the "Extended Options List" is not permitted.

4. Motivation for the Option

Eventually, a 257th TELNET option will be needed. This option will extend the option list for another 256 options in a manner which is easy to implement. The option is proposed now, rather than later (probably much later), in order to reserve the option number (255).

5. An Abstract Description of the Option

The EXOPL option has five subcommand codes: WILL, WON'T, DO, DON'T, and SB. They have exactly the same meanings as the TELNET commands with the same names, and are used in exactly the same way. For consistency, these subcommand codes will have the same values as the TELNET command codes (250-254). Thus, the format for negotiating a specific option on the "Extended Options List" (once both parties have agreed to use it) is:

IAC SB EXOPL DO/DON'T/WILL/WON'T/<option code> IAC SE

Once both sides have agreed to use the specific option specified by <option code>, subnegotiation may be required. In this case the format to be used is:

IAC SB EXOPL SB <option code> <parameters> SE IAC SE

Network Working Group
Request for Comments: 959

Obsoletes RFC: 765 (IEN 149)

J. Postel
J. Reynolds
ISI

October 1985

FILE TRANSFER PROTOCOL (FTP)

Status of this Memo

This memo is the official specification of the File Transfer Protocol (FTP). Distribution of this memo is unlimited.

The following new optional commands are included in this edition of the specification:

CDUP (Change to Parent Directory), SMNT (Structure Mount), STOU (Store Unique), RMD (Remove Directory), MKD (Make Directory), PWD (Print Directory), and SYST (System).

Note that this specification is compatible with the previous edition.

1. INTRODUCTION

The objectives of FTP are 1) to promote sharing of files (computer programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among hosts, and 4) to transfer data reliably and efficiently. FTP, though usable directly by a user at a terminal, is designed mainly for use by programs.

The attempt in this specification is to satisfy the diverse needs of users of maxi-hosts, mini-hosts, personal workstations, and TACs, with a simple, and easily implemented protocol design.

This paper assumes knowledge of the Transmission Control Protocol (TCP) [2] and the Telnet Protocol [3]. These documents are contained in the ARPA-Internet protocol handbook [1].

2. OVERVIEW

In this section, the history, the terminology, and the FTP model are discussed. The terms defined in this section are only those that have special significance in FTP. Some of the terminology is very specific to the FTP model; some readers may wish to turn to the section on the FTP model while reviewing the terminology.

2.1. HISTORY

FTP has had a long evolution over the years. Appendix III is a chronological compilation of Request for Comments documents relating to FTP. These include the first proposed file transfer mechanisms in 1971 that were developed for implementation on hosts at M.I.T. (RFC 114), plus comments and discussion in RFC 141.

RFC 172 provided a user-level oriented protocol for file transfer between host computers (including terminal IMPs). A revision of this as RFC 265, restated FTP for additional review, while RFC 281 suggested further changes. The use of a "Set Data Type" transaction was proposed in RFC 294 in January 1982.

RFC 354 obsoleted RFCs 264 and 265. The File Transfer Protocol was now defined as a protocol for file transfer between HOSTs on the ARPANET, with the primary function of FTP defined as transfering files efficiently and reliably among hosts and allowing the convenient use of remote file storage capabilities. RFC 385 further commented on errors, emphasis points, and additions to the protocol, while RFC 414 provided a status report on the working server and user FTPs. RFC 430, issued in 1973, (among other RFCs too numerous to mention) presented further comments on FTP. Finally, an "official" FTP document was published as RFC 454.

By July 1973, considerable changes from the last versions of FTP were made, but the general structure remained the same. RFC 542 was published as a new "official" specification to reflect these changes. However, many implementations based on the older specification were not updated.

In 1974, RFCs 607 and 614 continued comments on FTP. RFC 624 proposed further design changes and minor modifications. In 1975, RFC 686 entitled, "Leaving Well Enough Alone", discussed the differences between all of the early and later versions of FTP. RFC 691 presented a minor revision of RFC 686, regarding the subject of print files.

Motivated by the transition from the NCP to the TCP as the underlying protocol, a phoenix was born out of all of the above efforts in RFC 765 as the specification of FTP for use on TCP.

This current edition of the FTP specification is intended to correct some minor documentation errors, to improve the explanation of some protocol features, and to add some new optional commands.

In particular, the following new optional commands are included in this edition of the specification:

CDUP - Change to Parent Directory

SMNT - Structure Mount

STOU - Store Unique

RMD - Remove Directory

MKD - Make Directory

PWD - Print Directory

SYST - System

This specification is compatible with the previous edition. A program implemented in conformance to the previous specification should automatically be in conformance to this specification.

2.2. TERMINOLOGY

ASCII

The ASCII character set is as defined in the ARPA-Internet Protocol Handbook. In FTP, ASCII characters are defined to be the lower half of an eight-bit code set (i.e., the most significant bit is zero).

access controls

Access controls define users' access privileges to the use of a system, and to the files in that system. Access controls are necessary to prevent unauthorized or accidental use of files. It is the prerogative of a server-FTP process to invoke access controls.

byte size

There are two byte sizes of interest in FTP: the logical byte size of the file, and the transfer byte size used for the transmission of the data. The transfer byte size is always 8 bits. The transfer byte size is not necessarily the byte size in which data is to be stored in a system, nor the logical byte size for interpretation of the structure of the data.

control connection

The communication path between the USER-PI and SERVER-PI for the exchange of commands and replies. This connection follows the Telnet Protocol.

data connection

A full duplex connection over which data is transferred, in a specified mode and type. The data transferred may be a part of a file, an entire file or a number of files. The path may be between a server-DTP and a user-DTP, or between two server-DTPs.

data port

The passive data transfer process "listens" on the data port for a connection from the active transfer process in order to open the data connection.

DTP

The data transfer process establishes and manages the data connection. The DTP can be passive or active.

End-of-Line

The end-of-line sequence defines the separation of printing lines. The sequence is Carriage Return, followed by Line Feed.

EOF

The end-of-file condition that defines the end of a file being transferred.

EOR

The end-of-record condition that defines the end of a record being transferred.

error recovery

A procedure that allows a user to recover from certain errors such as failure of either host system or transfer process. In FTP, error recovery may involve restarting a file transfer at a given checkpoint.

FTP commands

A set of commands that comprise the control information flowing from the user-FTP to the server-FTP process.

file

An ordered set of computer data (including programs), of arbitrary length, uniquely identified by a pathname.

mode

The mode in which data is to be transferred via the data connection. The mode defines the data format during transfer including EOR and EOF. The transfer modes defined in FTP are described in the Section on Transmission Modes.

NVT

The Network Virtual Terminal as defined in the Telnet Protocol.

NVFS

The Network Virtual File System. A concept which defines a standard network file system with standard commands and pathname conventions.

page

A file may be structured as a set of independent parts called pages. FTP supports the transmission of discontinuous files as independent indexed pages.

pathname

Pathname is defined to be the character string which must be input to a file system by a user in order to identify a file. Pathname normally contains device and/or directory names, and file name specification. FTP does not yet specify a standard pathname convention. Each user must follow the file naming conventions of the file systems involved in the transfer.

PI

The protocol interpreter. The user and server sides of the protocol have distinct roles implemented in a user-PI and a server-PI.

record

A sequential file may be structured as a number of contiguous parts called records. Record structures are supported by FTP but a file need not have record structure.

reply

A reply is an acknowledgment (positive or negative) sent from server to user via the control connection in response to FTP commands. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

server-DTP

The data transfer process, in its normal "active" state, establishes the data connection with the "listening" data port. It sets up parameters for transfer and storage, and transfers data on command from its PI. The DTP can be placed in a "passive" state to listen for, rather than initiate a connection on the data port.

server-FTP process

A process or set of processes which perform the function of file transfer in cooperation with a user-FTP process and, possibly, another server. The functions consist of a protocol interpreter (PI) and a data transfer process (DTP).

server-PI

The server protocol interpreter "listens" on Port L for a connection from a user-PI and establishes a control communication connection. It receives standard FTP commands from the user-PI, sends replies, and governs the server-DTP.

type

The data representation type used for data transfer and storage. Type implies certain transformations between the time of data storage and data transfer. The representation types defined in FTP are described in the Section on Establishing Data Connections.

user

A person or a process on behalf of a person wishing to obtain file transfer service. The human user may interact directly with a server-FTP process, but use of a user-FTP process is preferred since the protocol design is weighted towards automata.

user-DTP

The data transfer process "listens" on the data port for a connection from a server-FTP process. If two servers are transferring data between them, the user-DTP is inactive.

user-FTP process

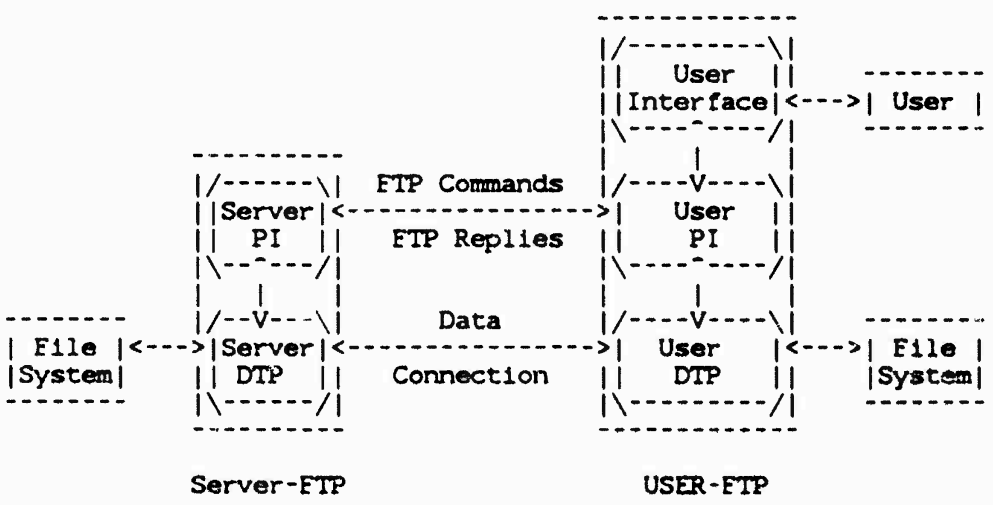
A set of functions including a protocol interpreter, a data transfer process and a user interface which together perform the function of file transfer in cooperation with one or more server-FTP processes. The user interface allows a local language to be used in the command-reply dialogue with the user.

user-PI

The user protocol interpreter initiates the control connection from its port U to the server-FTP process, initiates FTP commands, and governs the user-DTP if that process is part of the file transfer.

2.3. THE FTP MODEL

With the above definitions in mind, the following model (shown in Figure 1) may be diagrammed for an FTP service.



NOTES: 1. The data connection may be used in either direction.
2. The data connection need not exist all of the time.

Figure 1 Model for FTP Use

In the model described in Figure 1, the user-protocol interpreter initiates the control connection. The control connection follows the Telnet protocol. At the initiation of the user, standard FTP commands are generated by the user-PI and transmitted to the server process via the control connection. (The user may establish a direct control connection to the server-FTP, from a TAC terminal for example, and generate standard FTP commands independently, bypassing the user-FTP process.) Standard replies are sent from the server-PI to the user-PI over the control connection in response to the commands.

The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of file system operation (store, retrieve, append, delete, etc.). The user-DTP or its designate should "listen" on the specified data port, and the server initiate the data connection and data transfer in accordance with the specified parameters. It should be noted that the data port need not be in

the same host that initiates the FTP commands via the control connection, but the user or the user-FTP process must ensure a "listen" on the specified data port. It ought to also be noted that the data connection may be used for simultaneous sending and receiving.

In another situation a user might wish to transfer files between two hosts, neither of which is a local host. The user sets up control connections to the two servers and then arranges for a data connection between them. In this manner, control information is passed to the user-PI but data is transferred between the server data transfer processes. Following is a model of this server-server interaction.

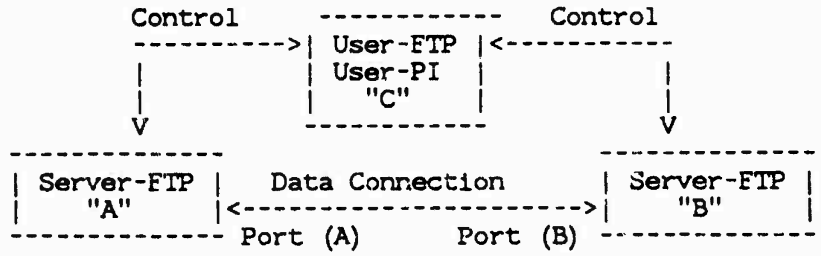


Figure 2

The protocol requires that the control connections be open while data transfer is in progress. It is the responsibility of the user to request the closing of the control connections when finished using the FTP service, while it is the server who takes the action. The server may abort data transfer if the control connections are closed without command.

The Relationship between FTP and Telnet:

The FTP uses the Telnet protocol on the control connection. This can be achieved in two ways: first, the user-PI or the server-PI may implement the rules of the Telnet Protocol directly in their own procedures; or, second, the user-PI or the server-PI may make use of the existing Telnet module in the system.

Ease of implementaion, sharing code, and modular programming argue for the second approach. Efficiency and independence

argue for the first approach. In practice, FTP relies on very little of the Telnet Protocol, so the first approach does not necessarily involve a large amount of code.

3. DATA TRANSFER FUNCTIONS

Files are transferred only via the data connection. The control connection is used for the transfer of commands, which describe the functions to be performed, and the replies to these commands (see the Section on FTP Replies). Several commands are concerned with the transfer of data between hosts. These data transfer commands include the MODE command which specify how the bits of the data are to be transmitted, and the STRUcture and TYPE commands, which are used to define the way in which the data are to be represented. The transmission and representation are basically independent but the "Stream" transmission mode is dependent on the file structure attribute and if "Compressed" transmission mode is used, the nature of the filler byte depends on the representation type.

3.1. DATA REPRESENTATION AND STORAGE

Data is transferred from a storage device in the sending host to a storage device in the receiving host. Often it is necessary to perform certain transformations on the data because data storage representations in the two systems are different. For example, NVT-ASCII has different data storage representations in different systems. DEC TOPS-20s's generally store NVT-ASCII as five 7-bit ASCII characters, left-justified in a 36-bit word. IBM Mainframe's store NVT-ASCII as 8-bit EBCDIC codes. Multics stores NVT-ASCII as four 9-bit characters in a 36-bit word. It is desirable to convert characters into the standard NVT-ASCII representation when transmitting text between dissimilar systems. The sending and receiving sites would have to perform the necessary transformations between the standard representation and their internal representations.

A different problem in representation arises when transmitting binary data (not character codes) between host systems with different word lengths. It is not always clear how the sender should send data, and the receiver store it. For example, when transmitting 32-bit bytes from a 32-bit word-length system to a 36-bit word-length system, it may be desirable (for reasons of efficiency and usefulness) to store the 32-bit bytes right-justified in a 36-bit word in the latter system. In any case, the user should have the option of specifying data representation and transformation functions. It should be noted

that FTP provides for very limited data type representations. Transformations desired beyond this limited capability should be performed by the user directly.

3.1.1. DATA TYPES

Data representations are handled in FTP by a user specifying a representation type. This type may implicitly (as in ASCII or EBCDIC) or explicitly (as in Local byte) define a byte size for interpretation which is referred to as the "logical byte size." Note that this has nothing to do with the byte size used for transmission over the data connection, called the "transfer byte size", and the two should not be confused. For example, NVT-ASCII has a logical byte size of 8 bits. If the type is Local byte, then the TYPE command has an obligatory second parameter specifying the logical byte size. The transfer byte size is always 8 bits.

3.1.1.1. ASCII TYPE

This is the default type and must be accepted by all FTP implementations. It is intended primarily for the transfer of text files, except when both hosts would find the EBCDIC type more convenient.

The sender converts the data from an internal character representation to the standard 8-bit NVT-ASCII representation (see the Telnet specification). The receiver will convert the data from the standard form to his own internal form.

In accordance with the NVT standard, the <CRLF> sequence should be used where necessary to denote the end of a line of text. (See the discussion of file structure at the end of the Section on Data Representation and Storage.)

Using the standard NVT-ASCII representation means that data must be interpreted as 8-bit bytes.

The Format parameter for ASCII and EBCDIC types is discussed below.

3.1.1.2. EBCDIC TYPE

This type is intended for efficient transfer between hosts which use EBCDIC for their internal character representation.

For transmission, the data are represented as 8-bit EBCDIC characters. The character code is the only difference between the functional specifications of EBCDIC and ASCII types.

End-of-line (as opposed to end-of-record--see the discussion of structure) will probably be rarely used with EBCDIC type for purposes of denoting structure, but where it is necessary the <NL> character should be used.

3.1.1.3. IMAGE TYPE

The data are sent as contiguous bits which, for transfer, are packed into the 8-bit transfer bytes. The receiving site must store the data as contiguous bits. The structure of the storage system might necessitate the padding of the file (or of each record, for a record-structured file) to some convenient boundary (byte, word or block). This padding, which must be all zeros, may occur only at the end of the file (or at the end of each record) and there must be a way of identifying the padding bits so that they may be stripped off if the file is retrieved. The padding transformation should be well publicized to enable a user to process a file at the storage site.

Image type is intended for the efficient storage and retrieval of files and for the transfer of binary data. It is recommended that this type be accepted by all FTP implementations.

3.1.1.4. LOCAL TYPE

The data is transferred in logical bytes of the size specified by the obligatory second parameter, Byte size. The value of Byte size must be a decimal integer; there is no default value. The logical byte size is not necessarily the same as the transfer byte size. If there is a difference in byte sizes, then the logical bytes should be packed contiguously, disregarding transfer byte boundaries and with any necessary padding at the end.

When the data reaches the receiving host, it will be transformed in a manner dependent on the logical byte size and the particular host. This transformation must be invertible (i.e., an identical file can be retrieved if the same parameters are used) and should be well publicized by the FTP implementors.

For example, a user sending 36-bit floating-point numbers to a host with a 32-bit word could send that data as Local byte with a logical byte size of 36. The receiving host would then be expected to store the logical bytes so that they could be easily manipulated; in this example putting the 36-bit logical bytes into 64-bit double words should suffice.

In another example, a pair of hosts with a 36-bit word size may send data to one another in words by using TYPE L 36. The data would be sent in the 8-bit transmission bytes packed so that 9 transmission bytes carried two host words.

3.1.1.5. FORMAT CONTROL

The types ASCII and EBCDIC also take a second (optional) parameter; this is to indicate what kind of vertical format control, if any, is associated with a file. The following data representation types are defined in FTP:

A character file may be transferred to a host for one of three purposes: for printing, for storage and later retrieval, or for processing. If a file is sent for printing, the receiving host must know how the vertical format control is represented. In the second case, it must be possible to store a file at a host and then retrieve it later in exactly the same form. Finally, it should be possible to move a file from one host to another and process the file at the second host without undue trouble. A single ASCII or EBCDIC format does not satisfy all these conditions. Therefore, these types have a second parameter specifying one of the following three formats:

3.1.1.5.1. NON PRINT

This is the default format to be used if the second (format) parameter is omitted. Non-print format must be accepted by all FTP implementations.

The file need contain no vertical format information. If it is passed to a printer process, this process may assume standard values for spacing and margins.

Normally, this format will be used with files destined for processing or just storage.

3.1.1.5.2. TELNET FORMAT CONTROLS

The file contains ASCII/EBCDIC vertical format controls (i.e., <CR>, <LF>, <NL>, <VT>, <FF>) which the printer process will interpret appropriately. <CRLF>, in exactly this sequence, also denotes end-of-line.

3.1.1.5.2. CARRIAGE CONTROL (ASA)

The file contains ASA (FORTRAN) vertical format control characters. (See RFC 740 Appendix C; and Communications of the ACM, Vol. 7, No. 10, p. 606, October 1964.) In a line or a record formatted according to the ASA Standard, the first character is not to be printed. Instead, it should be used to determine the vertical movement of the paper which should take place before the rest of the record is printed.

The ASA Standard specifies the following control characters:

Character	Vertical Spacing
blank	Move paper up one line
0	Move paper up two lines
1	Move paper to top of next page
+	No movement, i.e., overprint

Clearly there must be some way for a printer process to distinguish the end of the structural entity. If a file has record structure (see below) this is no problem; records will be explicitly marked during transfer and storage. If the file has no record structure, the <CRLF> end-of-line sequence is used to separate printing lines, but these format effectors are overridden by the ASA controls.

3.1.2. DATA STRUCTURES

In addition to different representation types, FTP allows the structure of a file to be specified. Three file structures are defined in FTP:

file-structure,	where there is no internal structure and the file is considered to be a continuous sequence of data bytes,
record-structure,	where the file is made up of sequential records,
and page-structure,	where the file is made up of independent indexed pages.

File-structure is the default to be assumed if the STRUcture command has not been used but both file and record structures must be accepted for "text" files (i.e., files with TYPE ASCII or EBCDIC) by all FTP implementations. The structure of a file will affect both the transfer mode of a file (see the Section on Transmission Modes) and the interpretation and storage of the file.

The "natural" structure of a file will depend on which host stores the file. A source-code file will usually be stored on an IBM Mainframe in fixed length records but on a DEC TOPS-20 as a stream of characters partitioned into lines, for example by <CRLF>. If the transfer of files between such disparate sites is to be useful, there must be some way for one site to recognize the other's assumptions about the file.

With some sites being naturally file-oriented and others naturally record-oriented there may be problems if a file with one structure is sent to a host oriented to the other. If a text file is sent with record-structure to a host which is file oriented, then that host should apply an internal transformation to the file based on the record structure. Obviously, this transformation should be useful, but it must also be invertible so that an identical file may be retrieved using record structure.

In the case of a file being sent with file-structure to a record-oriented host, there exists the question of what criteria the host should use to divide the file into records which can be processed locally. If this division is necessary, the FTP implementation should use the end-of-line sequence,

<CRLF> for ASCII, or <NL> for EBCDIC text files, as the delimiter. If an FTP implementation adopts this technique, it must be prepared to reverse the transformation if the file is retrieved with file-structure.

3.1.2.1. FILE STRUCTURE

File structure is the default to be assumed if the STRUcture command has not been used.

In file-structure there is no internal structure and the file is considered to be a continuous sequence of data bytes.

3.1.2.2. RECORD STRUCTURE

Record structures must be accepted for "text" files (i.e., files with TYPE ASCII or EBCDIC) by all FTP implementations.

In record-structure the file is made up of sequential records.

3.1.2.3. PAGE STRUCTURE

To transmit files that are discontinuous, FTP defines a page structure. Files of this type are sometimes known as "random access files" or even as "holey files". In these files there is sometimes other information associated with the file as a whole (e.g., a file descriptor), or with a section of the file (e.g., page access controls), or both. In FTP, the sections of the file are called pages.

To provide for various page sizes and associated information, each page is sent with a page header. The page header has the following defined fields:

Header Length

The number of logical bytes in the page header including this byte. The minimum header length is 4.

Page Index

The logical page number of this section of the file. This is not the transmission sequence number of this page, but the index used to identify this page of the file.

Data Length

The number of logical bytes in the page data. The minimum data length is 0.

Page Type

The type of page this is. The following page types are defined:

0 = Last Page

This is used to indicate the end of a paged structured transmission. The header length must be 4, and the data length must be 0.

1 = Simple Page

This is the normal type for simple paged files with no page level associated control information. The header length must be 4.

2 = Descriptor Page

This type is used to transmit the descriptive information for the file as a whole.

3 = Access Controlled Page

This type includes an additional header field for paged files with page level access control information. The header length must be 5.

Optional Fields

Further header fields may be used to supply per page control information, for example, per page access control.

All fields are one logical byte in length. The logical byte size is specified by the TYPE command. See Appendix I for further details and a specific case at the page structure.

A note of caution about parameters: a file must be stored and retrieved with the same parameters if the retrieved version is to

be identical to the version originally transmitted. Conversely, FTP implementations must return a file identical to the original if the parameters used to store and retrieve a file are the same.

3.2. ESTABLISHING DATA CONNECTIONS

The mechanics of transferring data consists of setting up the data connection to the appropriate ports and choosing the parameters for transfer. Both the user and the server-DTPs have a default data port. The user-process default data port is the same as the control connection port (i.e., U). The server-process default data port is the port adjacent to the control connection port (i.e., L-1).

The transfer byte size is 8-bit bytes. This byte size is relevant only for the actual transfer of the data; it has no bearing on representation of the data within a host's file system.

The passive data transfer process (this may be a user-DTP or a second server-DTP) shall "listen" on the data port prior to sending a transfer request command. The FTP request command determines the direction of the data transfer. The server, upon receiving the transfer request, will initiate the data connection to the port. When the connection is established, the data transfer begins between DTP's, and the server-PI sends a confirming reply to the user-PI.

Every FTP implementation must support the use of the default data ports, and only the USER-PI can initiate a change to non-default ports.

It is possible for the user to specify an alternate data port by use of the PORT command. The user may want a file dumped on a TAC line printer or retrieved from a third party host. In the latter case, the user-PI sets up control connections with both server-PI's. One server is then told (by an FTP command) to "listen" for a connection which the other will initiate. The user-PI sends one server-PI a PORT command indicating the data port of the other. Finally, both are sent the appropriate transfer commands. The exact sequence of commands and replies sent between the user-controller and the servers is defined in the Section on FTP Replies.

In general, it is the server's responsibility to maintain the data connection--to initiate it and to close it. The exception to this

is when the user-DTP is sending the data in a transfer mode that requires the connection to be closed to indicate EOF. The server MUST close the data connection under the following conditions:

1. The server has completed sending data in a transfer mode that requires a close to indicate EOF.

2. The server receives an ABORT command from the user.

3. The port specification is changed by a command from the user.

4. The control connection is closed legally or otherwise.

5. An irrecoverable error condition occurs.

Otherwise the close is a server option, the exercise of which the server must indicate to the user-process by either a 250 or 226 reply only.

3.3. DATA CONNECTION MANAGEMENT

Default Data Connection Ports: All FTP implementations must support use of the default data connection ports, and only the User-PI may initiate the use of non-default ports.

Negotiating Non-Default Data Ports: The User-PI may specify a non-default user side data port with the PORT command. The User-PI may request the server side to identify a non-default server side data port with the PASV command. Since a connection is defined by the pair of addresses, either of these actions is enough to get a different data connection, still it is permitted to do both commands to use new ports on both ends of the data connection.

Reuse of the Data Connection: When using the stream mode of data transfer the end of the file must be indicated by closing the connection. This causes a problem if multiple files are to be transfered in the session, due to need for TCP to hold the connection record for a time out period to guarantee the reliable communication. Thus the connection can not be reopened at once.

There are two solutions to this problem. The first is to negotiate a non-default port. The second is to use another transfer mode.

A comment on transfer modes. The stream transfer mode is

inherently unreliable, since one can not determine if the connection closed prematurely or not. The other transfer modes (Block, Compressed) do not close the connection to indicate the end of file. They have enough FTP encoding that the data connection can be parsed to determine the end of the file. Thus using these modes one can leave the data connection open for multiple file transfers.

3.4. TRANSMISSION MODES

The next consideration in transferring data is choosing the appropriate transmission mode. There are three modes: one which formats the data and allows for restart procedures; one which also compresses the data for efficient transfer; and one which passes the data with little or no processing. In this last case the mode interacts with the structure attribute to determine the type of processing. In the compressed mode, the representation type determines the filler byte.

All data transfers must be completed with an end-of-file (EOF) which may be explicitly stated or implied by the closing of the data connection. For files with record structure, all the end-of-record markers (EOR) are explicit, including the final one. For files transmitted in page structure a "last-page" page type is used.

NOTE: In the rest of this section, byte means "transfer byte" except where explicitly stated otherwise.

For the purpose of standardized transfer, the sending host will translate its internal end of line or end of record denotation into the representation prescribed by the transfer mode and file structure, and the receiving host will perform the inverse translation to its internal denotation. An IBM Mainframe record count field may not be recognized at another host, so the end-of-record information may be transferred as a two byte control code in Stream mode or as a flagged bit in a Block or Compressed mode descriptor. End-of-line in an ASCII or EBCDIC file with no record structure should be indicated by <CRLF> or <NL>, respectively. Since these transformations imply extra work for some systems, identical systems transferring non-record structured text files might wish to use a binary representation and stream mode for the transfer.

The following transmission modes are defined in FTP:

3.4.1. STREAM MODE

The data is transmitted as a stream of bytes. There is no restriction on the representation type used; record structures are allowed.

In a record structured file EOR and EOF will each be indicated by a two-byte control code. The first byte of the control code will be all ones, the escape character. The second byte will have the low order bit on and zeros elsewhere for EOR and the second low order bit on for EOF; that is, the byte will have value 1 for EOR and value 2 for EOF. EOR and EOF may be indicated together on the last byte transmitted by turning both low order bits on (i.e., the value 3). If a byte of all ones was intended to be sent as data, it should be repeated in the second byte of the control code.

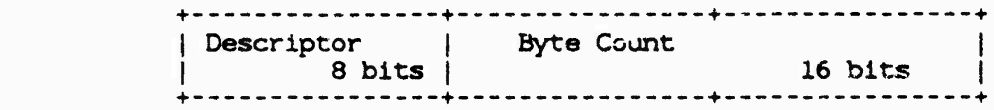
If the structure is a file structure, the EOF is indicated by the sending host closing the data connection and all bytes are data bytes.

3.4.2. BLOCK MODE

The file is transmitted as a series of data blocks preceded by one or more header bytes. The header bytes contain a count field, and descriptor code. The count field indicates the total length of the data block in bytes, thus marking the beginning of the next data block (there are no filler bits). The descriptor code defines: last block in the file (EOF) last block in the record (EOR), restart marker (see the Section on Error Recovery and Restart) or suspect data (i.e., the data being transferred is suspected of errors and is not reliable). This last code is NOT intended for error control within FTP. It is motivated by the desire of sites exchanging certain types of data (e.g., seismic or weather data) to send and receive all the data despite local errors (such as "magnetic tape read errors"), but to indicate in the transmission that certain portions are suspect). Record structures are allowed in this mode, and any representation type may be used.

The header consists of the three bytes. Of the 24 bits of header information, the 16 low order bits shall represent byte count, and the 8 high order bits shall represent descriptor codes as shown below.

Block Header



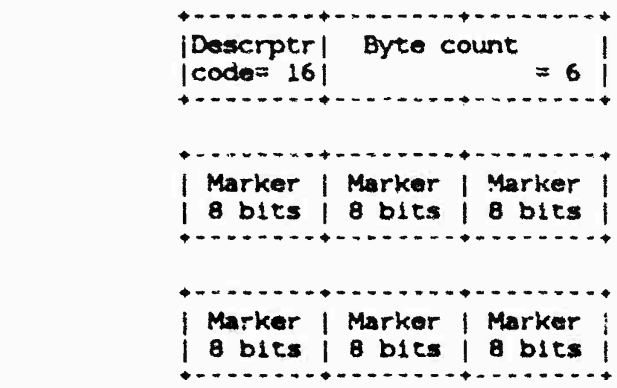
The descriptor codes are indicated by bit flags in the descriptor byte. Four codes have been assigned, where each code number is the decimal value of the corresponding bit in the byte.

Code	Meaning
128	End of data block is EOR
64	End of data block is EOF
32	Suspected errors in data block
16	Data block is a restart marker

With this encoding, more than one descriptor coded condition may exist for a particular block. As many bits as necessary may be flagged.

The restart marker is embedded in the data stream as an integral number of 8-bit bytes representing printable characters in the language being used over the control connection (e.g., default--NVT-ASCII). <SP> (Space, in the appropriate language) must not be used WITHIN a restart marker.

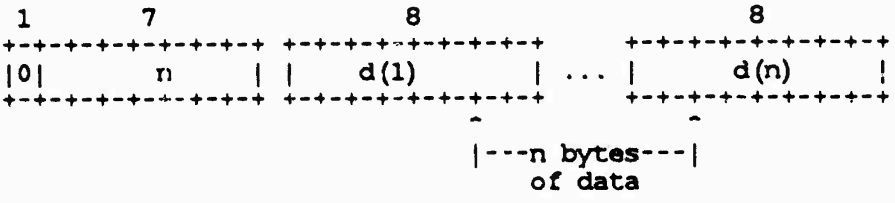
For example, to transmit a six-character marker, the following would be sent:



3.4.3. COMPRESSED MODE

There are three kinds of information to be sent: regular data, sent in a byte string; compressed data, consisting of replications or filler; and control information, sent in a two-byte escape sequence. If n>0 bytes (up to 127) of regular data are sent, these n bytes are preceded by a byte with the left-most bit set to 0 and the right-most 7 bits containing the number n.

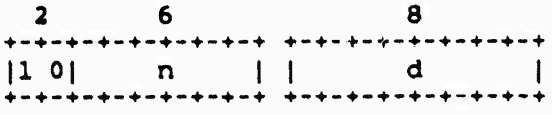
Byte string:



String of n data bytes d(1),..., d(n)
Count n must be positive.

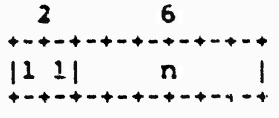
To compress a string of n replications of the data byte d, the following 2 bytes are sent:

Replicated Byte:



A string of n filler bytes can be compressed into a single byte, where the filler byte varies with the representation type. If the type is ASCII or EBCDIC the filler byte is <SP> (Space, ASCII code 32, EBCDIC code 64). If the type is Image or Local byte the filler is a zero byte.

Filler String:



The escape sequence is a double byte, the first of which is the

escape byte (all zeros) and the second of which contains descriptor codes as defined in Block mode. The descriptor codes have the same meaning as in Block mode and apply to the succeeding string of bytes.

Compressed mode is useful for obtaining increased bandwidth on very large network transmissions at a little extra CPU cost. It can be most effectively used to reduce the size of printer files such as those generated by RJE hosts.

3.5. ERROR RECOVERY AND RESTART

There is no provision for detecting bits lost or scrambled in data transfer; this level of error control is handled by the TCP. However, a restart procedure is provided to protect users from gross system failures (including failures of a host, an FTP-process, or the underlying network).

The restart procedure is defined only for the block and compressed modes of data transfer. It requires the sender of data to insert a special marker code in the data stream with some marker information. The marker information has meaning only to the sender, but must consist of printable characters in the default or negotiated language of the control connection (ASCII or EBCDIC). The marker could represent a bit-count, a record-count, or any other information by which a system may identify a data checkpoint. The receiver of data, if it implements the restart procedure, would then mark the corresponding position of this marker in the receiving system, and return this information to the user.

In the event of a system failure, the user can restart the data transfer by identifying the marker point with the FTP restart procedure. The following example illustrates the use of the restart procedure.

The sender of the data inserts an appropriate marker block in the data stream at a convenient point. The receiving host marks the corresponding data point in its file system and conveys the last known sender and receiver marker information to the user, either directly or over the control connection in a 110 reply (depending on who is the sender). In the event of a system failure, the user or controller process restarts the server at the last server marker by sending a restart command with server's marker code as its argument. The restart command is transmitted over the control

connection and is immediately followed by the command (such as RETR, STOR or LIST) which was being executed when the system failure occurred.

4. FILE TRANSFER FUNCTIONS

The communication channel from the user-PI to the server-PI is established as a TCP connection from the user to the standard server port. The user protocol interpreter is responsible for sending FTP commands and interpreting the replies received; the server-PI interprets commands, sends replies and directs its DTP to set up the data connection and transfer the data. If the second party to the data transfer (the passive transfer process) is the user-DTP, then it is governed through the internal protocol of the user-FTP host; if it is a second server-DTP, then it is governed by its PI on command from the user-PI. The FTP replies are discussed in the next section. In the description of a few of the commands in this section, it is helpful to be explicit about the possible replies.

4.1. FTP COMMANDS

4.1.1. ACCESS CONTROL COMMANDS

The following commands specify access control identifiers (command codes are shown in parentheses).

USER NAME (USER)

The argument field is a Telnet string identifying the user. The user identification is that which is required by the server for access to its file system. This command will normally be the first command transmitted by the user after the control connections are made (some servers may require this). Additional identification information in the form of a password and/or an account command may also be required by some servers. Servers may allow a new USER command to be entered at any point in order to change the access control and/or accounting information. This has the effect of flushing any user, password, and account information already supplied and beginning the login sequence again. All transfer parameters are unchanged and any file transfer in progress is completed under the old access control parameters.

PASSWORD (PASS)

The argument field is a Telnet string specifying the user's password. This command must be immediately preceded by the user name command, and, for some sites, completes the user's identification for access control. Since password information is quite sensitive, it is desirable in general to "mask" it or suppress typeout. It appears that the server has no foolproof way to achieve this. It is therefore the responsibility of the user-FTP process to hide the sensitive password information.

ACCOUNT (ACCT)

The argument field is a Telnet string identifying the user's account. The command is not necessarily related to the USER command, as some sites may require an account for login and others only for specific access, such as storing files. In the latter case the command may arrive at any time.

There are reply codes to differentiate these cases for the automation: when account information is required for login, the response to a successful PASSword command is reply code 332. On the other hand, if account information is NOT required for login, the reply to a successful PASSword command is 230; and if the account information is needed for a command issued later in the dialogue, the server should return a 332 or 532 reply depending on whether it stores (pending receipt of the ACCounT command) or discards the command, respectively.

CHANGE WORKING DIRECTORY (CWD)

This command allows the user to work with a different directory or dataset for file storage or retrieval without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

CHANGE TO PARENT DIRECTORY (CDUP)

This command is a special case of CWD, and is included to simplify the implementation of programs for transferring directory trees between operating systems having different

syntaxes for naming the parent directory. The reply codes shall be identical to the reply codes of CWD. See Appendix II for further details.

STRUCTURE MOUNT (SMNT)

This command allows the user to mount a different file system data structure without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

REINITIALIZE (REIN)

This command terminates a USER, flushing all I/O and account information, except to allow any transfer in progress to be completed. All parameters are reset to the default settings and the control connection is left open. This is identical to the state in which a user finds himself immediately after the control connection is opened. A USER command may be expected to follow.

LOGOUT (QUIT)

This command terminates a USER and if file transfer is not in progress, the server closes the control connection. If file transfer is in progress, the connection will remain open for result response and the server will then close it. If the user-process is transferring files for several USERs but does not wish to close and then reopen connections for each, then the REIN command should be used instead of QUIT.

An unexpected close on the control connection will cause the server to take the effective action of an abort (ABOR) and a logout (QUIT).

4.1.2. TRANSFER PARAMETER COMMANDS

All data transfer parameters have default values, and the commands specifying data transfer parameters are required only if the default parameter values are to be changed. The default value is the last specified value, or if no value has been specified, the standard default value is as stated here. This implies that the server must "remember" the applicable default values. The commands may be in any order except that they must precede the FTP service request. The following commands specify data transfer parameters:

DATA PORT (PORT)

The argument is a HOST-PORT specification for the data port to be used in data connection. There are defaults for both the user and server data ports, and under normal circumstances this command and its reply are not needed. If this command is used, the argument is the concatenation of a 32-bit internet host address and a 16-bit TCP port address. This address information is broken into 8-bit fields and the value of each field is transmitted as a decimal number (in character string representation). The fields are separated by commas. A port command would be:

PORT h1,h2,h3,h4,p1,p2

where h1 is the high order 8 bits of the internet host address.

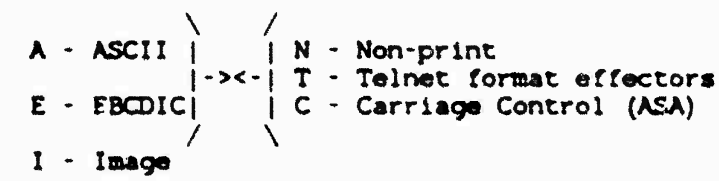
PASSIVE (PASV)

This command requests the server-DTP to "listen" on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command. The response to this command includes the host and port address this server is listening on.

REPRESENTATION TYPE (TYPE)

The argument specifies the representation type as described in the Section on Data Representation and Storage. Several types take a second parameter. The first parameter is denoted by a single Telnet character, as is the second Format parameter for ASCII and EBCDIC; the second parameter for local byte is a decimal integer to indicate Bytesize. The parameters are separated by a <SP> (Space, ASCII code 32).

The following codes are assigned for type:



L <byte size> - Local byte Byte size

The default representation type is ASCII Non-print. If the Format parameter is changed, and later just the first argument is changed, Format then returns to the Non-print default.

FILE STRUCTURE (STRU)

The argument is a single Telnet character code specifying file structure described in the Section on Data Representation and Storage.

The following codes are assigned for structure:

F - File (no record structure)
R - Record structure
P - Page structure

The default structure is File.

TRANSFER MODE (MODE)

The argument is a single Telnet character code specifying the data transfer modes described in the Section on Transmission Modes.

The following codes are assigned for transfer modes:

S - Stream
B - Block
C - Compressed

The default transfer mode is Stream.

4.1.3. FTP SERVICE COMMANDS

The FTP service commands define the file transfer or the file system function requested by the user. The argument of an FTP service command will normally be a pathname. The syntax of pathnames must conform to server site conventions (with standard defaults applicable), and the language conventions of the control connection. The suggested default handling is to use the last specified device, directory or file name, or the standard default defined for local users. The commands may be in any order except that a "rename from" command must be followed by a "rename to" command and the restart command must be followed by the interrupted service command (e.g., STOR or RETR). The data, when transferred in response to FTP service

commands, shall always be sent over the data connection, except for certain informative replies. The following commands specify FTP service requests:

RETRIEVE (RETR)

This command causes the server-DTP to transfer a copy of the file, specified in the pathname, to the server- or user-DTP at the other end of the data connection. The status and contents of the file at the server site shall be unaffected.

STORE (STOR)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data as a file at the server site. If the file specified in the pathname exists at the server site, then its contents shall be replaced by the data being transferred. A new file is created at the server site if the file specified in the pathname does not already exist.

STORE UNIQUE (STOU)

This command behaves like STOR except that the resultant file is to be created in the current directory under a name unique to that directory. The 250 Transfer Started response must include the name generated.

APPEND (with create) (APPE)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data in a file at the server site. If the file specified in the pathname exists at the server site, then the data shall be appended to that file; otherwise the file specified in the pathname shall be created at the server site.

ALLOCATE (ALLO)

This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred. The argument shall be a decimal integer representing the number of bytes (using the logical byte size) of storage to be reserved for the file. For files sent with record or page structure a maximum record or page size (in logical bytes) might also be necessary; this is indicated by a decimal integer in a second argument field of

the command. This second argument is optional, but when present should be separated from the first by the three Telnet characters <SP> R <SP>. This command shall be followed by a STORe or APPEnd command. The ALLO command should be treated as a NOOP (no operation) by those servers which do not require that the maximum size of the file be declared beforehand, and those servers interested in only the maximum record or page size should accept a dummy value in the first argument and ignore it.

RESTART (REST)

The argument field represents the server marker at which file transfer is to be restarted. This command does not cause file transfer but skips over the file to the specified data checkpoint. This command shall be immediately followed by the appropriate FTP service command which shall cause file transfer to resume.

RENAME FROM (RNFR)

This command specifies the old pathname of the file which is to be renamed. This command must be immediately followed by a "rename to" command specifying the new file pathname.

RENAME TO (RNTO)

This command specifies the new pathname of the file specified in the immediately preceding "rename from" command. Together the two commands cause a file to be renamed.

ABORT (ABOR)

This command tells the server to abort the previous FTP service command and any associated transfer of data. The abort command may require "special action", as discussed in the Section on FTP Commands, to force recognition by the server. No action is to be taken if the previous command has been completed (including data transfer). The control connection is not to be closed by the server, but the data connection must be closed.

There are two cases for the server upon receipt of this command: (1) the FTP service command was already completed, or (2) the FTP service command is still in progress.

In the first case, the server closes the data connection (if it is open) and responds with a 226 reply, indicating that the abort command was successfully processed.

In the second case, the server aborts the FTP service in progress and closes the data connection, returning a 426 reply to indicate that the service request terminated abnormally. The server then sends a 226 reply, indicating that the abort command was successfully processed.

DELETE (DELE)

This command causes the file specified in the pathname to be deleted at the server site. If an extra level of protection is desired (such as the query, "Do you really wish to delete?"), it should be provided by the user-FTP process.

REMOVE DIRECTORY (RMD)

This command causes the directory specified in the pathname to be removed as a directory (if the pathname is absolute) or as a subdirectory of the current working directory (if the pathname is relative). See Appendix II.

MAKE DIRECTORY (MKD)

This command causes the directory specified in the pathname to be created as a directory (if the pathname is absolute) or as a subdirectory of the current working directory (if the pathname is relative). See Appendix II.

PRINT WORKING DIRECTORY (PWD)

This command causes the name of the current working directory to be returned in the reply. See Appendix II.

LIST (LIST)

This command causes a list to be sent from the server to the passive DTP. If the pathname specifies a directory or other group of files, the server should transfer a list of files in the specified directory. If the pathname specifies a file then the server should send current information on the file. A null argument implies the user's current working or default directory. The data transfer is over the data connection in type ASCII or type EBCDIC. (The user must

ensure that the TYPE is appropriately ASCII or EBCDIC). Since the information on a file may vary widely from system to system, this information may be hard to use automatically in a program, but may be quite useful to a human user.

NAME LIST (NLST)

This command causes a directory listing to be sent from server to user site. The pathname should specify a directory or other system-specific file group descriptor; a null argument implies the current directory. The server will return a stream of names of files and no other information. The data will be transferred in ASCII or EBCDIC type over the data connection as valid pathname strings separated by <CRLF> or <NL>. (Again the user must ensure that the TYPE is correct.) This command is intended to return information that can be used by a program to further process the files automatically. For example, in the implementation of a "multiple get" function.

SITE PARAMETERS (SITE)

This command is used by the server to provide services specific to his system that are essential to file transfer but not sufficiently universal to be included as commands in the protocol. The nature of these services and the specification of their syntax can be stated in a reply to the HELP SITE command.

SYSTEM (SYST)

This command is used to find out the type of operating system at the server. The reply shall have as its first word one of the system names listed in the current version of the Assigned Numbers document [4].

STATUS (STAT)

This command shall cause a status response to be sent over the control connection in the form of a reply. The command may be sent during a file transfer (along with the Telnet IP and Synch signals--see the Section on FTP Commands) in which case the server will respond with the status of the operation in progress, or it may be sent between file transfers. In the latter case, the command may have an argument field. If the argument is a pathname, the command is analogous to the "list" command except that data shall be

transferred over the control connection. If a partial pathname is given, the server may respond with a list of file names or attributes associated with that specification. If no argument is given, the server should return general status information about the server FTP process. This should include current values of all transfer parameters and the status of connections.

HELP (HELP)

This command shall cause the server to send helpful information regarding its implementation status over the control connection to the user. The command may take an argument (e.g., any command name) and return more specific information as a response. The reply is type 211 or 214. It is suggested that HELP be allowed before entering a USER command. The server may use this reply to specify site-dependent parameters, e.g., in response to HELP SITE.

NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the server send an OK reply.

The File Transfer Protocol follows the specifications of the Telnet protocol for all communications over the control connection. Since the language used for Telnet communication may be a negotiated option, all references in the next two sections will be to the "Telnet language" and the corresponding "Telnet end-of-line code". Currently, one may take these to mean NVT-ASCII and <CRLF>. No other specifications of the Telnet protocol will be cited.

FTP commands are "Telnet strings" terminated by the "Telnet end of line code". The command codes themselves are alphabetic characters terminated by the character <SP> (Space) if parameters follow and Telnet-EOL otherwise. The command codes and the semantics of commands are described in this section; the detailed syntax of commands is specified in the Section on Commands, the reply sequences are discussed in the Section on Sequencing of Commands and Replies, and scenarios illustrating the use of commands are provided in the Section on Typical FTP Scenarios.

FTP commands may be partitioned as those specifying access-control identifiers, data transfer parameters, or FTP service requests. Certain commands (such as ABOR, STAT, QUIT) may be sent over the control connection while a data transfer is in progress. Some

servers may not be able to monitor the control and data connections simultaneously, in which case some special action will be necessary to get the server's attention. The following ordered format is tentatively recommended:

1. User system inserts the Telnet "Interrupt Process" (IP) signal in the Telnet stream.

2. User system sends the Telnet "Synch" signal.

3. User system inserts the command (e.g., ABOR) in the Telnet stream.

4. Server PI, after receiving "IP", scans the Telnet stream for EXACTLY ONE FTP command.

(For other servers this may not be necessary but the actions listed above should have no unusual effect.)

4.2. FTP REPLIES

Replies to File Transfer Protocol commands are devised to ensure the synchronization of requests and actions in the process of file transfer, and to guarantee that the user process always knows the state of the Server. Every command must generate at least one reply, although there may be more than one; in the latter case, the multiple replies must be easily distinguished. In addition, some commands occur in sequential groups, such as USER, PASS and ACCT, or RNFR and RNTO. The replies show the existence of an intermediate state if all preceding commands have been successful. A failure at any point in the sequence necessitates the repetition of the entire sequence from the beginning.

The details of the command-reply sequence are made explicit in a set of state diagrams below.

An FTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is intended for the human user. It is intended that the three digits contain enough encoded information that the user-process (the User-PI) will not need to examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be server-dependent, so there are likely to be varying texts for each reply code.

A reply is defined to contain the 3-digit code, followed by Space

<SP>, followed by one line of text (where some maximum line length has been specified), and terminated by the Telnet end-of-line code. There will be cases however, where the text is longer than a single line. In these cases the complete text must be bracketed so the User-process knows when it may stop reading the reply (i.e. stop processing input on the control connection) and go do other things. This requires a special format on the first line to indicate that more than one line is coming, and another on the last line to designate it as the last. At least one of these must contain the appropriate reply code to indicate the state of the transaction. To satisfy all factions, it was decided that both the first and last line codes should be the same.

Thus the format for multi-line replies is that the first line will begin with the exact required reply code, followed immediately by a Hyphen, "-" (also known as Minus), followed by text. The last line will begin with the same code, followed immediately by Space <SP>, optionally some text, and the Telnet end-of-line code.

For example:

```
123-First line
Second line
  234 A line beginning with numbers
123 The last line
```

The user-process then simply needs to search for the second occurrence of the same reply code, followed by <SP> (Space), at the beginning of a line, and ignore all intermediary lines. If an intermediary line begins with a 3-digit number, the Server must pad the front to avoid confusion.

This scheme allows standard system routines to be used for reply information (such as for the STAT reply), with "artificial" first and last lines tacked on. In rare cases where these routines are able to generate three digits and a Space at the beginning of any line, the beginning of each text line should be offset by some neutral text, like Space.

This scheme assumes that multi-line replies may not be nested.

The three digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated responses by the user-process. The first digit denotes whether the response is good, bad or incomplete. (Referring to the state diagram), an unsophisticated user-process will be able to determine its next action (proceed as planned,

redo, retrench, etc.) by simply examining this first digit. A user-process that wants to know approximately what kind of error occurred (e.g. file system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information (e.g., RNTO command without a preceding RNFR).

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult. The server-FTP process may send at most, one 1yz reply per command.

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server- and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the

user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g., the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered his directory status.)

The following function groupings are encoded in the second digit:

x0z Syntax - These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, unimplemented or superfluous commands.

x1z Information - These are replies to requests for information, such as status or help.

x2z Connections - Replies referring to the control and data connections.

x3z Authentication and accounting - Replies for the login process and accounting procedures.

x4z Unspecified as yet.

x5z File system - These replies indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

The third digit gives a finer gradation of meaning in each of the function categories, specified by the second digit. The list of replies below will illustrate this. Note that the text

associated with each reply is recommended, rather than mandatory, and may even change according to the command with which it is associated. The reply codes, on the other hand, must strictly follow the specifications in the last section; that is, Server implementations should not invent new codes for situations that are only slightly different from the ones described here, but rather should adapt codes already defined.

A command such as TYPE or ALLO whose successful execution does not offer the user-process any new information will cause a 200 reply to be returned. If the command is not implemented by a particular Server-FTP process because it has no relevance to that computer system, for example ALLO at a TOPS20 site, a Positive Completion reply is still desired so that the simple User-process knows it can proceed with its course of action. A 202 reply is used in this case with, for example, the reply text: "No storage allocation necessary." If, on the other hand, the command requests a non-site-specific action and is unimplemented, the response is 502. A refinement of that is the 504 reply for a command that is implemented, but that requests an unimplemented parameter.

4.2.1 Reply Codes by Function Groups

200 Command okay.
500 Syntax error, command unrecognized.
This may include errors such as command line too long.
501 Syntax error in parameters or arguments.
202 Command not implemented, superfluous at this site.
502 Command not implemented.
503 Bad sequence of commands.
504 Command not implemented for that parameter.

110 Restart marker reply.
In this case, the text is exact and not left to the particular implementation; it must read:
MARK yyyy = mmmm
Where yyyy is User-process data stream marker, and mmmm server's equivalent marker (note the spaces between markers and "=").
211 System status, or system help reply.
212 Directory status.
213 File status.
214 Help message.
On how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.
215 NAME system type.
Where NAME is an official system name from the list in the Assigned Numbers document.

120 Service ready in nnn minutes.
220 Service ready for new user.
221 Service closing control connection.
Logged out if appropriate.
421 Service not available, closing control connection.
This may be a reply to any command if the service knows it must shut down.
125 Data connection already open; transfer starting.
225 Data connection open; no transfer in progress.
425 Can't open data connection.
226 Closing data connection.
Requested file action successful (for example, file transfer or file abort).
426 Connection closed; transfer aborted.
227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).

230 User logged in, proceed.
530 Not logged in.
331 User name okay, need password.
332 Need account for login.
532 Need account for storing files.

150 File status okay; about to open data connection.
250 Requested file action okay, completed.
257 "PATHNAME" created.
350 Requested file action pending further information.
450 Requested file action not taken.
File unavailable (e.g., file busy).
550 Requested action not taken.
File unavailable (e.g., file not found, no access).
451 Requested action aborted. Local error in processing.
551 Requested action aborted. Page type unknown.
452 Requested action not taken.
Insufficient storage space in system.
552 Requested file action aborted.
Exceeded storage allocation (for current directory or dataset).
553 Requested action not taken.
File name not allowed.

4.2.2 Numeric Order List of Reply Codes

110 Restart marker reply.
In this case, the text is exact and not left to the particular implementation; it must read:
MARK yyyy = mmmm
Where yyyy is User-process data stream marker, and mmmm server's equivalent marker (note the spaces between markers and "=").
120 Service ready in nnn minutes.
125 Data connection already open; transfer starting.
150 File status okay; about to open data connection.

200 Command okay.
202 Command not implemented, superfluous at this site.
211 System status, or system help reply.
212 Directory status.
213 File status.
214 Help message.
On how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.
215 NAME system type.
Where NAME is an official system name from the list in the Assigned Numbers document.
220 Service ready for new user.
221 Service closing control connection.
Logged out if appropriate.
225 Data connection open; no transfer in progress.
226 Closing data connection.
Requested file action successful (for example, file transfer or file abort).
227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).
230 User logged in, proceed.
250 Requested file action okay, completed.
257 "PATHNAME" created.

331 User name okay, need password.
332 Need account for login.
350 Requested file action pending further information.

421 Service not available, closing control connection.
This may be a reply to any command if the service knows it must shut down.
425 Can't open data connection.
426 Connection closed; transfer aborted.
450 Requested file action not taken.
File unavailable (e.g., file busy).
451 Requested action aborted: local error in processing.
452 Requested action not taken.
Insufficient storage space in system.

500 Syntax error, command unrecognized.
This may include errors such as command line too long.
501 Syntax error in parameters or arguments.
502 Command not implemented.
503 Bad sequence of commands.
504 Command not implemented for that parameter.
530 Not logged in.
532 Need account for storing files.
550 Requested action not taken.
File unavailable (e.g., file not found, no access).
551 Requested action aborted: page type unknown.
552 Requested file action aborted.
Exceeded storage allocation (for current directory or dataset).
553 Requested action not taken.
File name not allowed.

5. DECLARATIVE SPECIFICATIONS

5.1. MINIMUM IMPLEMENTATION

In order to make FTP workable without needless error messages, the following minimum implementation is required for all servers:

TYPE - ASCII Non-print
MODE - Stream
STRUCTURE - File, Record
COMMANDS - USER, QUIT, PORT,
TYPE, MODE, STRU,
for the default values
RETR, STOR,
NOOP.

The default values for transfer parameters are:

TYPE - ASCII Non-print
MODE - Stream
STRU - File

All hosts must accept the above as the standard defaults.

5.2. CONNECTIONS

The server protocol interpreter shall "listen" on Port L. The user or user protocol interpreter shall initiate the full-duplex control connection. Server- and user- processes should follow the conventions of the Telnet protocol as specified in the ARPA-Internet Protocol Handbook [1]. Servers are under no obligation to provide for editing of command lines and may require that it be done in the user host. The control connection shall be closed by the server at the user's request after all transfers and replies are completed.

The user-DTP must "listen" on the specified data port; this may be the default user port (U) or a port specified in the PORT command. The server shall initiate the data connection from his own default data port (L-1) using the specified user data port. The direction of the transfer and the port used will be determined by the FTP service command.

Note that all FTP implementation must support data transfer using the default port, and that only the USER-PI may initiate the use of non-default ports.

When data is to be transferred between two servers, A and B (refer to Figure 2), the user-PI, C, sets up control connections with both server-PI's. One of the servers, say A, is then sent a PASV command telling him to "listen" on his data port rather than initiate a connection when he receives a transfer service command. When the user-PI receives an acknowledgment to the PASV command, which includes the identity of the host and port being listened on, the user-PI then sends A's port, a, to B in a PORT command; a reply is returned. The user-PI may then send the corresponding service commands to A and B. Server B initiates the connection and the transfer proceeds. The command-reply sequence is listed below where the messages are vertically synchronous but horizontally asynchronous:

```
User-PI - Server A                  User-PI - Server B
------------------                  ------------------

C->A : Connect                      C->B : Connect
C->A : PASV
A->C : 227 Entering Passive Mode. A1,A2,A3,A4,a1,a2
                                    C->B : PORT A1,A2,A3,A4,a1,a2
                                    B->C : 200 Okay
C->A : STOR                         C->B : RETR
          B->A : Connect to HOST-A, PORT-a
```

Figure 3

The data connection shall be closed by the server under the conditions described in the Section on Establishing Data Connections. If the data connection is to be closed following a data transfer where closing the connection is not required to indicate the end-of-file, the server must do so immediately. Waiting until after a new transfer command is not permitted because the user-process will have already tested the data connection to see if it needs to do a "listen"; (remember that the user must "listen" on a closed data port BEFORE sending the transfer request). To prevent a race condition here, the server sends a reply (226) after closing the data connection (or if the connection is left open, a "file transfer completed" reply (250) and the user-PI should wait for one of these replies before issuing a new transfer command).

Any time either the user or server see that the connection is being closed by the other side, it should promptly read any remaining data queued on the connection and issue the close on its own side.

5.3. COMMANDS

The commands are Telnet character strings transmitted over the control connections as described in the Section on FTP Commands. The command functions and semantics are described in the Section on Access Control Commands, Transfer Parameter Commands, FTP Service Commands, and Miscellaneous Commands. The command syntax is specified here.

The commands begin with a command code followed by an argument field. The command codes are four or fewer alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus, any of the following may represent the retrieve command:

RETR Retr retr ReTr rETr

This also applies to any symbols representing parameter values, such as A or a for ASCII TYPE. The command codes and the argument fields are separated by one or more spaces.

The argument field consists of a variable length character string ending with the character sequence <CRLF> (Carriage Return, Line Feed) for NVT-ASCII representation; for other negotiated languages a different end of line character might be used. It should be noted that the server is to take no action until the end of line code is received.

The syntax is specified below in NVT-ASCII. All characters in the argument field are ASCII characters including any ASCII represented decimal integers. Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

5.3.1. FTP COMMANDS

The following are the FTP commands:

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
ACCT <SP> <account-information> <CRLF>
CWD  <SP> <pathname> <CRLF>
CDUP <CRLF>
SMNT <SP> <pathname> <CRLF>
QUIT <CRLF>
REIN <CRLF>
PORT <SP> <host-port> <CRLF>
PASV <CRLF>
TYPE <SP> <type-code> <CRLF>
STRU <SP> <structure-code> <CRLF>
MODE <SP> <mode-code> <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
STOU <CRLF>
APPE <SP> <pathname> <CRLF>
ALLO <SP> <decimal-integer>
    [<SP> R <SP> <decimal-integer>] <CRLF>
REST <SP>  marker> <CRLF>
RNFR <SP>  pathname> <CRLF>
RNTO <SP> <pathname> <CRLF>
ABOR <CRLF>
DELE <SP> <pathname> <CRLF>
RMD  <SP> <pathname> <CRLF>
MKD  <SP> <pathname> <CRLF>
PWD  <CRLF>
LIST [<SP> <pathname>] <CRLF>
NLST [<SP> <pathname>] <CRLF>
SITE <SP> <string> <CRLF>
SYST <CRLF>
STAT [<SP> <pathname>] <CRLF>
HELP [<SP> <string>] <CRLF>
NOOP <CRLF>
```

5.3.2. FTP COMMAND ARGUMENTS

The syntax of the above argument fields (using BNF notation where applicable) is:

```
<username> ::= <string>
<password> ::= <string>
<account-information> ::= <string>
<string> ::= <char> | <char><string>
<char> ::= any of the 128 ASCII characters except <CR> and
<LF>
<marker> ::= <pr-string>
<pr-string> ::= <pr-char> | <pr-char><pr-string>
<pr-char> ::= printable characters, any
              ASCII code 33 through 126
<byte-size> ::= <number>
<host-port> ::= <host-number>,<port-number>
<host-number> ::= <number>,<number>,<number>,<number>
<port-number> ::= <number>,<number>
<number> ::= any decimal integer 1 through 255
<form-code> ::= N | T | C
<type-code> ::= A [<sp> <form-code>]
              | E [<sp> <form-code>]
              | I
              | L <sp> <byte-size>
<structure-code> ::= F | R | P
<mode-code> ::= S | B | C
<pathname> ::= <string>
<decimal-integer> ::= any decimal integer
```

5.4. SEQUENCING OF COMMANDS AND REPLIES

The communication between the user and server is intended to be an alternating dialogue. As such, the user issues an FTP command and the server responds with a prompt primary reply. The user should wait for this initial primary success or failure response before sending further commands.

Certain commands require a second reply for which the user should also wait. These replies may, for example, report on the progress or completion of file transfer or the closing of the data connection. They are secondary replies to file transfer commands.

One important group of informational replies is the connection greetings. Under normal circumstances, a server will send a 220 reply, "awaiting input", when the connection is completed. The user should wait for this greeting message before sending any commands. If the server is unable to accept input right away, a 120 "expected delay" reply should be sent immediately and a 220 reply when ready. The user will then know not to hang up if there is a delay.

Spontaneous Replies

Sometimes "the system" spontaneously has a message to be sent to a user (usually all users). For example, "System going down in 15 minutes". There is no provision in FTP for such spontaneous information to be sent from the server to the user. It is recommended that such information be queued in the server-PI and delivered to the user-PI in the next reply (possibly making it a multi-line reply).

The table below lists alternative success and failure replies for each command. These must be strictly adhered to; a server may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

Command-Reply Sequences

In this section, the command-reply sequence is presented. Each command is listed with its possible replies; command groups are listed together. Preliminary replies are listed first (with their succeeding replies indented and under them), then positive and negative completion, and finally intermediary

replies with the remaining commands from the sequence following. This listing forms the basis for the state diagrams, which will be presented separately.

```
Connection Establishment
   120
      220
   220
   421
Login
   USER
      230
      530
      500, 501, 421
      331, 332
   PASS
      230
      202
      530
      500, 501, 503, 421
      332
   ACCT
      230
      202
      530
      500, 501, 503, 421
   CWD
      250
      500, 501, 502, 421, 530, 550
   CDUP
      200
      500, 501, 502, 421, 530, 550
   SMNT
      202, 250
      500, 501, 502, 421, 530, 550
Logout
   REIN
      120
         220
      220
      421
      500, 502
   QUIT
      221
      500
```

```
            Transfer parameters
               PORT
                  200
                  500, 501, 421, 530
               PASV
                  227
                  500, 501, 502, 421, 530
               MODE
                  200
                  500, 501, 504, 421, 530
               TYPE
                  200
                  500, 501, 504, 421, 530
               STRU
                  200
                  500, 501, 504, 421, 530
            File action commands
               ALLO
                  200
                  202
                  500, 501, 504, 421, 530
               REST
                  500, 501, 502, 421, 530
                  350
               STOR
                  125, 150
                     (110)
                     226, 250
                     425, 426, 451, 551, 552
                  532, 450, 452, 553
                  500, 501, 421, 530
               STOU
                  125, 150
                     (110)
                     226, 250
                     425, 426, 451, 551, 552
                  532, 450, 452, 553
                  500, 501, 421, 530
               RETR
                  125, 150
                     (110)
                     226, 250
                     425, 426, 451
                  450, 550
                  500, 501, 421, 530
```

```
LIST
   125, 150
      226, 250
      425, 426, 451
   450
   500, 501, 502, 421, 530
NLST
   125, 150
      226, 250
      425, 426, 451
   450
   500, 501, 502, 421, 530
APPE
   125, 150
      (110)
      226, 250
      425, 426, 451, 551, 552
   532, 450, 550, 452, 553
   500, 501, 502, 421, 530
RNFR
   450, 550
   500, 501, 502, 421, 530
   350
RNTO
   250
   532, 553
   500, 501, 502, 503, 421, 530
DELE
   250
   450, 550
   500, 501, 502, 421, 530
RMD
   250
   500, 501, 502, 421, 530, 550
MKD
   257
   500, 501, 502, 421, 530, 550
PWD
   257
   500, 501, 502, 421, 550
ABOR
   225, 226
   500, 501, 502, 421
```

```
Informational commands
   SYST
      215
      500, 501, 502, 421
   STAT
      211, 212, 213
      450
      500, 501, 502, 421, 530
   HELP
      211, 214
      500, 501, 502, 421
Miscellaneous commands
   SITE
      200
      202
      500, 501, 530
   NOOP
      200
      500 421
```

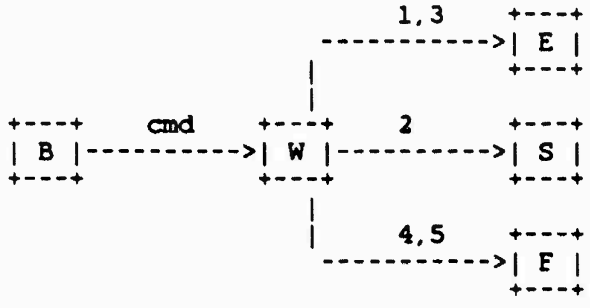
6. STATE DIAGRAMS

Here we present state diagrams for a very simple minded FTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of FTP commands or command sequences.

The command groupings were determined by constructing a model for each command then collecting together the commands with structurally identical models.

For each command or command sequence there are three possible outcomes: success (S), failure (F), and error (E). In the state diagrams below we use the symbol B for "begin", and the symbol W for "wait for reply".

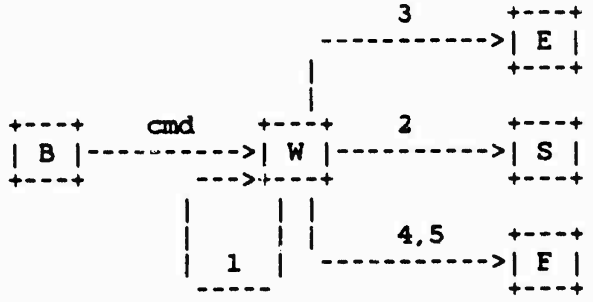
We first present the diagram that represents the largest group of FTP commands:



This diagram models the commands:

ABOR, ALLO, DELE, CWD, CDUP, SMNT, HELP, MODE, NOOP, PASV, QUIT, SITE, PORT, SYST, STAT, RMD, MKD, PWD, STRU, and TYPE.

The other large group of commands is represented by a very similar diagram:

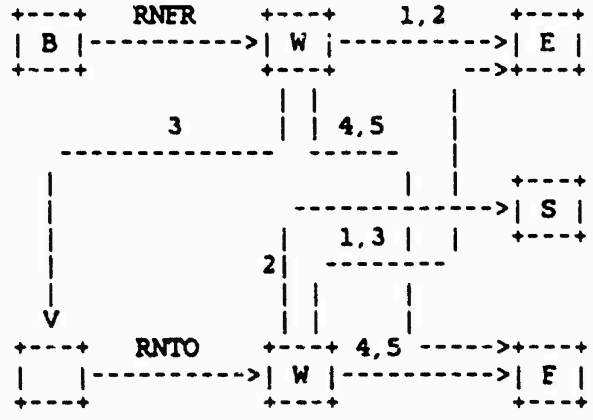


This diagram models the commands:

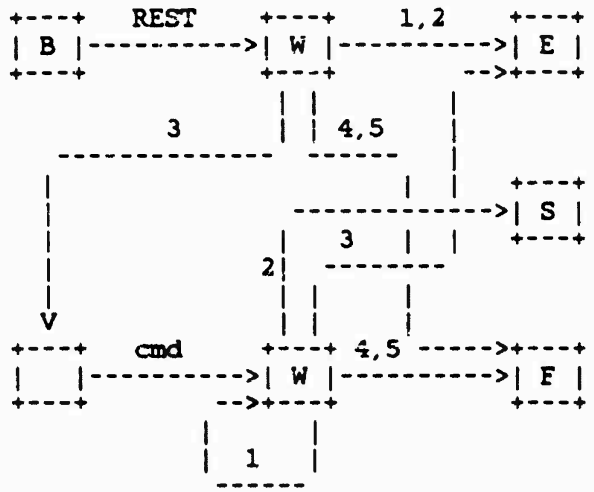
APPE, LIST, NLST, REIN, RETR, STOR, and STOU.

Note that this second model could also be used to represent the first group of commands, the only difference being that in the first group the 100 series replies are unexpected and therefore treated as error, while the second group expects (some may require) 100 series replies. Remember that at most, one 100 series reply is allowed per command.

The remaining diagrams model command sequences, perhaps the simplest of these is the rename sequence:



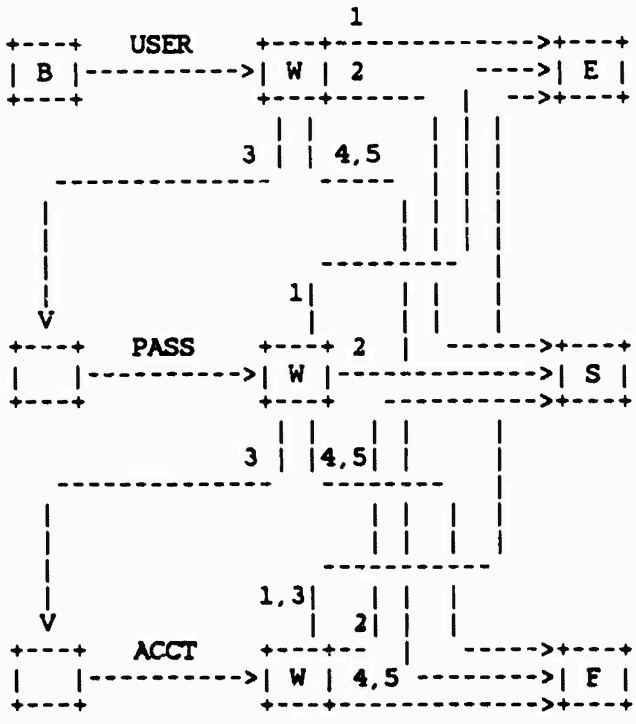
The next diagram is a simple model of the Restart command:



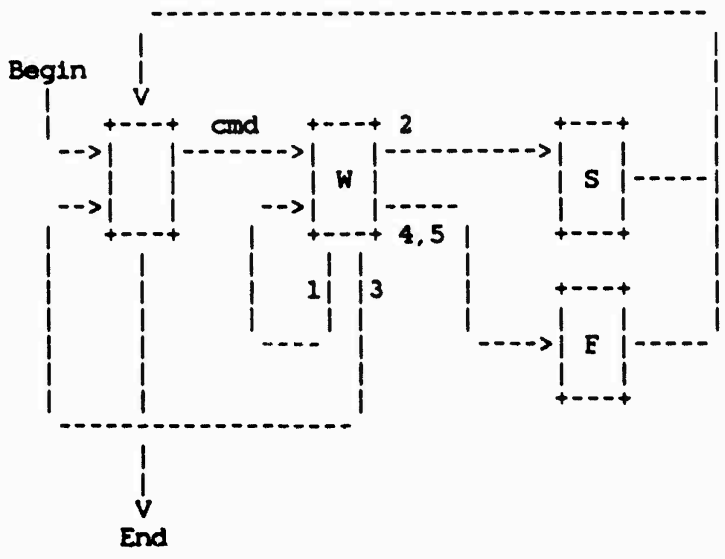
Where "cmd" is APPE, STOR, or RETR.

We note that the above three models are similar. The Restart differs from the Rename two only in the treatment of 100 series replies at the second stage, while the second group expects (some may require) 100 series replies. Remember that at most, one 100 series reply is allowed per command.

The most complicated diagram is for the Login sequence:



Finally, we present a generalized diagram that could be used to model the command and reply interchange:



7. TYPICAL FTP SCENARIO

User at host U wanting to transfer files to/from host S:

In general, the user will communicate to the server via a mediating user-FTP process. The following may be a typical scenario. The user-FTP prompts are shown in parentheses, '---->' represents commands from host U to host S, and '<----' represents replies from host S to host U.

LOCAL COMMANDS BY USER	ACTION INVOLVED
ftp (host) multics<CR>	Connect to host S, port L, establishing control connections. <---- 220 Service ready <CRLF>.
username Doe <CR>	USER Doe<CRLF>---->\ <---- 331 User name ok, need password<CRLF>.
password mumble <CR>	PASS mumble<CRLF>---->\ <---- 230 User logged in<CRLF>.
retrieve (local type) ASCII<CR> (local pathname) test 1 <CR> (for. pathname) test.pl1<CR>	User-FTP opens local file in ASCII. RETR test.pl1<CRLF> ---->\ <---- 150 File status okay; about to open data connection<CRLF>. Server makes data connection to port U. <---- 226 Closing data connection, file transfer successful<CRLF>.
type Image<CR>	TYPE I<CRLF> ---->\ <---- 200 Command OK<CRLF>
store (local type) image<CR> (local pathname) file dump<CR> (for.pathname) >udd>cn>fd<CR>	User-FTP opens local file in Image. STOR >udd>cn>fd<CRLF> ---->\ <---- 550 Access denied<CRLF>
terminate	QUIT <CRLF> ---->\ Server closes all connections.

8. CONNECTION ESTABLISHMENT

The FTP control connection is established via TCP between the user process port U and the server process port L. This protocol is assigned the service port 21 (25 octal), that is L=21.

APPENDIX I - PAGE STRUCTURE

The need for FTP to support page structure derives principally from the need to support efficient transmission of files between TOPS-20 systems, particularly the files used by NLS.

The file system of TOPS-20 is based on the concept of pages. The operating system is most efficient at manipulating files as pages. The operating system provides an interface to the file system so that many applications view files as sequential streams of characters. However, a few applications use the underlying page structures directly, and some of these create holey files.

A TOPS-20 disk file consists of four things: a pathname, a page table, a (possibly empty) set of pages, and a set of attributes.

The pathname is specified in the RETR or STOR command. It includes the directory name, file name, file name extension, and generation number.

The page table contains up to 2**18 entries. Each entry may be EMPTY, or may point to a page. If it is not empty, there are also some page-specific access bits; not all pages of a file need have the same access protection.

A page is a contiguous set of 512 words of 36 bits each.

The attributes of the file, in the File Descriptor Block (FDB), contain such things as creation time, write time, read time, writer's byte-size, end-of-file pointer, count of reads and writes, backup system tape numbers, etc.

Note that there is NO requirement that entries in the page table be contiguous. There may be empty page table slots between occupied ones. Also, the end of file pointer is simply a number. There is no requirement that it in fact point at the "last" datum in the file. Ordinary sequential I/O calls in TOPS-20 will cause the end of file pointer to be left after the last datum written, but other operations may cause it not to be so, if a particular programming system so requires.

In fact, in both of these special cases, "holey" files and end-of-file pointers NOT at the end of the file, occur with NLS data files.

The TOPS-20 paged files can be sent with the FTP transfer parameters: TYPE L 36, STRU P, and MODE S (in fact, any mode could be used).

Each page of information has a header. Each header field, which is a logical byte, is a TOPS-20 word, since the TYPE is L 36.

The header fields are:

Word 0: Header Length.

The header length is 5.

Word 1: Page Index.

If the data is a disk file page, this is the number of that page in the file's page map. Empty pages (holes) in the file are simply not sent. Note that a hole is NOT the same as a page of zeros.

Word 2: Data Length.

The number of data words in this page, following the header. Thus, the total length of the transmission unit is the Header Length plus the Data Length.

Word 3: Page Type.

A code for what type of chunk this is. A data page is type 3, the FDB page is type 2.

Word 4: Page Access Control.

The access bits associated with the page in the file's page map. (This full word quantity is put into AC2 of an SPACS by the program reading from net to disk.)

After the header are Data Length data words. Data Length is currently either 512 for a data page or 31 for an FDB. Trailing zeros in a disk file page may be discarded, making Data Length less than 512 in that case.

APPENDIX II - DIRECTORY COMMANDS

Since UNIX has a tree-like directory structure in which directories are as easy to manipulate as ordinary files, it is useful to expand the FTP servers on these machines to include commands which deal with the creation of directories. Since there are other hosts on the ARPA-Internet which have tree-like directories (including TOPS-20 and Multics), these commands are as general as possible.

Four directory commands have been added to FTP:

MKD pathname

Make a directory with the name "pathname".

RMD pathname

Remove the directory with the name "pathname".

PWD

Print the current working directory name.

CDUP

Change to the parent of the current working directory.

The "pathname" argument should be created (removed) as a subdirectory of the current working directory, unless the "pathname" string contains sufficient information to specify otherwise to the server, e.g., "pathname" is an absolute pathname (in UNIX and Multics), or pathname is something like "<abso.lute.path>" to TOPS-20.

REPLY CODES

The CDUP command is a special case of CWD, and is included to simplify the implementation of programs for transferring directory trees between operating systems having different syntaxes for naming the parent directory. The reply codes for CDUP be identical to the reply codes of CWD.

The reply codes for RMD be identical to the reply codes for its file analogue, DELE.

The reply codes for MKD, however, are a bit more complicated. A freshly created directory will probably be the object of a future

CWD command. Unfortunately, the argument to MKD may not always be a suitable argument for CWD. This is the case, for example, when a TOPS-20 subdirectory is created by giving just the subdirectory name. That is, with a TOPS-20 server FTP, the command sequence

```
MKD MYDIR
CWD MYDIR
```

will fail. The new directory may only be referred to by its "absolute" name; e.g., if the MKD command above were issued while connected to the directory <DFRANKLIN>, the new subdirectory could only be referred to by the name <DFRANKLIN.MYDIR>.

Even on UNIX and Multics, however, the argument given to MKD may not be suitable. If it is a "relative" pathname (i.e., a pathname which is interpreted relative to the current directory), the user would need to be in the same current directory in order to reach the subdirectory. Depending on the application, this may be inconvenient. It is not very robust in any case.

To solve these problems, upon successful completion of an MKD command, the server should return a line of the form:

```
257<space>"<directory-name>"<space><commentary>
```

That is, the server will tell the user what string to use when referring to the created directory. The directory name can contain any character; embedded double-quotes should be escaped by double-quotes (the "quote-doubling" convention).

For example, a user connects to the directory /usr/dm, and creates a subdirectory, named pathname:

```
CWD /usr/dm
200 directory changed to /usr/dm
MKD pathname
257 "/usr/dm/pathname" directory created
```

An example with an embedded double quote:

```
MKD foo"bar
257 "/usr/dm/foo""bar" directory created
CWD /usr/dm/foo"bar
200 directory changed to /usr/dm/foo"bar
```

The prior existence of a subdirectory with the same name is an error, and the server must return an "access denied" error reply in that case.

```
CWD /usr/dm
200 directory changed to /usr/dm
MKD pathname
521-"/usr/dm/pathname" directory already exists;
521 taking no action.
```

The failure replies for MKD are analogous to its file creating cousin, STOR. Also, an "access denied" return is given if a file name with the same name as the subdirectory will conflict with the creation of the subdirectory (this is a problem on UNIX, but shouldn't be one on TOPS-20).

Essentially because the PWD command returns the same type of information as the successful MKD command, the successful PWD command uses the 257 reply code as well.

SUBTLETIES

Because these commands will be most useful in transferring subtrees from one machine to another, carefully observe that the argument to MKD is to be interpreted as a sub-directory of the current working directory, unless it contains enough information for the destination host to tell otherwise. A hypothetical example of its use in the TOPS-20 world:

```
CWD <some.where>
200 Working directory changed
MKD overrainbow
257 "<some.where.overrainbow>" directory created
CWD overrainbow
431 No such directory
CWD <some.where.overrainbow>
200 Working directory changed

CWD <some.where>
200 Working directory changed to <some.where>
MKD <unambiguous>
257 "<unambiguous>" directory created
CWD <unambiguous>
```

Note that the first example results in a subdirectory of the connected directory. In contrast, the argument in the second example contains enough information for TOPS-20 to tell that the

<unambiguous> directory is a top-level directory. Note also that in the first example the user "violated" the protocol by attempting to access the freshly created directory with a name other than the one returned by TOPS-20. Problems could have resulted in this case had there been an <overrainbow> directory; this is an ambiguity inherent in some TOPS-20 implementations. Similar considerations apply to the RMD command. The point is this: except where to do so would violate a host's conventions for denoting relative versus absolute pathnames, the host should treat the operands of the MKD and RMD commands as subdirectories. The 257 reply to the MKD command must always contain the absolute pathname of the created directory.

APPENDIX III - RFCs on FTP

Bhushan, Abhay, "A File Transfer Protocol", RFC 114 (NIC 5823), MIT-Project MAC, 16 April 1971.

Harslem, Eric, and John Heafner, "Comments on RFC 114 (A File Transfer Protocol)", RFC 141 (NIC 6726), RAND, 29 April 1971.

Bhushan, Abhay, et al, "The File Transfer Protocol", RFC 172 (NIC 6794), MIT-Project MAC, 23 June 1971.

Braden, Bob, "Comments on DTP and FTP Proposals", RFC 238 (NIC 7663), UCLA/CCN, 29 September 1971.

Bhushan, Abhay, et al, "The File Transfer Protocol", RFC 265 (NIC 7813), MIT-Project MAC, 17 November 1971.

McKenzie, Alex, "A Suggested Addition to File Transfer Protocol", RFC 281 (NIC 8163), BBN, 8 December 1971.

Bhushan, Abhay, "The Use of "Set Data Type" Transaction in File Transfer Protocol", RFC 294 (NIC 8304), MIT-Project MAC, 25 January 1972.

Bhushan, Abhay, "The File Transfer Protocol", RFC 354 (NIC 10596), MIT-Project MAC, 8 July 1972.

Bhushan, Abhay, "Comments on the File Transfer Protocol (RFC 354)", RFC 385 (NIC 11357), MIT-Project MAC, 18 August 1972.

Hicks, Greg, "User FTP Documentation", RFC 412 (NIC 12404), Utah, 27 November 1972.

Bhushan, Abhay, "File Transfer Protocol (FTP) Status and Further Comments", RFC 414 (NIC 12406), MIT-Project MAC, 20 November 1972.

Braden, Bob, "Comments on File Transfer Protocol", RFC 430 (NIC 13299), UCLA/CCN, 7 February 1973.

Thomas, Bob, and Bob Clements, "FTP Server-Server Interaction", RFC 436 (NIC 13770), BBN, 15 January 1973.

Braden, Bob, "Print Files in FTP", RFC 448 (NIC 13299), UCLA/CCN, 27 February 1973.

McKenzie, Alex, "File Transfer Protocol", RFC 454 (NIC 14333), BBN, 16 February 1973.

Bressler, Bob, and Bob Thomas, "Mail Retrieval via FTP", RFC 458 (NIC 14378), BBN-NET and BBN-TENEX, 20 February 1973.

Neigus, Nancy, "File Transfer Protocol", RFC 542 (NIC 17759), BBN, 12 July 1973.

Krilanovich, Mark, and George Gregg, "Comments on the File Transfer Protocol", RFC 607 (NIC 21255), UCSB, 7 January 1974.

Pogran, Ken, and Nancy Neigus, "Response to RFC 607 - Comments on the File Transfer Protocol", RFC 614 (NIC 21530), BBN, 28 January 1974.

Krilanovich, Mark, George Gregg, Wayne Hathaway, and Jim White, "Comments on the File Transfer Protocol", RFC 624 (NIC 22054), UCSB, Ames Research Center, SRI-ARC, 28 February 1974.

Bhushan, Abhay, "FTP Comments and Response to RFC 430", RFC 463 (NIC 14573), MIT-DMCG, 21 February 1973.

Braden, Bob, "FTP Data Compression", RFC 468 (NIC 14742), UCLA/CCN, 8 March 1973.

Bhushan, Abhay, "FTP and Network Mail System", RFC 475 (NIC 14919), MIT-DMCG, 6 March 1973.

Bressler, Bob, and Bob Thomas "FTP Server-Server Interaction - II", RFC 478 (NIC 14947), BBN-NET and BBN-TENEX, 26 March 1973.

White, Jim, "Use of FTP by the NIC Journal", RFC 479 (NIC 14948), SRI-ARC, 8 March 1973.

White, Jim, "Host-Dependent FTP Parameters", RFC 480 (NIC 14949), SRI-ARC, 8 March 1973.

Padlipsky, Mike, "An FTP Command-Naming Problem", RFC 506 (NIC 16157), MIT-Multics, 26 June 1973.

Day, John, "Memo to FTP Group (Proposal for File Access Protocol)", RFC 520 (NIC 16819), Illinois, 25 June 1973.

Merryman, Robert, "The UCSD-CC Server-FTP Facility", RFC 532 (NIC 17451), UCSD-CC, 22 June 1973.

Braden, Bob, "TENEX FTP Problem", RFC 571 (NIC 18974), UCLA/CCN, 15 November 1973.

McKenzie, Alex, and Jon Postel, "Telnet and FTP Implementation - Schedule Change", RFC 593 (NIC 20615), BBN and MITRE, 29 November 1973.

Sussman, Julie, "FTP Error Code Usage for More Reliable Mail Service", RFC 630 (NIC 30237), BBN, 10 April 1974.

Postel, Jon, "Revised FTP Reply Codes", RFC 640 (NIC 30843), UCLA/NMC, 5 June 1974.

Harvey, Brian, "Leaving Well Enough Alone", RFC 686 (NIC 32481), SU-AI, 10 May 1975.

Harvey, Brian, "One More Try on the FTP", RFC 691 (NIC 32700), SU-AI, 28 May 1975.

Lieb, J., "CWD Command of FTP", RFC 697 (NIC 32963), 14 July 1975.

Harrenstien, Ken, "FTP Extension: XSEN", RFC 737 (NIC 42217), SRI-KL, 31 October 1977.

Harrenstien, Ken, "FTP Extension: XRSQ/XRCP", RFC 743 (NIC 42758), SRI-KL, 30 December 1977.

Lebling, P. David, "Survey of FTP Mail and MLFL", RFC 751, MIT, 10 December 1978.

Postel, Jon, "File Transfer Protocol Specification", RFC 765, ISI, June 1980.

Mankins, David, Dan Franklin, and Buzz Owen, "Directory Oriented FTP Commands", RFC 776, BBN, December 1980.

Padlipsky, Michael, "FTP Unique-Named Store Command", RFC 949, MITRE, July 1985.

REFERENCES

[1] Feinler, Elizabeth, "Internet Protocol Transition Workbook", Network Information Center, SRI International, March 1982.

[2] Postel, Jon, "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, DARPA, September 1981.

[3] Postel, Jon, and Joyce Reynolds, "Telnet Protocol Specification", RFC 854, ISI, May 1983.

[4] Reynolds, Joyce, and Jon Postel, "Assigned Numbers", RFC 943, ISI, April 1985.

RFC 821

SIMPLE MAIL TRANSFER PROTOCCL

Jonathan B. Postel

August 1982

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

(213) 822-1511

RFC 821 August 1982
Simple Mail Transfer Protocol

TABLE OF CONTENTS

Network Working Group
Request for Comments: DRAFT
Replaces: RFC 788, 780, 772

J. Postel
ISI
August 1982

SIMPLE MAIL TRANSFER PROTOCOL

1. INTRODUCTION

The objective of Simple Mail Transfer Protocol (SMTP) is to transfer mail reliably and efficiently.

SMTP is independent of the particular transmission subsystem and requires only a reliable ordered data stream channel. Appendices A, B, C, and D describe the use of SMTP with various transport services. A Glossary provides the definitions of terms as used in this document.

An important feature of SMTP is its capability to relay mail across transport service environments. A transport service provides an interprocess communication environment (IPCE). An IPCE may cover one network, several networks, or a subset of a network. It is important to realize that transport systems (or IPCEs) are not one-to-one with networks. A process can communicate directly with another process through any mutually known IPCE. Mail is an application or use of interprocess communication. Mail can be communicated between processes in different IPCEs by relaying through a process connected to two (or more) IPCEs. More specifically, mail can be relayed between hosts on different transport systems by a host on both transport systems.

2. THE SMTP MODEL

The SMTP design is based on the following model of communication: as the result of a user mail request, the sender-SMTP establishes a two-way transmission channel to a receiver-SMTP. The receiver-SMTP may be either the ultimate destination or an intermediate. SMTP commands are generated by the sender-SMTP and sent to the receiver-SMTP. SMTP replies are sent from the receiver-SMTP to the sender-SMTP in response to the commands.

Once the transmission channel is established, the SMTP-sender sends a MAIL command indicating the sender of the mail. If the SMTP-receiver can accept mail it responds with an OK reply. The SMTP-sender then sends a RCPT command identifying a recipient of the mail. If the SMTP-receiver can accept mail for that recipient it responds with an OK reply; if not, it responds with a reply rejecting that recipient (but not the whole mail transaction). The SMTP-sender and SMTP-receiver may negotiate several recipients. When the recipients have been negotiated the SMTP-sender sends the mail data, terminating with a special sequence. If the SMTP-receiver successfully processes the mail data it responds with an OK reply. The dialog is purposely lock-step, one-at-a-time.

Model for SMTP Use

Figure 1

The SMTP provides mechanisms for the transmission of mail; directly from the sending user's host to the receiving user's host when the

two host are connected to the same transport service, or via one or more relay SMTP-servers when the source and destination hosts are not connected to the same transport service.

To be able to provide the relay capability the SMTP-server must be supplied with the name of the ultimate destination host as well as the destination mailbox name.

The argument to the MAIL command is a reverse-path, which specifies who the mail is from. The argument to the RCPT command is a forward-path, which specifies who the mail is to. The forward-path is a source route, while the reverse-path is a return route (which may be used to return a message to the sender when an error occurs with a relayed message).

When the same message is sent to multiple recipients the SMTP encourages the transmission of only one copy of the data for all the recipients at the same destination host.

The mail commands and replies have a rigid syntax. Replies also have a numeric code. In the following, examples appear which use actual commands and replies. The complete lists of commands and replies appears in Section 4 on specifications.

Commands and replies are not case sensitive. That is, a command or reply word may be upper case, lower case, or any mixture of upper and lower case. Note that this is not true of mailbox user names. For some hosts the user name is case sensitive, and SMTP implementations must take case to preserve the case of user names as they appear in mailbox arguments. Host names are not case sensitive.

Commands and replies are composed of characters from the ASCII character set [1]. When the transport service provides an 8-bit byte (octet) transmission channel, each 7-bit character is transmitted right justified in an octet with the high order bit cleared to zero.

When specifying the general form of a command or reply, an argument (or special symbol) will be denoted by a meta-linguistic variable (or constant), for example, "<string>" or "<reverse-path>". Here the angle brackets indicate these are meta-linguistic variables. However, some arguments use the angle brackets literally. For example, an actual reverse-path is enclosed in angle brackets, i.e., "<John.Smith@USC-ISI.ARPA>" is an instance of <reverse-path> (the angle brackets are actually transmitted in the command or reply).

3. THE SMTP PROCEDURES

This section presents the procedures used in SMTP in several parts. First comes the basic mail procedure defined as a mail transaction. Following this are descriptions of forwarding mail, verifying mailbox names and expanding mailing lists, sending to terminals instead of or in combination with mailboxes, and the opening and closing exchanges. At the end of this section are comments on relaying, a note on mail domains, and a discussion of changing roles. Throughout this section are examples of partial command and reply sequences, several complete scenarios are presented in Appendix F.

3.1. MAIL

There are three steps to SMTP mail transactions. The transaction is started with a MAIL command which gives the sender identification. A series of one or more RCPT commands follows giving the receiver information. Then a DATA command gives the mail data. And finally, the end of mail data indicator confirms the transaction.

The first step in the procedure is the MAIL command. The <reverse-path> contains the source mailbox.

MAIL <SP> FROM:<reverse-path> <CRLF>

This command tells the SMTP-receiver that a new mail transaction is starting and to reset all its state tables and buffers, including any recipients or mail data. It gives the reverse-path which can be used to report errors. If accepted, the receiver-SMTP returns a 250 OK reply.

The <reverse-path> can contain more than just a mailbox. The <reverse-path> is a reverse source routing list of hosts and source mailbox. The first host in the <reverse-path> should be the host sending this command.

The second step in the procedure is the RCPT command.

RCPT <SP> TO:<forward-path> <CRLF>

This command gives a forward-path identifying one recipient. If accepted, the receiver-SMTP returns a 250 OK reply, and stores the forward-path. If the recipient is unknown the receiver-SMTP returns a 550 Failure reply. This second step of the procedure can be repeated any number of times.

The <forward-path> can contain more than just a mailbox. The <forward-path> is a source routing list of hosts and the destination mailbox. The first host in the <forward-path> should be the host receiving this command.

The third step in the procedure is the DATA command.

DATA <CRLF>

If accepted, the receiver-SMTP returns a 354 Intermediate reply and considers all succeeding lines to be the message text. When the end of text is received and stored the SMTP-receiver sends a 250 OK reply.

Since the mail data is sent on the transmission channel the end of the mail data must be indicated so that the command and reply dialog can be resumed. SMTP indicates the end of the mail data by sending a line containing only a period. A transparency procedure is used to prevent this from interfering with the user's text (see Section 4.5.2).

Please note that the mail data includes the memo header items such as Date, Subject, To, Cc, From [2].

The end of mail data indicator also confirms the mail transaction and tells the receiver-SMTP to now process the stored recipients and mail data. If accepted, the receiver-SMTP returns a 250 OK reply. The DATA command should fail only if the mail transaction was incomplete (for example, no recipients), or if resources are not available.

The above procedure is an example of a mail transaction. These commands must be used only in the order discussed above. Example 1 (below) illustrates the use of these commands in a mail transaction.

Example of the SMTP Procedure

This SMTP example shows mail sent by Smith at host Alpha.ARPA, to Jones, Green, and Brown at host Beta.ARPA. Here we assume that host Alpha contacts host Beta directly.

```
S: MAIL FROM:<Smith@Alpha.ARPA>
R: 250 OK

S: RCPT TO:<Jones@Beta.ARPA>
R: 250 OK

S: RCPT TO:<Green@Beta.ARPA>
R: 550 No such user here

S: RCPT TO:<Brown@Beta.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: <CRLF>.<CRLF>
R: 250 OK
```

The mail has now been accepted for Jones and Brown. Green did not have a mailbox at host Beta.

Example 1

3.2. FORWARDING

There are some cases where the destination information in the <forward-path> is incorrect, but the receiver-SMTP knows the correct destination. In such cases, one of the following replies should be used to allow the sender to contact the correct destination.

251 User not local; will forward to <forward-path>

This reply indicates that the receiver-SMTP knows the user's mailbox is on another host and indicates the correct forward-path to use in the future. Note that either the host or user or both may be different. The receiver takes responsibility for delivering the message.

551 User not local; please try <forward-path>

This reply indicates that the receiver-SMTP knows the user's mailbox is on another host and indicates the correct forward-path to use. Note that either the host or user or both may be different. The receiver refuses to accept mail for this user, and the sender must either redirect the mail according to the information provided or return an error response to the originating user.

Example 2 illustrates the use of these responses.

Example of Forwarding

Either

S: RCPT TO:<Postel@USC-ISI.ARPA>
R: 251 User not local; will forward to <Postel@USC-ISIF.ARPA>

Or

S: RCPT TO:<Paul@USC-ISIB.ARPA>
R: 551 User not local; please try <Mockapetris@USC-ISIF.ARPA>

Example 2

3.3. VERIFYING AND EXPANDING

SMTP provides as additional features, commands to verify a user name or expand a mailing list. This is done with the VRFY and EXPN commands, which have character string arguments. For the VRFY command, the string is a user name, and the response may include the full name of the user and must include the mailbox of the user. For the EXPN command, the string identifies a mailing list, and the multiline response may include the full name of the users and must give the mailboxes on the mailing list.

"User name" is a fuzzy term and used purposely. If a host implements the VRFY or EXPN commands then at least local mailboxes must be recognized as "user names". If a host chooses to recognize other strings as "user names" that is allowed.

In some hosts the distinction between a mailing list and an alias for a single mailbox is a bit fuzzy, since a common data structure may hold both types of entries, and it is possible to have mailing lists of one mailbox. If a request is made to verify a mailing list a positive response can be given if on receipt of a message so addressed it will be delivered to everyone on the list, otherwise an error should be reported (e.g., "550 That is a mailing list, not a user"). If a request is made to expand a user name a positive response can be formed by returning a list containing one name, or an error can be reported (e.g., "550 That is a user name, not a mailing list").

In the case of a multiline reply (normal for EXPN) exactly one mailbox is to be specified on each line of the reply. In the case of an ambiguous request, for example, "VRFY Smith", where there are two Smith's the response must be "553 User ambiguous".

The case of verifying a user name is straightforward as shown in example 3.

Example of Verifying a User Name

Either

S: VRFY Smith
R: 250 Fred Smith <Smith@USC-ISIF.ARPA>

Or

S: VRFY Smith
R: 251 User not local; will forward to <Smith@USC-ISIQ.ARPA>

Or

S: VRFY Jones
R: 550 String does not match anything.

Or

S: VRFY Jones
R: 551 User not local; please try <Jones@USC-ISIQ.ARPA>

Or

S: VRFY Gourzenkyinplatz
R: 553 User ambiguous.

Example 3

The case of expanding a mailbox list requires a multiline reply as shown in example 4.

Example of Expanding a Mailing List

Either

```
S: EXPN Example-People
R: 250-Jon Postel <Postel@USC-ISIF.ARPA>
R: 250-Fred Fonebone <Fonebone@USC-ISIQ.ARPA>
R: 250-Sam Q. Smith <SQSmith@USC-ISIQ.ARPA>
R: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
R: 250-<joe@foo-unix.ARPA>
R: 250 <xyz@bar-unix.ARPA>
```

Or

```
S: EXPN Executive-Washroom-List
R: 550 Access Denied to You.
```

Example 4

The character string arguments of the VRFY and EXPN commands cannot be further restricted due to the variety of implementations of the user name and mailbox list concepts. On some systems it may be appropriate for the argument of the EXPN command to be a file name for a file containing a mailing list, but again there is a variety of file naming conventions in the Internet.

The VRFY and EXPN commands are not included in the minimum implementation (Section 4.5.1), and are not required to work across relays when they are implemented.

3.4. SENDING AND MAILING

The main purpose of SMTP is to deliver messages to user's mailboxes. A very similar service provided by some hosts is to deliver messages to user's terminals (provided the user is active on the host). The delivery to the user's mailbox is called "mailing", the delivery to the user's terminal is called "sending". Because in many hosts the implementation of sending is nearly identical to the implementation of mailing these two functions are combined in SMTP. However the sending commands are not included in the required minimum implementation (Section 4.5.1). Users should have the ability to control the writing of messages on their terminals. Most hosts permit the users to accept or refuse such messages.

The following three command are defined to support the sending options. These are used in the mail transaction instead of the MAIL command and inform the receiver-SMTP of the special semantics of this transaction:

SEND <SP> FROM:<reverse-path> <CRLF>

The SEND command requires that the mail data be delivered to the user's terminal. If the user is not active (or not accepting terminal messages) on the host a 450 reply may returned to a RCPT command. The mail transaction is successful if the message is delivered the terminal.

SOML <SP> FROM:<reverse-path> <CRLF>

The Send Or MaiL command requires that the mail data be delivered to the user's terminal if the user is active (and accepting terminal messages) on the host. If the user is not active (or not accepting terminal messages) then the mail data is entered into the user's mailbox. The mail transaction is successful if the message is delivered either to the terminal or the mailbox.

SAML <SP> FROM:<reverse-path> <CRLF>

The Send And MaiL command requires that the mail data be delivered to the user's terminal if the user is active (and accepting terminal messages) on the host. In any case the mail data is entered into the user's mailbox. The mail transaction is successful if the message is delivered the mailbox.

The same reply codes that are used for the MAIL commands are used for these commands.

3.5. OPENING AND CLOSING

At the time the transmission channel is opened there is an exchange to ensure that the hosts are communicating with the hosts they think they are.

The following two commands are used in transmission channel opening and closing:

HELO <SP> <domain> <CRLF>

QUIT <CRLF>

In the HELO command the host sending the command identifies itself; the command may be interpreted as saying "Hello, I am <domain>".

Example of Connection Opening

R: 220 BBN-UNIX.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIF.ARPA
R: 250 BBN-UNIX.ARPA

Example 5

Example of Connection Closing

S: QUIT
R: 221 BBN-UNIX.ARPA Service closing transmission channel

Example 6

3.6. RELAYING

The forward-path may be a source route of the form "@ONE,@TWO:JOE@THREE", where ONE, TWO, and THREE are hosts. This form is used to emphasize the distinction between an address and a route. The mailbox is an absolute address, and the route is information about how to get there. The two concepts should not be confused.

Conceptually the elements of the forward-path are moved to the reverse-path as the message is relayed from one server-SMTP to another. The reverse-path is a reverse source route, (i.e., a source route from the current location of the message to the originator of the message). When a server-SMTP deletes its identifier from the forward-path and inserts it into the reverse-path, it must use the name it is known by in the environment it is sending into, not the environment the mail came from, in case the server-SMTP is known by different names in different environments.

If when the message arrives at an SMTP the first element of the forward-path is not the identifier of that SMTP the element is not deleted from the forward-path and is used to determine the next SMTP to send the message to. In any case, the SMTP adds its own identifier to the reverse-path.

Using source routing the receiver-SMTP receives mail to be relayed to another server-SMTP The receiver-SMTP may accept or reject the task of relaying the mail in the same way it accepts or rejects mail for a local user. The receiver-SMTP transforms the command arguments by moving its own identifier from the forward-path to the beginning of the reverse-path. The receiver-SMTP then becomes a sender-SMTP, establishes a transmission channel to the next SMTP in the forward-path, and sends it the mail.

The first host in the reverse-path should be the host sending the SMTP commands, and the first host in the forward-path should be the host receiving the SMTP commands.

Notice that the forward-path and reverse-path appear in the SMTP commands and replies, but not necessarily in the message. That is, there is no need for these paths and especially this syntax to appear in the "To:" , "From:", "CC:", etc. fields of the message header.

If a server-SMTP has accepted the task of relaying the mail and

later finds that the forward-path is incorrect or that the mail cannot be delivered for whatever reason, then it must construct an "undeliverable mail" notification message and send it to the originator of the undeliverable mail (as indicated by the reverse-path).

This notification message must be from the server-SMTP at this host. Of course, server-SMTPs should not send notification messages about problems with notification messages. One way to prevent loops in error reporting is to specify a null reverse-path in the MAIL command of a notification message. When such a message is relayed it is permissible to leave the reverse-path null. A MAIL command with a null reverse-path appears as follows:

```
MAIL FROM:<>
```

An undeliverable mail notification message is shown in example 7. This notification is in response to a message originated by JOE at HOSTW and sent via HOSTX to HOSTY with instructions to relay it on to HOSTZ. What we see in the example is the transaction between HOSTY and HOSTX, which is the first step in the return of the notification message.

Example Undeliverable Mail Notification Message

```
S: MAIL FROM:<>
R: 250 ok
S: RCPT TO:<@HOSTX.ARPA:JOE@HOSTW.ARPA>
R: 250 ok
S: DATA
R: 354 send the mail data, end with .
S: Date: 23 Oct 81 11:22:33
S: From: SMTP@HOSTY.ARPA
S: To: JOE@HOSTW.ARPA
S: Subject: Mail System Problem
S:
S:   Sorry JOE, your message to SAM@HOSTZ.ARPA lost.
S:   HOSTZ.ARPA said this:
S:    "550 No Such User"
S: .
R: 250 ok
```

Example 7

3.7. DOMAINS

Domains are a recently introduced concept in the ARPA Internet mail system. The use of domains changes the address space from a flat global space of simple character string host names to a hierarchically structured rooted tree of global addresses. The host name is replaced by a domain and host designator which is a sequence of domain element strings separated by periods with the understanding that the domain elements are ordered from the most specific to the most general.

For example, "USC-ISIF.ARPA", "Fred.Cambridge.UK", and "PC7.LCS.MIT.ARPA" might be host-and-domain identifiers.

Whenever domain names are used in SMTP only the official names are used, the use of nicknames or aliases is not allowed.

3.8. CHANGING ROLES

The TURN command may be used to reverse the roles of the two programs communicating over the transmission channel.

If program-A is currently the sender-SMTP and it sends the TURN command and receives an ok reply (250) then program-A becomes the receiver-SMTP.

If program-B is currently the receiver-SMTP and it receives the TURN command and sends an ok reply (250) then program-B becomes the sender-SMTP.

To refuse to change roles the receiver sends the 502 reply.

Please note that this command is optional. It would not normally be used in situations where the transmission channel is TCP. However, when the cost of establishing the transmission channel is high, this command may be quite useful. For example, this command may be useful in supporting be mail exchange using the public switched telephone system as a transmission channel, especially if some hosts poll other hosts for mail exchanges.

4. THE SMTP SPECIFICATIONS

4.1. SMTP COMMANDS

4.1.1. COMMAND SEMANTICS

The SMTP commands define the mail transfer or the mail system function requested by the user. SMTP commands are character strings terminated by <CRLF>. The command codes themselves are alphabetic characters terminated by <SP> if parameters follow and <CRLF> otherwise. The syntax of mailboxes must conform to receiver site conventions. The SMTP commands are discussed below. The SMTP replies are discussed in the Section 4.2.

A mail transaction involves several data objects which are communicated as arguments to different commands. The reverse-path is the argument of the MAIL command, the forward-path is the argument of the RCPT command, and the mail data is the argument of the DATA command. These arguments or data objects must be transmitted and held pending the confirmation communicated by the end of mail data indication which finalizes the transaction. The model for this is that distinct buffers are provided to hold the types of data objects, that is, there is a reverse-path buffer, a forward-path buffer, and a mail data buffer. Specific commands cause information to be appended to a specific buffer, or cause one or more buffers to be cleared.

HELLO (HELO)

This command is used to identify the sender-SMTP to the receiver-SMTP. The argument field contains the host name of the sender-SMTP.

The receiver-SMTP identifies itself to the sender-SMTP in the connection greeting reply, and in the response to this command.

This command and an OK reply to it confirm that both the sender-SMTP and the receiver-SMTP are in the initial state, that is, there is no transaction in progress and all state tables and buffers are cleared.

MAIL (MAIL)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more mailboxes. The argument field contains a reverse-path.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different). In some types of error reporting messages (for example, undeliverable mail notifications) the reverse-path may be null (see Example 7).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

RECIPIENT (RCPT)

This command is used to identify an individual recipient of the mail data; multiple recipients are specified by multiple use of this command.

The forward-path consists of an optional list of hosts and a required destination mailbox. When the list of hosts is present, it is a source route and indicates that the mail must be relayed to the next host on the list. If the receiver-SMTP does not implement the relay function it may user the same reply it would for an unknown local user (550).

When mail is relayed, the relay host must remove itself from the beginning forward-path and put itself at the beginning of the reverse-path. When mail reaches its ultimate destination (the forward-path contains only a destination mailbox), the receiver-SMTP inserts it into the destination mailbox in accordance with its host mail conventions.

For example, mail received at relay host A with arguments

FROM:<USERX@HOSTY.ARPA>
TO:<@HOSTA.ARPA,@HOSTB.ARPA:USERC@HOSTD.ARPA>

will be relayed on to host B with arguments

FROM:<@HOSTA.ARPA:USERX@HOSTY.ARPA>
TO:<@HOSTB.ARPA:USERC@HOSTD.ARPA>.

This command causes its forward-path argument to be appended to the forward-path buffer.

DATA (DATA)

The receiver treats the lines following the command as mail data from the sender. This command causes the mail data from this command to be appended to the mail data buffer. The mail data may contain any of the 128 ASCII character codes.

The mail data is terminated by a line containing only a period, that is the character sequence "<CRLF>.<CRLF>" (see Section 4.5.2 on Transparency). This is the end of mail data indication.

The end of mail data indication requires that the receiver must now process the stored mail transaction information. This processing consumes the information in the reverse-path buffer, the forward-path buffer, and the mail data buffer, and on the completion of this command these buffers are cleared. If the processing is successful the receiver must send an OK reply. If the processing fails completely the receiver must send a failure reply.

When the receiver-SMTP accepts a message either for relaying or for final delivery it inserts at the beginning of the mail data a time stamp line. The time stamp line indicates the identity of the host that sent the message, and the identity of the host that received the message (and is inserting this time stamp), and the date and time the message was received. Relayed messages will have multiple time stamp lines.

When the receiver-SMTP makes the "final delivery" of a message it inserts at the beginning of the mail data a

return path line. The return path line preserves the information in the <reverse-path> from the MAIL command. Here, final delivery means the message leaves the SMTP world. Normally, this would mean it has been delivered to the destination user, but in some cases it may be further processed and transmitted by another mail system.

> It is possible for the mailbox in the return path be different from the actual sender's mailbox, for example, if error responses are to be delivered a special error handling mailbox rather than the message senders.

The preceding two paragraphs imply that the final mail data will begin with a return path line, followed by one or more time stamp lines. These lines will be followed by the mail data header and body [2]. See Example 8.

Special mention is needed of the response and further action required when the processing following the end of mail data indication is partially successful. This could arise if after accepting several recipients and the mail data, the receiver-SMTP finds that the mail data can be successfully delivered to some of the recipients, but it cannot be to others (for example, due to mailbox space allocation problems). In such a situation, the response to the DATA command must be an OK reply. But, the receiver-SMTP must compose and send an "undeliverable mail" notification message to the originator of the message. Either a single notification which lists all of the recipients that failed to get the message, or separate notification messages must be sent for each failed recipient (see Example 7). All undeliverable mail notification messages are sent using the MAIL command (even if they result from processing a SEND, SOML, or SAML command).

Example of Return Path and Received Time Stamps

```
Return-Path: <@GHI.ARPA,@DEF.ARPA,@ABC.ARPA:JOE@ABC.ARPA>
Received: from GHI.ARPA by JKL.ARPA ; 27 Oct 81 15:27:39 PST
Received: from DEF.ARPA by GHI.ARPA ; 27 Oct 81 15:15:13 PST
Received: from ABC.ARPA by DEF.ARPA ; 27 Oct 81 15:01:59 PST
Date: 27 Oct 81 15:01:01 PST
From: JOE@ABC.ARPA
Subject: Improved Mailing System Installed
To: SAM@JKL.ARPA

This is to inform you that ...
```

Example 8

SEND (SEND)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals. The argument field contains a reverse-path. This command is successful if the message is delivered to a terminal.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

SEND OR MAIL (SOML)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals or

mailboxes. For each recipient the mail data is delivered to the recipient's terminal if the recipient is active on the host (and accepting terminal messages), otherwise to the recipient's mailbox. The argument field contains a reverse-path. This command is successful if the message is delivered to a terminal or the mailbox.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

SEND AND MAIL (SAML)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals and mailboxes. For each recipient the mail data is delivered to the recipient's terminal if the recipient is active on the host (and accepting terminal messages), and for all recipients to the recipient's mailbox. The argument field contains a reverse-path. This command is successful if the message is delivered to the mailbox.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different).

This command clears the reverse-path buffer, the

forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

RESET (RSET)

This command specifies that the current mail transaction is to be aborted. Any stored sender, recipients, and mail data must be discarded, and all buffers and state tables cleared. The receiver must send an OK reply.

VERIFY (VRFY)

This command asks the receiver to confirm that the argument identifies a user. If it is a user name, the full name of the user (if known) and the fully specified mailbox are returned.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

EXPAND (EXPN)

This command asks the receiver to confirm that the argument identifies a mailing list, and if so, to return the membership of that list. The full name of the users (if known) and the fully specified mailboxes are returned in a multiline reply.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

HELP (HELP)

This command causes the receiver to send helpful information to the sender of the HELP command. The command may take an argument (e.g., any command name) and return more specific information as a response.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the receiver send an OK reply.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

QUIT (QUIT)

This command specifies that the receiver must send an OK reply, and then close the transmission channel.

The receiver should not close the transmission channel until it receives and replies to a QUIT command (even if there was an error). The sender should not close the transmission channel until it send a QUIT command and receives the reply (even if there was an error response to a previous command). If the connection is closed prematurely the receiver should act as if a RSET command had been received (canceling any pending transaction, but not undoing any previously completed transaction), the sender should act as if the command or transaction in progress had received a temporary error (4xx).

TURN (TURN)

This command specifies that the receiver must either (1) send an OK reply and then take on the role of the sender-SMTP, or (2) send a refusal reply and retain the role of the receiver-SMTP.

If program-A is currently the sender-SMTP and it sends the TURN command and receives an OK reply (250) then program-A becomes the receiver-SMTP. Program-A is then in the initial state as if the transmission channel just opened, and it then sends the 220 service ready greeting.

If program-B is currently the receiver-SMTP and it receives the TURN command and sends an OK reply (250) then program-B becomes the sender-SMTP. Program-B is then in the initial state as if the transmission channel just opened, and it then expects to receive the 220 service ready greeting.

To refuse to change roles the receiver sends the 502 reply.

There are restrictions on the order in which these command may be used.

The first command in a session must be the HELO command. The HELO command may be used later in a session as well. If the HELO command argument is not acceptable a 501 failure reply must be returned and the receiver-SMTP must stay in the same state.

The NOOP, HELP, EXPN, and VRFY commands can be used at any time during a session.

The MAIL, SEND, SOML, or SAML commands begin a mail transaction. Once started a mail transaction consists of one of the transaction beginning commands, one or more RCPT commands, and a DATA command, in that order. A mail transaction may be aborted by the RSET command. There may be zero or more transactions in a session.

If the transaction beginning command argument is not acceptable a 501 failure reply must be returned and the receiver-SMTP must stay in the same state. If the commands in a transaction are out of order a 503 failure reply must be returned and the receiver-SMTP must stay in the same state.

The last command in a session must be the QUIT command. The QUIT command can not be used at any other time in a session.

4.1.2. COMMAND SYNTAX

The commands consist of a command code followed by an argument field. Command codes are four alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus, any of the following may represent the mail command:

MAIL Mail mail MaIl mAIl

This also applies to any symbols representing parameter values, such as "TO" or "to" for the forward-path. Command codes and the argument fields are separated by one or more spaces. However, within the reverse-path and forward-path arguments case is important. In particular, in some hosts the user "smith" is different from the user "Smith".

The argument field consists of a variable length character string ending with the character sequence <CRLF>. The receiver is to take no action until this sequence is received.

Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

The following are the SMTP commands:

HELO <SP> <domain> <CRLF>

MAIL <SP> FROM:<reverse-path> <CRLF>

RCPT <SP> TO:<forward-path> <CRLF>

DATA <CRLF>

RSET <CRLF>

SEND <SP> FROM:<reverse-path> <CRLF>

SOML <SP> FROM:<reverse-path> <CRLF>

SAML <SP> FROM:<reverse-path> <CRLF>

VRFY <SP> <string> <CRLF>

EXPN <SP> <string> <CRLF>

HELP [<SP> <string>] <CRLF>

NOOP <CRLF>

QUIT <CRLF>

TURN <CRLF>

The syntax of the above argument fields (using BNF notation where applicable) is given below. The "..." notation indicates that a field may be repeated one or more times.

```
<reverse-path> ::= <path>

<forward-path> ::= <path>

<path> ::= "<" [ <a-d-l> ":" ] <mailbox> ">"

<a-d-l> ::= <at-domain> | <at-domain> "," <a-d-l>

<at-domain> ::= "@" <domain>

<domain> ::=  <element> | <element> "." <domain>

<element> ::= <name> | "#" <number> | "[" <dotnum> "]"

<mailbox> ::= <local-part> "@" <domain>

<local-part> ::= <dot-string> | <quoted-string>

<name> ::= <a> <ldh-str> <let-dig>

<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>

<let-dig> ::= <a> | <d>

<let-dig-hyp> ::= <a> | <d> | "-"

<dot-string> ::= <string> | <string> "." <dot-string>

<string> ::= <char> | <char> <string>

<quoted-string> ::=  """ <qtext> """

<qtext> ::=  "\" <x> | "\" <x> <qtext> | <q> | <q> <qtext>

<char> ::= <c> | "\" <x>

<dotnum> ::= <snum> "." <snum> "." <snum> "." <snum>

<number> ::= <d> | <d> <number>

<CRLF> ::= <CR> <LF>
```

<CR> ::= the carriage return character (ASCII code 13)

<LF> ::= the line feed character (ASCII code 10)

<SP> ::= the space character (ASCII code 32)

<snum> ::= one, two, or three digits representing a decimal integer value in the range 0 through 255

<a> ::= any one of the 52 alphabetic characters A through Z in upper case and a through z in lower case

<c> ::= any one of the 128 ASCII characters, but not any <special> or <SP>

<d> ::= any one of the ten digits 0 through 9

<q> ::= any one of the 128 ASCII characters except <CR>, <LF>, quote ("), or backslash (\)

<x> ::= any one of the 128 ASCII characters (no exceptions)

<special> ::= "<" | ">" | "(" | ")" | "[" | "]" | "\" | "." | "," | ";" | ":" | "@" """ | the control characters (ASCII codes 0 through 31 inclusive and 127)

Note that the backslash, "\", is a quote character, which is used to indicate that the next character is to be used literally (instead of its normal interpretation). For example, "Joe\,Smith" could be used to indicate a single nine character user field with comma being the fourth character of the field.

Hosts are generally known by names which are translated to addresses in each host. Note that the name elements of domains are the official names -- no use of nicknames or aliases is allowed.

Sometimes a host is not known to the translation function and communication is blocked. To bypass this barrier two numeric forms are also allowed for host "names". One form is a decimal integer prefixed by a pound sign, "#", which indicates the number is the address of the host. Another form is four small decimal integers separated by dots and enclosed by brackets, e.g., "[123.255.37.2]", which indicates a 32-bit ARPA Internet Address in four 8-bit fields.

The time stamp line and the return path line are formally defined as follows:

<return-path-line> ::= "Return-Path:" <SP><reverse-path><CRLF>

<time-stamp-line> ::= "Received:" <SP> <stamp> <CRLF>

<stamp> ::= <from-domain> <by-domain> <opt-info> ";"
<daytime>

<from-domain> ::= "FROM" <SP> <domain> <SP>

<by-domain> ::= "BY" <SP> <domain> <SP>

<opt-info> ::= [<via>] [<with>] [<id>] [<for>]

<via> ::= "VIA" <SP> <link> <SP>

<with> ::= "WITH" <SP> <protocol> <SP>

<id> ::= "ID" <SP> <string> <SP>

<for> ::= "FOR" <SP> <path> <SP>

<link> ::= The standard names for links are registered with the Network Information Center.

<protocol> ::= The standard names for protocols are registered with the Network Information Center.

<daytime> ::= <SP> <date> <SP> <time>

<date> ::= <dd> <SP> <mon> <SP> <yy>

<time> ::= <hh> ":" <mm> ":" <ss> <SP> <zone>

<dd> ::= the one or two decimal integer day of the month in the range 1 to 31.

<mon> ::= "JAN" | "FEB" | "MAR" | "APR" | "MAY" | "JUN" |
"JUL" | "AUG" | "SEP" | "OCT" | "NOV" | "DEC"

<yy> ::= the two decimal integer year of the century in the range 00 to 99.

<hh> ::= the two decimal integer hour of the day in the range 00 to 24.

<mm> ::= the two decimal integer minute of the hour in the range 00 to 59.

<ss> ::= the two decimal integer second of the minute in the range 00 to 59.

<zone> ::= "UT" for Universal Time (the default) or other time zone designator (as in [2]).

Return Path Example

Return-Path: <@CHARLIE.ARPA,@BAKER.ARPA:JOE@ABLE.ARPA>

Example 9

Time Stamp Line Example

Received: FROM ABC.ARPA BY XYZ.ARPA ; 22 OCT 81 09:23:59 PDT

Received: from ABC.ARPA by XYZ.ARPA via TELENET with X25
id M12345 for Smith@PDQ.ARPA ; 22 OCT 81 09:23:59 PDT

Example 10

4.2. SMTP REPLIES

Replies to SMTP commands are devised to ensure the synchronization of requests and actions in the process of mail transfer, and to guarantee that the sender-SMTP always knows the state of the receiver-SMTP. Every command must generate exactly one reply.

The details of the command-reply sequence are made explicit in Section 5.3 on Sequencing and Section 5.4 State Diagrams.

An SMTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is meant for the human user. It is intended that the three digits contain enough encoded information that the sender-SMTP need not examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be receiver-dependent and context dependent, so there are likely to be varying texts for each reply code. A discussion of the theory of reply codes is given in Appendix E. Formally, a reply is defined to be the sequence: a three-digit code, <SP>, one line of text, and <CRLF>, or a multiline reply (as defined in Appendix E). Only the EXPN and HELP commands are expected to result in multiline replies in normal circumstances, however multiline replies are allowed for any command.

4.2.1. REPLY CODES BY FUNCTION GROUPS

500 Syntax error, command unrecognized
[This may include errors such as command line too long]
501 Syntax error in parameters or arguments
502 Command not implemented
503 Bad sequence of commands
504 Command parameter not implemented

211 System status, or system help reply
214 Help message
[Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]

220 <domain> Service ready
221 <domain> Service closing transmission channel
421 <domain> Service not available,
closing transmission channel
[This may be a reply to any command if the service knows it must shut down]

250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>
450 Requested mail action not taken: mailbox unavailable
[E.g., mailbox busy]
550 Requested action not taken: mailbox unavailable
[E.g., mailbox not found, no access]
451 Requested action aborted: error in processing
551 User not local; please try <forward-path>
452 Requested action not taken: insufficient system storage
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed
[E.g., mailbox syntax incorrect]
354 Start mail input; end with <CRLF>.<CRLF>
554 Transaction failed

4.2.2. NUMERIC ORDER LIST OF REPLY CODES

211 System status, or system help reply
214 Help message
[Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]
220 <domain> Service ready
221 <domain> Service closing transmission channel
250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>

354 Start mail input; end with <CRLF>.<CRLF>

421 <domain> Service not available,
closing transmission channel
[This may be a reply to any command if the service knows it must shut down]
450 Requested mail action not taken: mailbox unavailable
[E.g., mailbox busy]
451 Requested action aborted: local error in processing
452 Requested action not taken: insufficient system storage

500 Syntax error, command unrecognized
[This may include errors such as command line too long]
501 Syntax error in parameters or arguments
502 Command not implemented
503 Bad sequence of commands
504 Command parameter not implemented
550 Requested action not taken: mailbox unavailable
[E.g., mailbox not found, no access]
551 User not local; please try <forward-path>
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed
[E.g., mailbox syntax incorrect]
554 Transaction failed

4.3. SEQUENCING OF COMMANDS AND REPLIES

The communication between the sender and receiver is intended to be an alternating dialogue, controlled by the sender. As such, the sender issues a command and the receiver responds with a reply. The sender must wait for this response before sending further commands.

One important reply is the connection greeting. Normally, a receiver will send a 220 "Service ready" reply when the connection is completed. The sender should wait for this greeting message before sending any commands.

Note: all the greeting type replies have the official name of the server host as the first word following the reply code.

For example,

220 <SP> USC-ISIF.ARPA <SP> Service ready <CRLF>

The table below lists alternative success and failure replies for each command. These must be strictly adhered to; a receiver may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

COMMAND-REPLY SEQUENCES

Each command is listed with its possible replies. The prefixes used before the possible replies are "P" for preliminary (not used in SMTP), "I" for intermediate, "S" for success, "F" for failure, and "E" for error. The 421 reply (service not available, closing transmission channel) may be given to any command if the SMTP-receiver knows it must shut down. This listing forms the basis for the State Diagrams in Section 4.4.

```
CONNECTION ESTABLISHMENT
   S: 220
   F: 421
HELO
   S: 250
   E: 500, 501, 504, 421
MAIL
   S: 250
   F: 552, 451, 452
   E: 500, 501, 421
```

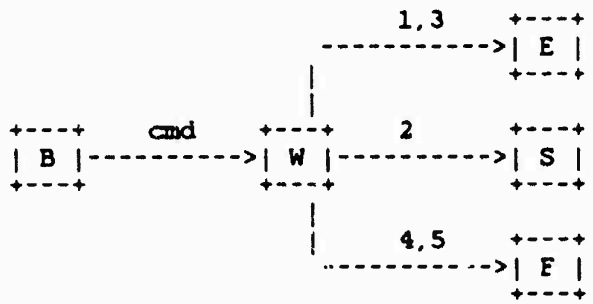
```
RCPT
   S: 250, 251
   F: 550, 551, 552, 553, 450, 451, 452
   E: 500, 501, 503, 421
DATA
   I: 354 -> data -> S: 250
                     F: 552, 554, 451, 452
   F: 451, 554
   E: 500, 501, 503, 421
RSET
   S: 250
   E: 500, 501, 504, 421
SEND
   S: 250
   F: 552, 451, 452
   E: 500, 501, 502, 421
SOML
   S: 250
   F: 552, 451, 452
   E: 500, 501, 502, 421
SAML
   S: 250
   F: 552, 451, 452
   E: 500, 501, 502, 421
VRFY
   S: 250, 251
   F: 550, 551, 553
   E: 500, 501, 502, 504, 421
EXPN
   S: 250
   F: 550
   E: 500, 501, 502, 504, 421
HELP
   S: 211, 214
   E: 500, 501, 502, 504, 421
NOOP
   S: 250
   E: 500, 421
QUIT
   S: 221
   E: 500
TURN
   S: 250
   F: 502
   E: 500, 503
```

4.4. STATE DIAGRAMS

Following are state diagrams for a simple-minded SMTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of SMTP commands. The command groupings were determined by constructing a model for each command and then collecting together the commands with structurally identical models.

For each command there are three possible outcomes: "success" (S), "failure" (F), and "error" (E). In the state diagrams below we use the symbol B for "begin", and the symbol W for "wait for reply".

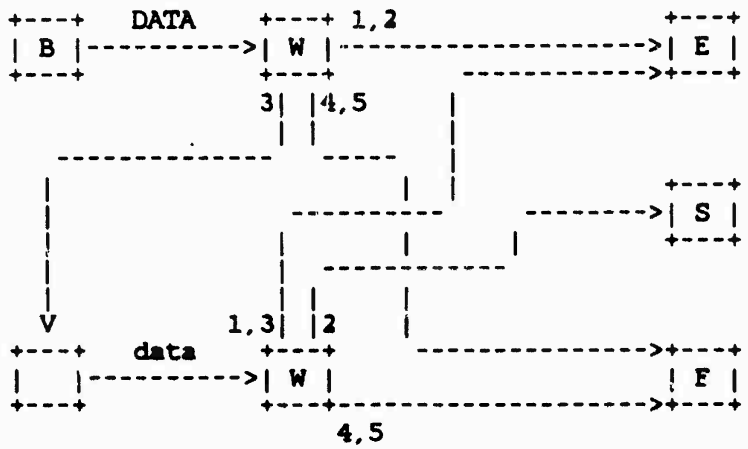
First, the diagram that represents most of the SMTP commands:



This diagram models the commands:

HELO, MAIL, RCPT, RSET, SEND, SOML, SAML, VRFY, EXPN, HELP, NOOP, QUIT, TURN.

A more complex diagram models the DATA command:



Note that the "data" here is a series of lines sent from the sender to the receiver with no response expected until the last line is sent.

4.5. DETAILS

4.5.1. MINIMUM IMPLEMENTATION

In order to make SMTP workable, the following minimum implementation is required for all receivers:

```
COMMANDS -- HELO
            MAIL
            RCPT
            DATA
            RSET
            NOOP
            QUIT
```

4.5.2. TRANSPARENCY

Without some provision for data transparency the character sequence "<CRLF>.<CRLF>" ends the mail text and cannot be sent by the user. In general, users are not aware of such "forbidden" sequences. To allow all user composed text to be transmitted transparently the following procedures are used.

1. Before sending a line of mail text the sender-SMTP checks the first character of the line. If it is a period, one additional period is inserted at the beginning of the line.

2. When a line of mail text is received by the receiver-SMTP it checks the line. If the line is composed of a single period it is the end of mail. If the first character is a period and there are other characters on the line, the first character is deleted.

The mail data may contain any of the 128 ASCII characters. All characters are to be delivered to the recipient's mailbox including format effectors and other control characters. If the transmission channel provides an 8-bit byte (octets) data stream, the 7-bit ASCII codes are transmitted right justified in the octets with the high order bits cleared to zero.

In some systems it may be necessary to transform the data as it is received and stored. This may be necessary for hosts that use a different character set than ASCII as their local character set, or that store data in records rather than

strings. If such transforms are necessary, they must be reversible -- especially if such transforms are applied to mail being relayed.

4.5.3. SIZES

There are several objects that have required minimum maximum sizes. That is, every implementation must be able to receive objects of at least these sizes, but must not send objects larger than these sizes.

```
**********************************************************
*                                                        *
*  TO THE MAXIMUM EXTENT POSSIBLE, IMPLEMENTATION        *
*  TECHNIQUES WHICH IMPOSE NO LIMITS ON THE LENGTH       *
*  OF THESE OBJECTS SHOULD BE USED.                      *
*                                                        *
**********************************************************
```

user

The maximum total length of a user name is 64 characters.

domain

The maximum total length of a domain name or number is 64 characters.

path

The maximum total length of a reverse-path or forward-path is 256 characters (including the punctuation and element separators).

command line

The maximum total length of a command line including the command word and the <CRLF> is 512 characters.

reply line

The maximum total length of a reply line including the reply code and the <CRLF> is 512 characters.

text line

The maximum total length of a text line including the <CRLF> is 1000 characters (but not counting the leading dot duplicated for transparency).

recipients buffer

The maximum total number of recipients that must be buffered is 100 recipients.

```
**********************************************************
*                                                        *
*  TO THE MAXIMUM EXTENT POSSIBLE, IMPLEMENTATION        *
*  TECHNIQUES WHICH IMPOSE NO LIMITS ON THE LENGTH       *
*  OF THESE OBJECTS SHOULD BE USED.                      *
*                                                        *
**********************************************************
```

Errors due to exceeding these limits may be reported by using the reply codes, for example:

500 Line too long.

501 Path too long

552 Too many recipients.

552 Too much mail data.

APPENDIX A

TCP Transport service

The Transmission Control Protocol [3] is used in the ARPA Internet, and in any network following the US DoD standards for internetwork protocols.

Connection Establishment

The SMTP transmission channel is a TCP connection established between the sender process port U and the receiver process port L. This single full duplex connection is used as the transmission channel. This protocol is assigned the service port 25 (31 octal), that is L=25.

Data Transfer

The TCP connection supports the transmission of 8-bit bytes. The SMTP data is 7-bit ASCII characters. Each character is transmitted as an 8-bit byte with the high-order bit cleared to zero.

APPENDIX C

NITS

The Network Independent Transport Service [6] may be used.

Connection Establishment

The SMTP transmission channel is established via NITS between the sender process and receiver process. The sender process executes the CONNECT primitive, and the waiting receiver process executes the ACCEPT primitive.

Data Transfer

The NITS connection supports the transmission of 8-bit bytes. The SMTP data is 7-bit ASCII characters. Each character is transmitted as an 8-bit byte with the high-order bit cleared to zero.

APPENDIX B

NCP Transport service

The ARPANET Host-to-Host Protocol [4] (implemented by the Network Control Program) may be used in the ARPANET.

Connection Establishment

The SMTP transmission channel is established via NCP between the sender process socket U and receiver process socket L. The Initial Connection Protocol [5] is followed resulting in a pair of simplex connections. This pair of connections is used as the transmission channel. This protocol is assigned the contact socket 25 (31 octal), that is L=25.

Data Transfer

The NCP data connections are established in 8-bit byte mode. The SMTP data is 7-bit ASCII characters. Each character is transmitted as an 8-bit byte with the high-order bit cleared to zero.

APPENDIX D

X.25 Transport service

It may be possible to use the X.25 service [7] as provided by the Public Data Networks directly, however, it is suggested that a reliable end-to-end protocol such as TCP be used on top of X.25 connections.

APPENDIX E

Theory of Reply Codes

The three digits of the reply each have a special significance. The first digit denotes whether the response is good, bad or incomplete. An unsophisticated sender-SMTP will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A sender-SMTP that wants to know approximately what kind of error occurred (e.g., mail system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information.

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The command has been accepted, but the requested action is being held in abeyance, pending confirmation of the information in this reply. The sender-SMTP should send another command specifying whether to continue or abort the action.

[Note: SMTP does not have any commands that allow this type of reply, and so does not have the continue or abort commands.]

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The sender-SMTP should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not occur. However, the error condition is temporary and the action may be requested again. The sender should

return to the beginning of the command sequence (if any). It is difficult to assign a meaning to "transient" when two different sites (receiver- and sender- SMTPs) must agree on the interpretation. Each reply in this category might have a different time value, but the sender-SMTP is encouraged to try again. A rule of thumb to determine if a reply fits into the 4yz or the 5yz category (see below) is that replies are 4yz if they can be repeated without any change in command form or in properties of the sender or receiver. (E.g., the command is repeated identically and the receiver does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not occur. The sender-SMTP is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct the sender-SMTP to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered the account status).

The second digit encodes responses in specific categories:

x0z Syntax -- These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, and unimplemented or superfluous commands.

x1z Information -- These are replies to requests for information, such as status or help.

x2z Connections -- These are replies referring to the transmission channel.

x3z Unspecified as yet.

x4z Unspecified as yet.

x5z Mail system -- These replies indicate the status of the receiver mail system vis-a-vis the requested transfer or other mail system action.

The third digit gives a finer gradation of meaning in each category specified by the second digit. The list of replies

illustrates this. Each reply text is recommended rather than mandatory, and may even change according to the command with which it is associated. On the other hand, the reply codes must strictly follow the specifications in this section. Receiver implementations should not invent new codes for slightly different situations from the ones described here, but rather adapt codes already defined.

For example, a command such as NOOP whose successful execution does not offer the sender-SMTP any new information will return a 250 reply. The response is 502 when the command requests an unimplemented non-site-specific action. A refinement of that is the 504 reply for a command that is implemented, but that requests an unimplemented parameter.

The reply text may be longer than a single line; in these cases the complete text must be marked so the sender-SMTP knows when it can stop reading the reply. This requires a special format to indicate a multiple line reply.

The format for multiline replies requires that every line, except the last, begin with the reply code, followed immediately by a hyphen, "-" (also known as minus), followed by text. The last line will begin with the reply code, followed immediately by <SP>, optionally some text, and <CRLF>.

For example:

```
123-First line
123-Second line
123-234 text beginning with numbers
123 The last line
```

In many cases the sender-SMTP then simply needs to search for the reply code followed by <SP> at the beginning of a line, and ignore all preceding lines. In a few cases, there is important data for the sender in the reply "text". The sender will know these cases from the current context.

APPENDIX F

Scenarios

This section presents complete scenarios of several types of SMTP sessions.

A Typical SMTP Transaction Scenario

This SMTP example shows mail sent by Smith at host USC-ISIF, to Jones, Green, and Brown at host BBN-UNIX. Here we assume that host USC-ISIF contacts host BBN-UNIX directly. The mail is accepted for Jones and Brown. Green does not have a mailbox at host BBN-UNIX.

```
R: 220 BBN-UNIX.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIF.ARPA
R: 250 BBN-UNIX.ARPA

S: MAIL FROM:<Smith@USC-ISIF.ARPA>
R: 250 OK

S: RCPT TO:<Jones@BBN-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<Green@BBN-UNIX.ARPA>
R: 550 No such user here

S: RCPT TO:<Brown@BBN-UNIX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 BBN-UNIX.ARPA Service closing transmission channel
```

Scenario 1

Aborted SMTP Transaction Scenario

```
R: 220 MIT-Multics.ARPA Simple Mail Transfer Service Ready
S: HELO ISI-VAXA.ARPA
R: 250 MIT-Multics.ARPA

S: MAIL FROM:<Smith@ISI-VAXA.ARPA>
R: 250 OK

S: RCPT TO:<Jones@MIT-Multics.ARPA>
R: 250 OK

S: RCPT TO:<Green@MIT-Multics.ARPA>
R: 550 No such user here

S: RSET
R: 250 OK

S: QUIT
R: 221 MIT-Multics.ARPA Service closing transmission channel
```

Scenario 2

Relayed Mail Scenario

Step 1 -- Source Host to Relay Host

```
R: 220 USC-ISIE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-AI.ARPA
R: 250 USC-ISIE.ARPA

S: MAIL FROM:<JQP@MIT-AI.ARPA>
R: 250 OK

S: RCPT TO:<@USC-ISIE.ARPA:Jones@BBN-VAX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Date: 2 Nov 81 22:33:44
S: From: John Q. Public <JQP@MIT-AI.ARPA>
S: Subject:  The Next Meeting of the Board
S: To: Jones@BBN-Vax.ARPA
S:
S: Bill:
S: The next meeting of the board of directors will be
S: on Tuesday.
S:                                                  John.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE.ARPA Service closing transmission channel
```

Step 2 -- Relay Host to Destination Host

```
R: 220 BBN-VAX.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIE.ARPA
R: 250 BBN-VAX.ARPA

S: MAIL FROM:<@USC-ISIE.ARPA:JQP@MIT-AI.ARPA>
R: 250 OK

S: RCPT TO:<Jones@BBN-VAX.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Received: from MIT-AI.ARPA by USC-ISIE.ARPA ;
   2 Nov 81 22:40:10 UT
S: Date: 2 Nov 81 22:33:44
S: From: John Q. Public <JQP@MIT-AI.ARPA>
S: Subject:  The Next Meeting of the Board
S: To: Jones@BBN-Vax.ARPA
S:
S: Bill:
S: The next meeting of the board of directors will be
S: on Tuesday.
S:                                                John.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE.ARPA Service closing transmission channel
```

Scenario 3

Verifying and Sending Scenario

```
R: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-MC.ARPA
R: 250 SU-SCORE.ARPA

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

S: SEND FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE.ARPA Service closing transmission channel
```

Scenario 4

Sending and Mailing Scenarios

First the user's name is verified, then an attempt is made to send to the user's terminal. When that fails, the messages is mailed to the user's mailbox.

```
R: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-MC.ARPA
R: 250 SU-SCORE.ARPA

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

S: SEND FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 450 User not active now

S: RSET
R: 250 OK

S: MAIL FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE.ARPA Service closing transmission channel
```

Scenario 5

Doing the preceding scenario more efficiently.

```
R: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
S: HELO MIT-MC.ARPA
R: 250 SU-SCORE.ARPA

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

S: SOML FROM:<EAK@MIT-MC.ARPA>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
R: 250 User not active now, so will do mail.

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE.ARPA Service closing transmission channel
```

Scenario 6

Mailing List Scenario

First each of two mailing lists are expanded in separate sessions with different hosts. Then the message is sent to everyone that appeared on either list (but no duplicates) via a relay host.

Step 1 -- Expanding the First List

```
R: 220 MIT-AI.ARPA Simple Mail Transfer Service Ready
S: HELO SU-SCORE.ARPA
R: 250 MIT-AI.ARPA

S: EXPN Example-People
R: 250-<ABC@MIT-MC.ARPA>
R: 250-Fred Fonebone <Fonebone@USC-ISIQ.ARPA>
R: 250-Xenon Y. Zither <XYZ@MIT-AI.ARPA>
R: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
R: 250-<joe@foo-unix.ARPA>
R: 250 <xyz@bar-unix.ARPA>

S: QUIT
R: 221 MIT-AI.ARPA Service closing transmission channel
```

Step 2 -- Expanding the Second List

```
R: 220 MIT-MC.ARPA Simple Mail Transfer Service Ready
S: HELO SU-SCORE.ARPA
R: 250 MIT-MC.ARPA

S: EXPN Interested-Parties
R: 250-Al Calico <ABC@MIT-MC.ARPA>
R: 250-<XYZ@MIT-AI.ARPA>
R: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
R: 250-<fred@BBN-UNIX.ARPA>
R: 250 <xyz@bar-unix.ARPA>

S: QUIT
R: 221 MIT-MC.ARPA Service closing transmission channel
```

Step 3 -- Mailing to All via a Relay Host

```
   R: 220 USC-ISIE.ARPA Simple Mail Transfer Service Ready
   S: HELO SU-SCORE.ARPA
   R: 250 USC-ISIE.ARPA

   S: MAIL FROM:<Account.Person@SU-SCORE.ARPA>
   R: 250 OK
   S: RCPT TO:<@USC-ISIE.ARPA:ABC@MIT-MC.ARPA>
   R: 250 OK
   S: RCPT TO:<@USC-ISIE.ARPA:Fonebone@USC-ISIQA.ARPA>
   R: 250 OK
   S: RCPT TO:<@USC-ISIE.ARPA:XYZ@MIT-AI.ARPA>
   R: 250 OK
   S: RCPT
       TO:<@USC-ISIE.ARPA,@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
   R: 250 OK
   S: RCPT TO:<@USC-ISIE.ARPA:joe@FOO-UNIX.ARPA>
   R: 250 OK
   S: RCPT TO:<@USC-ISIE.ARPA:xyz@BAR-UNIX.ARPA>
   R: 250 OK
   S: RCPT TO:<@USC-ISIE.ARPA:fred@BBN-UNIX.ARPA>
   R: 250 OK

   S: DATA
   R: 354 Start mail input; end with <CRLF>.<CRLF>
   S: Blah blah blah...
   S: ...etc. etc. etc.
   S: .
   R: 250 OK

   S: QUIT
   R: 221 USC-ISIE.ARPA Service closing transmission channel
```

Scenario 7

Forwarding Scenarios

```
R: 220 USC-ISIF.ARPA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX.ARPA
R: 250 USC-ISIF.ARPA

S: MAIL FROM:<mo@LBL-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<fred@USC-ISIF.ARPA>
R: 251 User not local; will forward to <Jones@USC-ISI.ARPA>

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIF.ARPA Service closing transmission channel
```

Scenario 8

Step 1 -- Trying the Mailbox at the First Host

```
R: 220 USC-ISIF.ARPA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX.ARPA
R: 250 USC-ISIF.ARPA

S: MAIL FROM:<mo@LBL-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<fred@USC-ISIF.ARPA>
R: 251 User not local; will forward to <Jones@USC-ISI.ARPA>

S: RSET
R: 250 OK

S: QUIT
R: 221 USC-ISIF.ARPA Service closing transmission channel
```

Step 2 -- Delivering the Mail at the Second Host

```
R: 220 USC-ISI.ARPA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX.ARPA
R: 250 USC-ISI.ARPA

S: MAIL FROM:<mo@LBL-UNIX.ARPA>
R: 250 OK

S: RCPT TO:<Jones@USC-ISI.ARPA>
R: OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISI.ARPA Service closing transmission channel
```

Scenario 9

Too Many Recipients Scenario

```
R: 220 BERKELEY.ARPA Simple Mail Transfer Service Ready
S: HELO USC-ISIF.ARPA
R: 250 BERKELEY.ARPA

S: MAIL FROM:<Postel@USC-ISIF.ARPA>
R: 250 OK

S: RCPT TO:<fabry@BERKELEY.ARPA>
R: 250 OK

S: RCPT TO:<eric@BERKELEY.ARPA>
R: 552 Recipient storage full, try again in another transaction

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: MAIL FROM:<Postel@USC-ISIF.ARPA>
R: 250 OK

S: RCPT TO:<eric@BERKELEY.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 BERKELEY.ARPA Service closing transmission channel
```

Scenario 10

Note that a real implementation must handle many recipients as specified in Section 4.5.3.

GLOSSARY

ASCII

American Standard Code for Information Interchange [1].

command

A request for a mail service action sent by the sender-SMTP to the receiver-SMTP.

domain

The hierarchially structured global character string address of a host computer in the mail system.

end of mail data indication

A special sequence of characters that indicates the end of the mail data. In particular, the five characters carriage return, line feed, period, carriage return, line feed, in that order.

host

A computer in the internetwork environment on which mailboxes or SMTP processes reside.

line

A a sequence of ASCII characters ending with a <CRLF>.

mail data

A sequence of ASCII characters of arbitrary length, which conforms to the standard set in the Standard for the Format of ARPA Internet Text Messages (RFC 822 [2]).

mailbox

A character string (address) which identifies a user to whom mail is to be sent. Mailbox normally consists of the host and user specifications. The standard mailbox naming convention is defined to be "user@domain". Additionally, the "container" in which mail is stored.

receiver-SMTP process

A process which transfers mail in cooperation with a sender-SMTP process. It waits for a connection to be established via the transport service. It receives SMTP commands from the sender-SMTP, sends replies, and performs the specified operations.

reply

A reply is an acknowledgment (positive or negative) sent from receiver to sender via the transmission channel in response to a command. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

sender-SMTP process

A process which transfers mail in cooperation with a receiver-SMTP process. A local language may be used in the user interface command/reply dialogue. The sender-SMTP initiates the transport service connection. It initiates SMTP commands, receives replies, and governs the transfer of mail.

session

The set of exchanges that occur while the transmission channel is open.

transaction

The set of exchanges required for one message to be transmitted for one or more recipients.

transmission channel

A full-duplex communication path between a sender-SMTP and a receiver-SMTP for the exchange of commands, replies, and mail text.

transport service

Any reliable stream-oriented data communication services. For example, NCP, TCP, NITS.

user

A human being (or a process on behalf of a human being) wishing to obtain mail transfer service. In addition, a recipient of computer mail.

word

A sequence of printing characters.

<CRLF>

The characters carriage return and line feed (in that order).

<SP>

The space character.

REFERENCES

[1] ASCII

ASCII, "USA Code for Information Interchange", United States of America Standards Institute, X3.4, 1968. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

[2] RFC 822

Crocker, D., "Standard for the Format of ARPA Internet Text Messages," RFC 822, Department of Electrical Engineering, University of Delaware, August 1982.

[3] TCP

Postel, J., ed., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, USC/Information Sciences Institute, NTIS AD Number A111091, September 1981. Also in: Feinler, E. and J. Postel, eds., "Internet Protocol Transition Workbook", SRI International, Menlo Park, California, March 1982.

[4] NCP

McKenzie,A., "Host/Host Protocol for the ARPA Network", NIC 8246, January 1972. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

[5] Initial Connection Protocol

Postel, J., "Official Initial Connection Protocol", NIC 7101, 11 June 1971. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

[6] NITS

PSS/SG3, "A Network Independent Transport Service", Study Group 3, The Post Office PSS Users Group, February 1980. Available from the DCPU, National Physical Laboratory, Teddington, UK.

[7] X.25

CCITT, "Recommendation X.25 - Interface Between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks," CCITT Orange Book, Vol. VIII.2, International Telephone and Telegraph Consultative Committee, Geneva, 1976.

Network Working Group
Request for Comments: 883

P. Mockapetris
ISI
November 1983

DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION

This memo discusses the implementation of domain name servers and resolvers, specifies the format of transactions, and discusses the use of domain names in the context of existing mail systems and other network software.

This memo assumes that the reader is familiar with RFC 882, "Domain Names - Concepts and Facilities" which discusses the basic principles of domain names and their use.

The algorithms and internal data structures used in this memo are offered as suggestions rather than requirements; implementers are free to design their own structures so long as the same external behavior is achieved.

***** WARNING *****

This RFC contains format specifications which are preliminary and are included for purposes of explanation only. Do not attempt to use this information for actual implementations.

RFC 883 November 1983
Domain Names - Implementation and Specification

INTRODUCTION

Overview

The goal of domain names is to provide a mechanism for naming resources in such a way that the names are usable in different hosts, networks, protocol families, internets, and administrative organizations.

From the user's point of view, domain names are useful as arguments to a local agent, called a resolver, which retrieves information associated with the domain name. Thus a user might ask for the host address or mail information associated with a particular domain name. To enable the user to request a particular type of information, an appropriate query type is passed to the resolver with the domain name. To the user, the domain tree is a single information space.

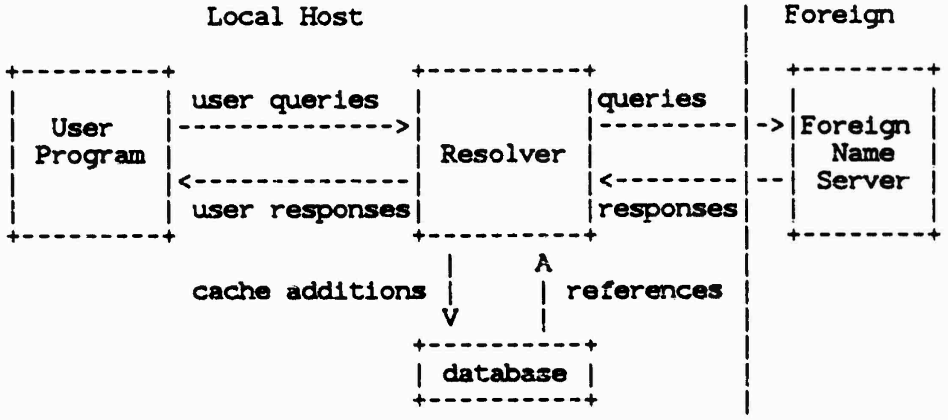
From the resolver's point of view, the database that makes up the domain space is distributed among various name servers. Different parts of the domain space are stored in different name servers, although a particular data item will usually be stored redundantly in two or more name servers. The resolver starts with knowledge of at least one name server. When the resolver processes a user query it asks a known name server for the information; in return, the resolver either receives the desired information or a referral to another name server. Using these referrals, resolvers learn the identities and contents of other name servers. Resolvers are responsible for dealing with the distribution of the domain space and dealing with the effects of name server failure by consulting redundant databases in other servers.

Name servers manage two kinds of data. The first kind of data held in sets called zones; each zone is the complete database for a particular subtree of the domain space. This data is called authoritative. A name server periodically checks to make sure that its zones are up to date, and if not obtains a new copy of updated zones from master files stored locally or in another name server. The second kind of data is cached data which was acquired by a local resolver. This data may be incomplete but improves the performance of the retrieval process when non-local data is repeatedly accessed. Cached data is eventually discarded by a timeout mechanism.

This functional structure isolates the problems of user interface, failure recovery, and distribution in the resolvers and isolates the database update and refresh problems in the name servers.

Implementation components

A host can participate in the domain name system in a number of ways, depending on whether the host runs programs that retrieve information from the domain system, name servers that answer queries from other hosts, or various combinations of both functions. The simplest, and perhaps most typical, configuration is shown below:



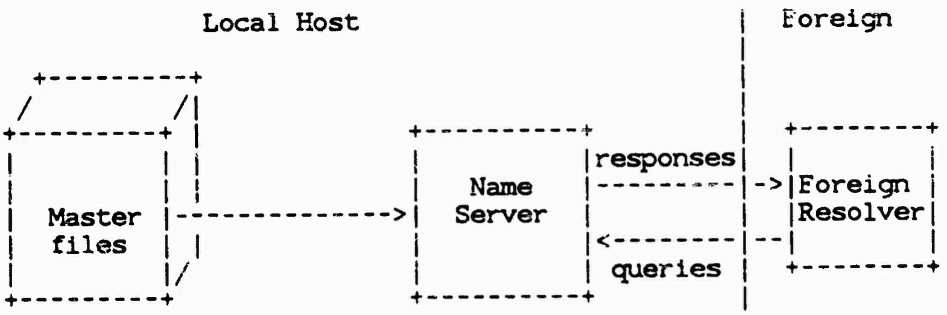
User programs interact with the domain name space through resolvers; the format of user queries and user responses is specific to the host and its operating system. User queries will typically be operating system calls, and the resolver and its database will be part of the host operating system. Less capable hosts may choose to implement the resolver as a subroutine to be linked in with every program that needs its services.

Resolvers answer user queries with information they acquire via queries to foreign name servers, and may also cache or reference domain information in the local database.

Note that the resolver may have to make several queries to several different foreign name servers to answer a particular user query, and hence the resolution of a user query may involve several network accesses and an arbitrary amount of time. The queries to foreign name servers and the corresponding responses have a standard format described in this memo, and may be datagrams.

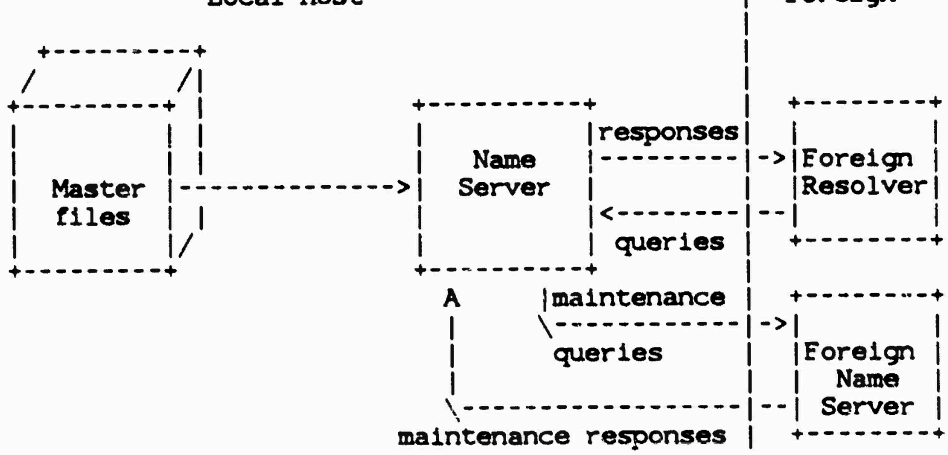
TABLE OF CONTENTS

Depending on its capabilities, a name server could be a stand alone program on a dedicated machine or a process or processes on a large timeshared host. A simple configuration might be:



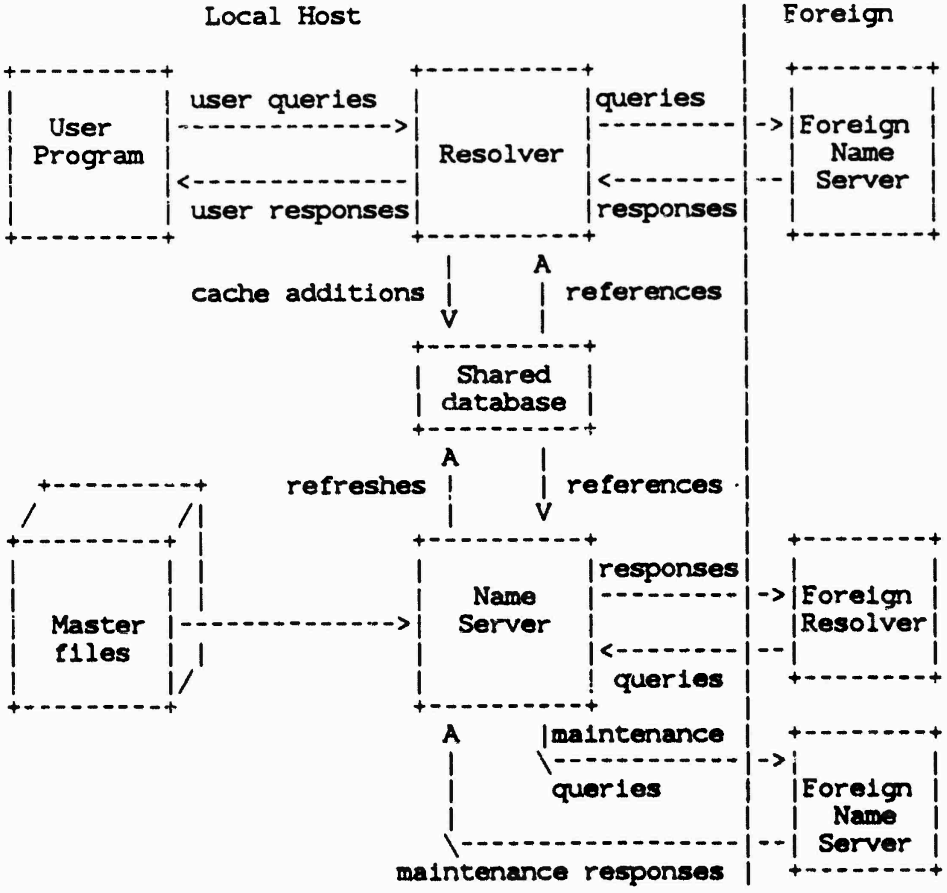
Here the name server acquires information about one or more zones by reading master files from its local file system, and answers queries about those zones that arrive from foreign resolvers.

A more sophisticated name server might acquire zones from foreign name servers as well as local master files. This configuration is shown below:



In this configuration, the name server periodically establishes a virtual circuit to a foreign name server to acquire a copy of a zone or to check that an existing copy has not changed. The messages sent for these maintenance activities follow the same form as queries and responses, but the message sequences are somewhat different.

The information flow in a host that supports all aspects of the domain name system is shown below:



The shared database holds domain space data for the local name server and resolver. The contents of the shared database will typically be a mixture of authoritative data maintained by the periodic refresh operations of the name server and cached data from previous resolver requests. The structure of the domain data and the necessity for synchronization between name servers and resolvers imply the general characteristics of this database, but the actual format is up to the local implementer. This memo suggests a multiple tree format.

This memo divides the implementation discussion into sections:

NAME SERVER TRANSACTIONS, which discusses the formats for name servers queries and the corresponding responses.

NAME SERVER MAINTENANCE, which discusses strategies, algorithms, and formats for maintaining the data residing in name servers. These services periodically refresh the local copies of zones that originate in other hosts.

RESOLVER ALGORITHMS, which discusses the internal structure of resolvers. This section also discusses data base sharing between a name server and a resolver on the same host.

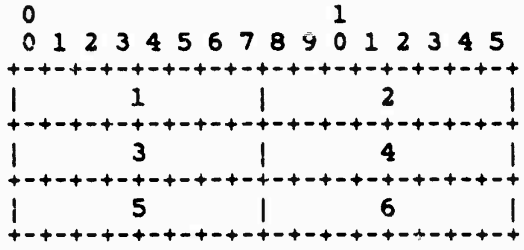
DOMAIN SUPPORT FOR MAIL, which discusses the use of the domain system to support mail transfer.

Conventions

The domain system has several conventions dealing with low-level, but fundamental, issues. While the implementer is free to violate these conventions WITHIN HIS OWN SYSTEM, he must observe these conventions in ALL behavior observed from other hosts.

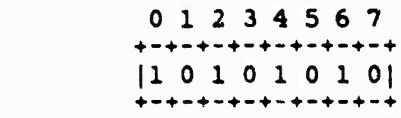
********** Data Transmission Order **********

The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they are numbered.



Transmission Order of Bytes

Whenever an octet represents a numeric quantity the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).



Significance of Bits

Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

********** Character Case **********

All comparisons between character strings (e.g. labels, domain names, etc.) are done in a case-insensitive manner.

When data enters the domain system, its original case should be preserved whenever possible. In certain circumstances this cannot be done. For example, if two domain names x.y and X.Y are entered into the domain database, they are interpreted as the same name, and hence may have a single representation. The basic rule is that case can be discarded only when data is used to define structure in a database, and two names are identical when compared in a case insensitive manner.

Loss of case sensitive data must be minimized. Thus while data for x.y and X.Y may both be stored under x.y, data for a.x and B.X can be stored as a.x and B.x, but not A.x, A.X, b.x, or b.X. In general, this prevents the first component of a domain name from loss of case information.

Systems administrators who enter data into the domain database should take care to represent the data they supply to the domain system in a case-consistent manner if their system is case-sensitive. The data distribution system in the domain system will ensure that consistent representations are preserved.

Design philosophy

The design presented in this memo attempts to provide a base which will be suitable for several existing networks. An equally important goal is to provide these services within a framework that is capable of adjustment to fit the evolution of services in early clients as well as to accommodate new networks.

Since it is impossible to predict the course of these developments, the domain system attempts to provide for evolution in the form of an extensible framework. This section describes the areas in which we expect to see immediate evolution.

DEFINING THE DATABASE

This memo defines methods for partitioning the database and data for host names, host addresses, gateway information, and mail support. Experience with this system will provide guidance for future additions.

While the present system allows for many new RR types, classes, etc., we feel that it is more important to get the basic services in operation than to cover an exhaustive set of information. Hence we have limited the data types to those we felt were essential, and would caution designers to avoid implementations which are based on the number of existing types and classes. Extensibility in this area is very important.

While the domain system provides techniques for partitioning the database, policies for administrating the orderly connection of separate domains and guidelines for constructing the data that makes up a particular domain will be equally important to the success of the system. Unfortunately, we feel that experience with prototype systems will be necessary before this question can be properly addressed. Thus while this memo has minimal discussion of these issues, it is a critical area for development.

TYING TOGETHER INTERNETS

Although it is very difficult to characterize the types of networks, protocols, and applications that will be clients of the domain system, it is very obvious that some of these applications will cross the boundaries of network and protocol. At the very least, mail is such a service.

Attempts to unify two such systems must deal with two major problems:

1. Differing formats for environment sensitive data. For example,

network addresses vary in format, and it is unreasonable to expect to enforce consistent conventions.

2. Connectivity may require intermediaries. For example, it is a frequent occurence that mail is sent between hosts that share no common protocol.

The domain system acknowledges that these are very difficult problems, and attempts to deal with both problems through its CLASS mechanism:

1. The CLASS field in RRs allows data to be tagged so that all programs in the domain system can identify the format in use.

2. The CLASS field allows the requestor to identify the format of data which can be understood by the requestor.

3. The CLASS field guides the search for the requested data.

The last point is central to our approach. When a query crosses protocol boundaries, it must be guided though agents capable of performing whatever translation is required. For example, when a mailer wants to identify the location of a mailbox in a portion of the domain system that doesn't have a compatible protocol, the query must be guided to a name server that can cross the boundary itself or form one link in a chain that can span the differences.

If query and response transport were the only problem, then this sort of problem could be dealt with in the name servers themselves. However, the applications that will use domain service have similar problems. For example, mail may need to be directed through mail gateways, and the characteristics of one of the environments may not permit frequent connectivity between name servers in all environments.

These problems suggest that connectivity will be achieved through a variety of measures:

Translation name servers that act as relays between different protocols.

Translation application servers that translate application level transactions.

Default database entries that route traffic through application level forwarders in ways that depend on the class of the requestor.

While this approach seems best given our current understanding of

the problem, we realize that the approach of using resource data that transcends class may be appropriate in future designs or applications. By not defining class to be directly related to protocol, network, etc., we feel that such services could be added by defining a new "universal" class, while the present use of class will provide immediate service.

This problem requires more thought and experience before solutions can be discovered. The concepts of CLASS, recursive servers and other mechanisms are intended as tools for acquiring experience and not as final solutions.

NAME SERVER TRANSACTIONS

Introduction

The primary purpose of name servers is to receive queries from resolvers and return responses. The overall model of this service is that a program (typically a resolver) asks the name server questions (queries) and gets responses that either answer the question or refer the questioner to another name server. Other functions related to name server database maintenance use similar procedures and formats and are discussed in a section later in this memo.

There are three kinds of queries presently defined:

1. Standard queries that ask for a specified resource attached to a given domain name.

2. Inverse queries that specify a resource and ask for a domain name that possesses that resource.

3. Completion queries that specify a partial domain name and a target domain and ask that the partial domain name be completed with a domain name close to the target domain

This memo uses an unqualified reference to queries to refer to either all queries or standard queries when the context is clear.

Query and response transport

Name servers and resolvers use a single message format for all communications. The message format consists of a variable-length octet string which includes binary values.

The messages used in the domain system are designed so that they can be carried using either datagrams or virtual circuits. To accommodate the datagram style, all responses carry the query as part of the response.

While the specification allows datagrams to be used in any context, some activities are ill suited to datagram use. For example, maintenance transactions and recursive queries typically require the error control of virtual circuits. Thus datagram use should be restricted to simple queries.

The domain system assumes that a datagram service provides:

1. A non-reliable (i.e. best effort) method of transporting a message of up to 512 octets.

Hence datagram messages are limited to 512 octets. If a datagram message would exceed 512 octets, it is truncated and a truncation flag is set in its header.

2. A message size that gives the number of octets in the datagram.

The main implications for programs accessing name servers via datagrams are:

1. Datagrams should not be used for maintenance transactions and recursive queries.

2. Since datagrams may be lost, the originator of a query must perform error recovery (such as retransmissions) as appropriate.

3. Since network or host delay may cause retransmission when a datagram has not been lost, the originator of a query must be ready to deal with duplicate responses.

The domain system assumes that a virtual circuit service provides:

1. A reliable method of transmitting a message of up to 65535 octets.

2. A message size that gives the number of octets in the message.

   If the virtual circuit service does not provide for message boundary detection or limits transmission size to less than 65535 octets, then messages are prefaced with an unsigned 16 bit length field and broken up into separate transmissions as required. The length field is only prefaced on the first message. This technique is used for TCP virtual circuits.

3. Multiple messages may be sent over a virtual circuit.

4. A method for closing a virtual circuit.

5. A method for detecting that the other party has requested that the virtual circuit be closed.

The main implications for programs accessing name servers via virtual circuits are:

1. Either end of a virtual circuit may initiate a close when there is no activity in progress. The other end should comply.

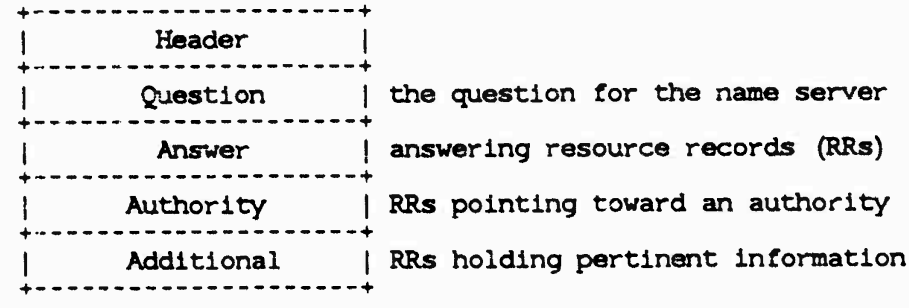
The decision to initiate a close is a matter of individual site policy; some name servers may leave a virtual circuit open for an indeterminate period following a query to allow for subsequent queries; other name servers may choose to initiate a close following the completion of the first query on a virtual circuit. Of course, name servers should not close the virtual circuit in the midst of a multiple message stream used for zone transfer.

2. Since network delay may cause one end to erroneously believe that no activity is in progress, a program which receives a virtual circuit close while a query is in progress should close the virtual circuit and resubmit the query on a new virtual circuit.

All messages may use a compression scheme to reduce the space consumed by repetitive domain names. The use of the compression scheme is optional for the sender of a message, but all receivers must be capable of decoding compressed domain names.

Overall message format

All messages sent by the domain system are divided into 5 sections (some of which are empty in certain cases) shown below:



The header section is always present. The header includes fields that specify which of the remaining sections are present, and also specify whether the message is a query, inverse query, completion query, or response.

The question section contains fields that describe a question to a name server. These fields are a query type (QTYPE), a query class (QCLASS), and a query domain name (QNAME).

The last three sections have the same format: a possibly empty list of concatenated resource records (RRs). The answer section contains RRs that answer the question; the authority section

contains RRs that point toward an authoritative name server; the additional records section contains RRs which relate to the query, but are not strictly answers for the question.

The next two sections of this memo illustrate the use of these message sections through examples; a detailed discussion of data formats follows the examples.

The contents of standard queries and responses

When a name server processes a standard query, it first determines whether it is an authority for the domain name specified in the query.

If the name server is an authority, it returns either:

1. the specified resource information
2. an indication that the specified name does not exist
3. an indication that the requested resource information does not exist

If the name server is not an authority for the specified name, it returns whatever relevant resource information it has along with resource records that the requesting resolver can use to locate an authoritative name server.

Standard query and response example

The overall structure of a query for retrieving information for Internet mail for domain F.ISI.ARPA is shown below:

Header	OPCODE=QUERY, ID=2304
Question	QTYPE=MAILA, QCLASS=IN, QNAME=F.ISI.ARPA
Answer	<empty>
Authority	<empty>
Additional	<empty>

The header includes an opcode field that specifies that this datagram is a query, and an ID field that will be used to associate replies with the original query. (Some additional header fields have been omitted for clarity.) The question section specifies that the type of the query is for mail agent information, that only ARPA Internet information is to be considered, and that the domain name of interest is F.ISI.ARPA. The remaining sections are empty, and would not use any octets in a real query.

One possible response to this query might be:

Header	OPCODE=RESPONSE, ID=2304
Question	QTYPE=MAILA, QCLASS=IN, QNAME=F.ISI.ARPA
Answer	<empty>
Authority	ARPA NS IN A.ISI.ARPA --- ARPA NS IN F.ISI.ARPA
Additional	F.ISI.ARPA A IN 10.2.0.52 --- A.ISI.ARPA A IN 10.1.0.22

This type of response would be returned by a name server that was not an authority for the domain name F.ISI.ARPA. The header field specifies that the datagram is a response to a query with an ID of 2304. The question section is copied from the question section in the query datagram.

The answer section is empty because the name server did not have any information that would answer the query. (Name servers may happen to have cached information even if they are not authoritative for the query.)

The best that this name server could do was to pass back information for the domain ARPA. The authority section specifies two name servers for the domain ARPA using the Internet family: A.ISI.ARPA and F.ISI.ARPA. Note that it is merely a coincidence that F.ISI.ARPA is a name server for ARPA as well as the subject of the query.

In this case, the name server included in the additional records section the Internet addresses for the two hosts specified in the authority section. Such additional data is almost always available.

Given this response, the process that originally sent the query might resend the query to the name server on A.ISI.ARPA, with a new ID of 2305.

The name server on A.ISI.ARPA might return a response:

```
                +-----------------------------------------+
Header          |        OPCODE=RESPONSE, ID=2305         |
                +-----------------------------------------+
Question        |QTYPE=MAILA, QCLASS=IN, QNAME=F.ISI.ARPA |
                +-----------------------------------------+
Answer          |       F.ISI.ARPA MD IN F.ISI.ARPA       |
                |                -------                  |
                |       F.ISI.ARPA MF IN A.ISI.ARPA       |
                +-----------------------------------------+
Authority       |                <empty>                  |
                +-----------------------------------------+
Additional      |       F.ISI.ARPA A IN 10.2.0.52         |
                |                -------                  |
                |       A.ISI.ARPA A IN 10.1.0.22         |
                +-----------------------------------------+
```

This query was directed to an authoritative name server, and hence the response includes an answer but no authority records. In this case, the answer section specifies that mail for F.ISI.ARPA can either be delivered to F.ISI.ARPA or forwarded to A.ISI.ARPA. The additional records section specifies the Internet addresses of these hosts.

The contents of inverse queries and responses

Inverse queries reverse the mappings performed by standard query operations; while a standard query maps a domain name to a resource, an inverse query maps a resource to a domain name. For example, a standard query might bind a domain name to a host address; the corresponding inverse query binds the host address to a domain name.

Inverse query mappings are not guaranteed to be unique or complete because the domain system does not have any internal mechanism for determining authority from resource records that parallels the capability for determining authority as a function of domain name. In general, resolvers will be configured to direct inverse queries to a name server which is known to have the desired information.

Name servers are not required to support any form of inverse queries; it is anticipated that most name servers will support address to domain name conversions, but no other inverse mappings. If a name server receives an inverse query that it does not support, it returns an error response with the "Not Implemented" error set in the header. While inverse query support is optional, all name servers must be at least able to return the error response.

When a name server processes an inverse query, it either returns:

1. zero, one, or multiple domain names for the specified resource

2. an error code indicating that the name server doesn't support inverse mapping of the specified resource type.

Inverse query and response example

The overall structure of an inverse query for retrieving the domain name that corresponds to Internet address 10.2.0.52 is shown below:

Header	OPCODE=IQUERY, ID=997
Question	<empty>
Answer	<anyname> A IN 10.2.0.52
Authority	<empty>
Additional	<empty>

This query asks for a question whose answer is the Internet style address 10.2.0.52. Since the owner name is not known, any domain name can be used as a placeholder (and is ignored). The response to this query might be:

Header	OPCODE=RESPONSE, ID=997
Question	QTYPE=A, QCLASS=IN, QNAME=F.ISI.ARPA
Answer	F.ISI.ARPA A IN 10.2.0.52
Authority	<empty>
Additional	<empty>

Note that the QTYPE in a response to an inverse query is the same as the TYPE field in the answer section of the inverse query. Responses to inverse queries may contain multiple questions when the inverse is not unique.

Completion queries and responses

Completion queries ask a name server to complete a partial domain name and return a set of RRs whose domain names meet a specified set of criteria for "closeness" to the partial input. This type of query can provide a local shorthand for domain names or command completion similar to that in TOPS-20.

Implementation of completion query processing is optional in a name server. However, a name server must return a "Not Implemented" (NI) error response if it does not support completion.

The arguments in a completion query specify:

1. A type in QTYPE that specifies the type of the desired name. The type is used to restrict the type of RRs which will match the partial input so that completion queries can be used for mailbox names, host names, or any other type of RR in the domain system without concern for matches to the wrong type of resource.

2. A class in QCLASS which specifies the desired class of the RR.

3. A partial domain name that gives the input to be completed. All returned RRs will begin with the partial string. The search process first looks for names which qualify under the assumption that the partial string ends with a full label ("whole label match"); if this search fails, the search continues under the assumption that the last label in the partial sting may be an incomplete label ("partial label match"). For example, if the partial string "Smith" was used in a mailbox completion, it would match Smith@ISI.ARPA in preference to Smithsonian@ISI.ARPA.

   The partial name is supplied by the user through the user program that is using domain services. For example, if the user program is a mail handler, the string might be "Mockap" which the user intends as a shorthand for the mailbox Mockapetris@ISI.ARPA; if the user program is TELNET, the user might specify "F" for F.ISI.ARPA.

   In order to make parsing of messages consistent, the partial name is supplied in domain name format (i.e. a sequence of labels terminated with a zero length octet). However, the trailing root label is ignored during matching.

4. A target domain name which specifies the domain which is to be examined for matches. This name is specified in the additional

section using a NULL RR. All returned names will end with the target name.

The user program which constructs the query uses the target name to restrict the search. For example, user programs running at ISI might restrict completion to names that end in ISI.ARPA; user programs running at MIT might restrict completion to the domain MIT.ARPA.

The target domain name is also used by the resolver to determine the name server which should be used to process the query. In general, queries should be directed to a name server that is authoritative for the target domain name. User programs which wish to provide completion for a more than one target can issue multiple completion queries, each directed at a different target. Selection of the target name and the number of searches will depend on the goals of the user program.

5. An opcode for the query. The two types of completion queries are "Completion Query - Multiple", or CQUERYM, which asks for all RRs which could complete the specified input, and "Completion Query - Unique", or CQUERYU, which asks for the "best" completion.

   CQUERYM is used by user programs which want to know if ambiguities exist or wants to do its own determinations as to the best choice of the available candidates.

   CQUERYU is used by user programs which either do not wish to deal with multiple choices or are willing to use the closeness criteria used by CQUERYU to select the best match.

When a name server receives either completion query, it first looks for RRs that begin (on the left) with the same labels as are found in QNAME (with the root deleted), and which match the QTYPE and QCLASS. This search is called "whole label" matching. If one or more hits are found the name server either returns all of the hits (CQUERYM) or uses the closeness criteria described below to eliminate all but one of the matches (CQUERYU).

If the whole label match fails to find any candidates, then the name server assumes that the rightmost label of QNAME (after root deletion) is not a complete label, and looks for candidates that would match if characters were added (on the right) to the rightmost label of QNAME. If one or more hits are found the name server either returns all of the hits (CQUERYM) or uses the closeness criteria described below to eliminate all but one of the matches (CQUERYU).

If a CQUERYU query encounters multiple hits, it uses the following sequence of rules to discard multiple hits:

1. Discard candidates that have more labels than others. Since all candidates start with the partial name and end with the target name, this means that we select those entries that require the fewest number of added labels. For example, a host search with a target of "ISI.ARPA" and a partial name of "A" will select A.ISI.ARPA in preference to A.IBM-PCS.ISI.ARPA.

2. If partial label matching was used, discard those labels which required more characters to be added. For example, a mailbox search for partial "X" and target "ISI.ARPA" would prefer XX@ISI.ARPA to XYZZY@ISI.ARPA.

If multiple hits are still present, return all hits.

Completion query mappings are not guaranteed to be unique or complete because the domain system does not have any internal mechanism for determining authority from a partial domain name that parallels the capability for determining authority as a function of a complete domain name. In general, resolvers will be configured to direct completion queries to a name server which is known to have the desired information.

When a name server processes a completion query, it either returns:

1. An answer giving zero, one, or more possible completions.

2. an error response with Not Implemented (NI) set.

Completion query and response example

Suppose that the completion service was used by a TELNET program to allow a user to specify a partial domain name for the desired host. Thus a user might ask to be connected to "B". Assuming that the query originated from an ISI machine, the query might look like:

Header	OPCODE=CQUERYU, ID=409
Question	QTYPE=A, QCLASS=IN, QNAME=B
Answer	<empty>
Authority	<empty>
Additional	ISI.ARPA NULL IN

The partial name in the query is "B", the mappings of interest are ARPA Internet address records, and the target domain is ISI.ARPA. Note that NULL is a special type of NULL resource record that is used as a placeholder and has no significance; NULL RRs obey the standard format but have no other function.

The response to this completion query might be:

Header	OPCODE=RESPONSE, ID=409
Question	QTYPE=A, QCLASS=IN, QNAME=B
Answer	B.ISI.ARPA A IN 10.3.0.52
Authority	<empty>
Additional	ISI.ARPA NULL IN

This response has completed B to mean B.ISI.ARPA.

Another query might be:

Header	OPCODE=CQUERYM, ID=410
Question	QTYPE=A, QCLASS=IN, QNAME=B
Answer	<empty>
Authority	<empty>
Additional	ARPA NULL IN

This query is similar to the previous one, but specifies a target of ARPA rather than ISI.ARPA. It also allows multiple matches. In this case the same name server might return:

Header	OPCODE=RESPONSE, ID=410
Question	QTYPE=A, QCLASS=IN, QNAME=B
Answer	B.ISI.ARPA A IN 10.3.0.52 - B.BBN.ARPA A IN 10.0.0.49 - B.BBNCC.ARPA A IN 8.1.0.2
Authority	<empty>
Additional	ARPA NULL IN

This response contains three answers, B.ISI.ARPA, B.BBN.ARPA, and B.BBNCC.ARPA.

Recursive Name Service

Recursive service is an optional feature of name servers.

When a name server receives a query regarding a part of the name space which is not in one of the name server's zones, the standard response is a message that refers the requestor to another name server. By iterating on these referrals, the requestor eventually is directed to a name server that has the required information.

Name servers may also implement recursive service. In this type of service, a name server either answers immediately based on local zone information, or pursues the query for the requestor and returns the eventual result back to the original requestor.

A name server that supports recursive service sets the Recursion Available (RA) bit in all responses it generates. A requestor asks for recursive service by setting the Recursion Desired (RD) bit in queries. In some situations where recursive service is the only path to the desired information (see below), the name server may go recursive even if RD is zero.

If a query requests recursion (RD set), but the name server does not support recursion, and the query needs recursive service for an answer, the name server returns a "Not Implemented" (NI) error code. If the query can be answered without recursion since the name server is authoritative for the query, it ignores the RD bit.

Because of the difficulty in selecting appropriate timeouts and error handling, recursive service is best suited to virtual circuits, although it is allowed for datagrams.

Recursive service is valuable in several special situations:

In a system of small personal computers clustered around one or more large hosts supporting name servers, the recursive approach minimizes the amount of code in the resolvers in the personal computers. Such a design moves complexity out of the resolver into the name server, and may be appropriate for such systems.

Name servers on the boundaries of different networks may wish to offer recursive service to create connectivity between different networks. Such name servers may wish to provide recursive service regardless of the setting of RD.

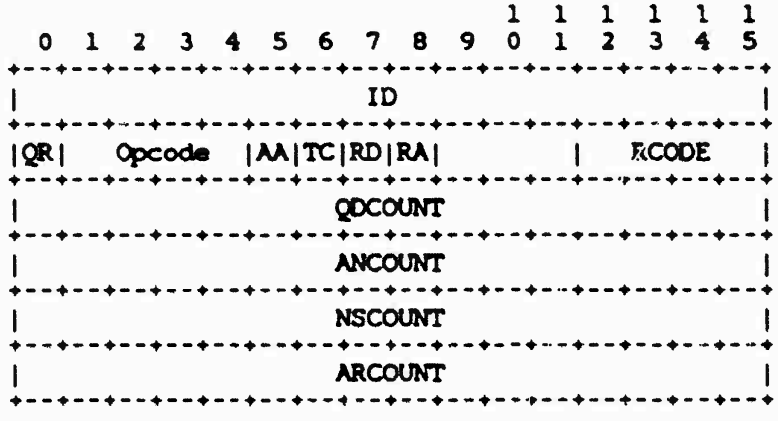
Name servers that translate between domain name service and some other name service may wish to adopt the recursive style. Implicit recursion may be valuable here as well.

These concepts are still under development.

Header section format

```
+---------------------------------------------------+
|                                                   |
|              ***** WARNING *****                  |
|                                                   |
|  The following format is preliminary and is       |
| included for purposes of explanation only. In     |
| particular, the size and position of the          |
| OPCODE, RCODE fields and the number and           |
| meaning of the single bit fields are subject      |
| to change.                                        |
|                                                   |
+---------------------------------------------------+
```

The header contains the following fields:



where:

ID - A 16 bit identifier assigned by the program that generates any kind of query. This identifier is copied into all replies and can be used by the requestor to relate replies to outstanding questions.

QR - A one bit field that specifies whether this message is a query (0), or a response (1).

OPCODE - A four bit field that specifies kind of query in this message. This value is set by the originator of a query and copied into the response. The values are:

0 a standard query (QUERY)

1 an inverse query (IQUERY)

2 an completion query allowing multiple answers (CQUERYM)

2 an completion query requesting a single answer (CQUERYU)

4-15 reserved for future use

AA - Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in the corresponding query.

TC - TrunCation - specifies that this message was truncated due to length greater than 512 characters. This bit is valid in datagram messages but not in messages sent over virtual circuits.

RD - Recursion Desired - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively. Recursive query support is optional.

RA - Recursion Available - this be is set or cleared in a response, and denotes whether recursive query support is available in the name server.

RCODE - Response code - this 4 bit field is set as part of responses. The values have the following interpretation:

0 No error condition

1 Format error - The name server was unable to interpret the query.

2 Server failure - The name server was unable to process this query due to a problem with the name server.

3 Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.

4 Not Implemented - The name server does not support the requested kind of query.

5 Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requestor, or a name server may not wish to perform a particular operation (e.g. zone transfer) for particular data.

6-15 Reserved for future use.

QDCOUNT - an unsigned 16 bit integer specifying the number of entries in the question section.

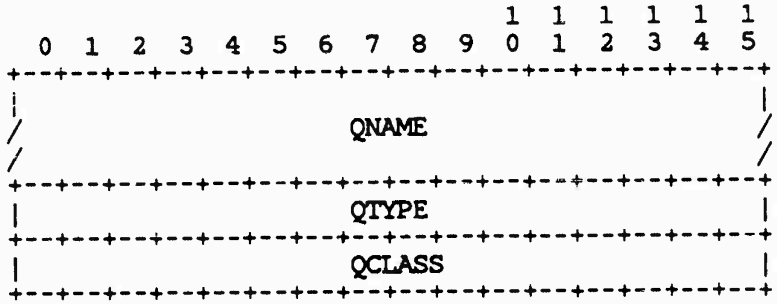
ANCOUNT - an unsigned 16 bit integer specifying the number of resource records in the answer section.

NSCOUNT - an unsigned 16 bit integer specifying the number of name server resource records in the authority records section.

ARCOUNT - an unsigned 16 bit integer specifying the number of resource records in the additional records section.

Question section format

The question section is used in all kinds of queries other than inverse queries. In responses to inverse queries, this section may contain multiple entries; for all other responses it contains a single entry. Each entry has the following format:



where:

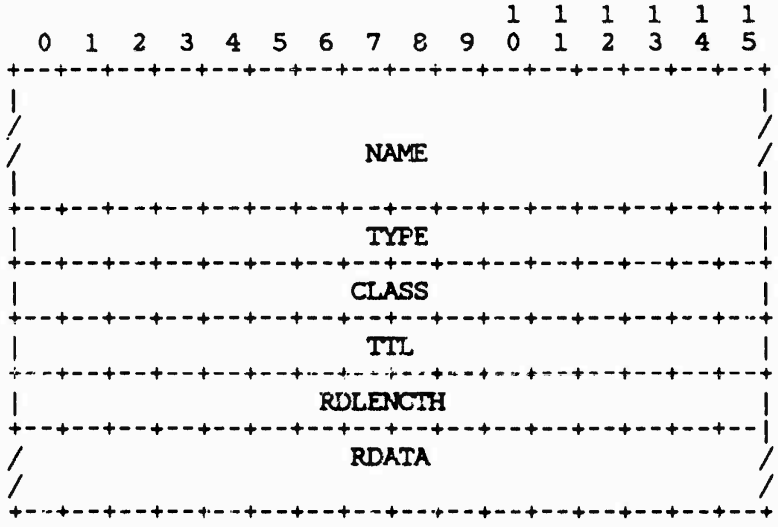
QNAME - a variable number of octets that specify a domain name. This field uses the compressed domain name format described in the next section of this memo. This field can be used to derive a text string for the domain name. Note that this field may be an odd number of octets; no padding is used.

QTYPE - a two octet code which specifies the type of the query. The values for this field include all codes valid for a TYPE field, together with some more general codes which can match more than one type of RR. For example, QTYPE might be A and only match type A RRs, or might be MAILA, which matches MF and MD type RRs. The values for this field are listed in Appendix 2.

QCLASS - a two octet code that specifies the class of the query. For example, the QCLASS field is IN for the ARPA Internet, CS for the CSNET, etc. The numerical values are defined in Appendix 2.

Resource record format

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format:



where:

NAME - a compressed domain name to which this resource record pertains.

TYPE - two octets containing one of the RR type codes defined in Appendix 2. This field specifies the meaning of the data in the RDATA field.

CLASS - two octets which specify the class of the data in the RDATA field.

TTL - a 16 bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. For example, SOA records are always distributed with a zero TTL to prohibit caching. Zero values can also be used for extremely volatile data.

RDLENGTH- an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

RDATA - a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record. For example, the if the TYPE is A and the CLASS is IN, the RDATA field is a 4 octet ARPA Internet address.

Formats for particular resource records are shown in Appendicies 2 and 3.

Domain name representation and compression

Domain names messages are expressed in terms of a sequence of labels. Each label is represented as a one octet length field followed by that number of octets. Since every domain name ends with the null label of the root, a compressed domain name is terminated by a length byte of zero. The high order two bits of the length field must be zero, and the remaining six bits of the length field limit the label to 63 octets or less.

To simplify implementations, the total length of label octets and label length octets that make up a domain name is restricted to 255 octets or less. Since the trailing root label and its dot are not printed, printed domain names are 254 octets or less.

Although labels can contain any 8 bit values in octets that make up a label, it is strongly recommended that labels follow the syntax described in Appendix 1 of this memo, which is compatible with existing host naming conventions. Name servers and resolvers must compare labels in a case-insensitive manner, i.e. A=a, and hence all character strings must be ASCII with zero parity. Non-alphabetic codes must match exactly.

Whenever possible, name servers and resolvers must preserve all 8 bits of domain names they process. When a name server is given data for the same name under two different case usages, this preservation is not always possible. For example, if a name server is given data for ISI.ARPA and isi.arpa, it should create a single node, not two, and hence will preserve a single casing of the label. Systems with case sensitivity should take special precautions to insure that the domain data for the system is created with consistent case.

In order to reduce the amount of space used by repetitive domain names, the sequence of octets that defines a domain name may be terminated by a pointer to the length octet of a previously specified label string. The label string that the pointer

specifies is appended to the already specified label string. Exact duplication of a previous label string can be done with a single pointer. Multiple levels are allowed.

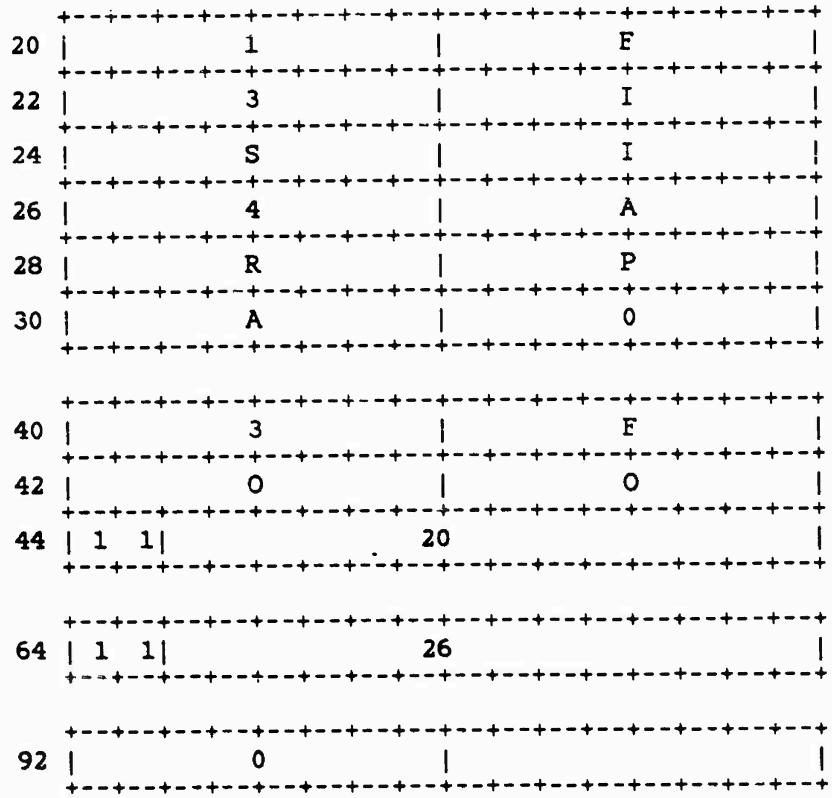
Pointers can only be used in positions in the message where the format is not class specific. If this were not the case, a name server that was handling a RR for another class could make erroneous copies of RRs. As yet, there are no such cases, but they may occur in future RDATA formats.

If a domain name is contained in a part of the message subject to a length field (such as the RDATA section of an RR), and compression is used, the length of the compressed name is used in the length calculation, rather than the length of the expanded name.

Pointers are represented as a two octet field in which the high order 2 bits are ones, and the low order 14 bits specify an offset from the start of the message. The 01 and 10 values of the high order bits are reserved for future use and should not be used.

Programs are free to avoid using pointers in datagrams they generate, although this will reduce datagram capacity. However all programs are required to understand arriving messages that contain pointers.

For example, a datagram might need to use the domain names F.ISI.ARPA, FOO.F.ISI.ARPA, ARPA, and the root. Ignoring the other fields of the message, these domain names might be represented as:



The domain name for F.ISI.ARPA is shown at offset 20. The domain name FOO.F.ISI.ARPA is shown at offset 40; this definition uses a pointer to concatenate a label for FOO to the previously defined F.ISI.ARPA. The domain name ARPA is defined at offset 64 using a pointer to the ARPA component of the name F.ISI.ARPA at 20; note that this reference relies on ARPA being the last label in the string at 20. The root domain name is defined by a single octet of zeros at 92; the root domain name has no labels.

Organization of the Shared database

While name server implementations are free to use any internal data structures they choose, the suggested structure consists of several separate trees. Each tree has structure corresponding to the domain name space, with RRs attached to nodes and leaves. Each zone of authoritative data has a separate tree, and one tree holds all non-authoritative data. All of the trees corresponding to zones are managed identically, but the non-authoritative or cache tree has different management procedures.

Data stored in the database can be kept in whatever form is convenient for the name server, so long as it can be transformed back into the format needed for messages. In particular, the database will probably use structure in place of expanded domain names, and will also convert many of the time intervals used in the domain systems to absolute local times.

Each tree corresponding to a zone has complete information for a "pruned" subtree of the domain space. The top node of a zone has a SOA record that marks the start of the zone. The bottom edge of the zone is delimited by nodes containing NS records signifying delegation of authority to other zones, or by leaves of the domain tree. When a name server contains abutting zones, one tree will have a bottom node containing a NS record, and the other tree will begin with a tree location containing a SOA record.

Note that there is one special case that requires consideration when a name server is implemented. A node that contains a SOA RR denoting a start of zone will also have NS records that identify the name servers that are expected to have a copy of the zone. Thus a name server will usually find itself (and possibly other redundant name servers) referred to in NS records occupying the same position in the tree as SOA records. The solution to this problem is to never interpret a NS record as delimiting a zone started by a SOA at the same point in the tree. (The sample programs in this memo deal with this problem by processing SOA records only after NS records have been processed.)

Zones may also overlap a particular part of the name space when they are of different classes.

Other than the abutting and separate class cases, trees are always expected to be disjoint. Overlapping zones are regarded as a non-fatal error. The scheme described in this memo avoids the overlap issue by maintaining separate trees; other designs must take the appropriate measures to defend against possible overlap.

Non-authoritative data is maintained in a separate tree. This tree is unlike the zone trees in that it may have "holes". Each RR in the cache tree has its own TTL that is separately managed. The data in this tree is never used if authoritative data is available from a zone tree; this avoids potential problems due to cached data that conflicts with authoritative data.

The shared database will also contain data structures to support the processing of inverse queries and completion queries if the local system supports these optional features. Although many schemes are possible, this memo describes a scheme that is based on tables of pointers that invert the database according to key.

Each kind of retrieval has a separate set of tables, with one table per zone. When a zone is updated, these tables must also be updated. The contents of these tables are discussed in the "Inverse query processing" and "Completion query processing" sections of this memo.

The database implementation described here includes two locks that are used to control concurrent access and modification of the database by name server query processing, name server maintenance operations, and resolver access:

The first lock ("main lock") controls access to all of the trees. Multiple concurrent reads are allowed, but write access can only be acquired by a single process. Read and write access are mutually exclusive. Resolvers and name server processes that answer queries acquire this lock in read mode, and unlock upon completion of the current message. This lock is acquired in write mode by a name server maintenance process when it is about to change data in the shared database. The actual update procedures are described under "NAME SERVER MAINTENANCE" but are designed to be brief.

The second lock ("cache queue lock") controls access to the cache queue. This queue is used by a resolver that wishes to add information to the cache tree. The resolver acquires this lock, then places the RRs to be cached into the queue. The name server maintenance procedure periodically acquires this lock and adds the queue information to the cache. The rationale for this procedure is that it allows the resolver to operate with read-only access to the shared database, and allows the update process to batch cache additions and the associated costs for inversion calculations. The name server maintenance procedure must take appropriate precautions to avoid problems with data already in the cache, inversions, etc.

This organization solves several difficulties:

When searching the domain space for the answer to a query, a name server can restrict its search for authoritative data to that tree that matches the most labels on the right side of the domain name of interest.

Since updates to a zone must be atomic with respect to searches, maintenance operations can simply acquire the main lock, insert a new copy of a particular zone without disturbing other zones, and then release the storage used by the old copy. Assuming a central table pointing to valid zone trees, this operation can be a simple pointer swap.

TTL management of zones can be performed using the SOA record for the zone. This avoids potential difficulties if individual RRs in a zone could be timed out separately. This issue is discussed further in the maintenance section.

Query processing

The following algorithm outlines processing that takes place at a name server when a query arrives:

1. Search the list of zones to find zones which have the same class as the QCLASS field in the query and have a top domain name that matches the right end of the QNAME field. If there are none, go to step 2. If there are more than one, pick the zone that has the longest match and go to step 3.

2. Since the zone search failed, the only possible RRs are contained in the non-authoritative tree. Search the cache tree for the NS record that has the same class as the QCLASS field and the largest right end match for domain name. Add the NS record or records to the authority section of the response. If the cache tree has RRs that are pertinent to the question (domain names match, classes agree, not timed-out, and the type field is relevant to the QTYPE), copy these RRs into the answer section of the response. The name server may also search the cache queue. Go to step 4.

3. Since this zone is the best match, the zone in which QNAME resides is either this zone or a zone to which this zone will directly or indirectly delegate authority. Search down the tree looking for a NS RR or the node specified by QNAME.

   If the node exists and has no NS record, copy the relevant RRs to the answer section of the response and go to step 4.

   If a NS RR is found, either matching a part or all of QNAME, then QNAME is in a delegated zone outside of this zone. If so, copy the NS record or records into the authority section of the response, and search the remainder of the zone for an A type record corresponding to the NS reference. If the A record is found, add it to the additional section. Go to step 2.

   If the node is not found and a NS is not found, there is no such name; set the Name error bit in the response and exit.

4. When this step is reached, the answer and authority sections are complete. What remains is to complete the additional section. This procedure is only possible if the name server

knows the data formats implied by the class of records in the answer and authority sections. Hence this procedure is class dependent. Appendix 3 discusses this procedure for Internet class data.

While this algorithm deals with typical queries and databases, several additions are required that will depend on the database supported by the name server:

QCLASS=*

Special procedures are required when the QCLASS of the query is "*". If the database contains several classes of data, the query processing steps above are performed separately for each CLASS, and the results are merged into a single response. The name error condition is not meaningful for a QCLASS=* query. If the requestor wants this information, it must test each class independently.

If the database is limited to data of a particular class, this operation can be performed by simply reseting the authoritative bit in the response, and performing the query as if QCLASS was the class used in the database.

* labels in database RRs

Some zones will contain default RRs that use * to match in cases where the search fails for a particular domain name. If the database contains these records then a failure must be retried using * in place of one or more labels of the search key. The procedure is to replace labels from the left with "*"s looking for a match until either all labels have been replaced, or a match is found. Note that these records can never be the result of caching, so a name server can omit this processing for zones that don't contain RRs with * in labels, or can omit this processing entirely if * never appears in local authoritative data.

Inverse query processing

Name servers that support inverse queries can support these operations through exhaustive searches of their databases, but this becomes impractical as the size of the database increases. An alternative approach is to invert the database according to the search key.

For name servers that support multiple zones and a large amount of data, the recommended approach is separate inversions for each

zone. When a particular zone is changed during a refresh, only its inversions need to be redone.

Support for transfer of this type of inversion may be included in future versions of the domain system, but is not supported in this version.

Completion query processing

Completion query processing shares many of the same problems in data structure design as are found in inverse queries, but is different due to the expected high rate of use of top level labels (ie., ARPA, CSNET). A name server that wishes to be efficient in its use of memory may well choose to invert only occurrences of ARPA, etc. that are below the top level, and use a search for the rare case that top level labels are used to constrain a completion.

NAME SERVER MAINTENANCE

Introduction

Name servers perform maintenance operations on their databases to insure that the data they distribute is accurate and timely. The amount and complexity of the maintenance operations that a name server must perform are related to the size, change rate, and complexity of the database that the name server manages.

Maintenance operations are fundamentally different for authoritative and non-authoritative data. A name server actively attempts to insure the accuracy and timeliness of authoritative data by refreshing the data from master copies. Non-authoritative data is merely purged when its time-to-live expires; the name server does not attempt to refresh it.

Although the refreshing scheme is fairly simple to implement, it is somewhat less powerful than schemes used in other distributed database systems. In particular, an update to the master does not immediately update copies, and should be viewed as gradually percolating though the distributed database. This is adequate for the vast majority of applications. In situations where timliness is critical, the master name server can prohibit caching of copies or assign short timeouts to copies.

Conceptual model of maintenance operations

The vast majority of information in the domain system is derived from master files scattered among hosts that implement name servers; some name servers will have no master files, other name servers will have one or more master files. Each master file contains the master data for a single zone of authority rather than data for the whole domain name space. The administrator of a particular zone controls that zone by updating its master file.

Master files and zone copies from remote servers may include RRs that are outside of the zone of authority when a NS record delegates authority to a domain name that is a descendant of the domain name at which authority is delegated. These forward references are a problem because there is no reasonable method to guarantee that the A type records for the delegatee are available unless they can somehow be attached to the NS records.

For example, suppose the ARPA zone delegates authority at MIT.ARPA, and states that the name server is on AI.MIT.ARPA. If a resolver gets the NS record but not the A type record for AI.MIT.ARPA, it might try to ask the MIT name server for the address of AI.MIT.ARPA.

The solution is to allow type A records that are outside of the zone of authority to be copied with the zone. While these records won't be found in a search for the A type record itself, they can be protected by the zone refreshing system, and will be passed back whenever the name server passes back a referral to the corresponding NS record. If a query is received for the A record, the name server will pass back a referral to the name server with the A record in the additional section, rather than answer section.

The only exception to the use of master files is a small amount of data stored in boot files. Boot file data is used by name servers to provide enough resource records to allow zones to be imported from foreign servers (e.g. the address of the server), and to establish the name and address of root servers. Boot file records establish the initial contents of the cache tree, and hence can be overridden by later loads of authoritative data.

The data in a master file first becomes available to users of the domain name system when it is loaded by the corresponding name server. By definition, data from a master file is authoritative.

Other name servers which wish to be authoritative for a particular zone do so by transferring a copy of the zone from the name server which holds the master copy using a virtual circuit. These copies include parameters which specify the conditions under which the data in the copy is authoritative. In the most common case, the conditions specify a refresh interval and policies to be followed when the refresh operation cannot be performed.

A name server may acquire multiple zones from different name servers and master files, but the name server must maintain each zone separately from others and from non-authoritative data.

When the refresh interval for a particular zone copy expires, the name server holding the copy must consult the name server that holds the master copy. If the data in the zone has not changed, the master name server instructs the copy name server to reset the refresh interval. If the data has changed, the master passes a new copy of the zone and its associated conditions to the copy name server. Following either of these transactions, the copy name server begins a new refresh interval.

Copy name servers must also deal with error conditions under which they are unable to communicate with the name server that holds the master copy of a particular zone. The policies that a copy name server uses are determined by other parameters in the conditions distributed with every copy. The conditions include a retry interval and a maximum holding time. When a copy name server is

unable to establish communications with a master or is unable to complete the refresh transaction, it must retry the refresh operation at the rate specified by the retry interval. This retry interval will usually be substantially shorter than the refresh interval. Retries continue until the maximum holding time is reached. At that time the copy name server must assume that its copy of the data for the zone in question is no longer authoritative.

Queries must be processed while maintenance operations are in progress because a zone transfer can take a long time. However, to avoid problems caused by access to partial databases, the maintenance operations create new copies of data rather than directly modifying the old copies. When the new copy is complete, the maintenance process locks out queries for a short time using the main lock, and switches pointers to replace the old data with the new. After the pointers are swapped, the maintenance process unlocks the main lock and reclaims the storage used by the old copy.

Name server data structures and top level logic

The name server must multiplex its attention between multiple activities. For example, a name server should be able to answer queries while it is also performing refresh activities for a particular zone. While it is possible to design a name server that devotes a separate process to each query and refresh activity in progress, the model described in this memo is based on the assumption that there is a single process performing all maintenance operations, and one or more processes devoted to handling queries. The model also assumes the existence of shared memory for several control structures, the domain database, locks, etc.

The model name server uses the following files and shared data structures:

1. A configuration file that describes the master and boot files which the name server should load and the zones that the name server should attempt to load from foreign name servers. This file establishes the initial contents of the status table.

2. Domain data files that contain master and boot data to be loaded.

3. A status table that is derived from the configuration file. Each entry in this table describes a source of data. Each entry has a zone number. The zone number is zero for

non-authoritative sources; authoritative sources are assigned separate non-zero numbers.

4. The shared database that holds the domain data. This database is assumed to be organized in some sort of tree structure paralleling the domain name space, with a list of resource records attached to each node and leaf in the tree. The elements of the resource record list need not contain the exact data present in the corresponding output format, but must contain data sufficient to create the output format; for example, these records need not contain the domain name that is associated with the resource because that name can be derived from the tree structure. Each resource record also internal data that the name server uses to organize its data.

5. Inversion data structures that allow the name server to process inverse queries and completion queries. Although many structures could be used, the implementation described in this memo supposes that there is one array for every inversion that the name server can handle. Each array contains a list of pointers to resource records such that the order of the inverted quantities is sorted.

6. The main and cache queue locks

7. The cache queue

The maintenance process begins by loading the status table from the configuration file. It then periodically checks each entry, to see if its refresh interval has elapsed. If not, it goes on to the next entry. If so, it performs different operations depending on the entry:

If the entry is for zone 0, or the cache tree, the maintenance process checks to see if additions or deletions are required. Additions are acquired from the cache queue using the cache queue lock. Deletions are detected using TTL checks. If any changes are required, the maintenance process recalculates inversion data structures and then alters the cache tree under the protection of the main lock. Whenever the maintenance process modifies the cache tree, it resets the refresh interval to the minimum of the contained TTLs and the desired time interval for cache additions.

If the entry is not zone 0, and the entry refers to a local file, the maintenance process checks to see if the file has been modified since its last load. If so the file is reloaded using the procedures specified under "Name server file

loading". The refresh interval is reset to that specified in the SOA record if the file is a master file.

If the entry is for a remote master file, the maintenance process checks for a new version using the procedure described in "Names server remote zone transfer".

Name server file loading

Master files are kept in text form for ease of editing by system maintainers. These files are not exchanged by name servers; name servers use the standard message format when transferring zones.

Organizations that want to have a domain, but do not want to run a name server, can use these files to supply a domain definition to another organization that will run a name server for them. For example, if organization X wants a domain but not a name server, it can find another organization, Y, that has a name server and is willing to provide service for X. Organization X defines domain X via the master file format and ships a copy of the master file to organization Y via mail, FTP, or some other method. A system administrator at Y configures Y's name server to read in X's file and hence support the X domain. X can maintain the master file using a text editor and send new versions to Y for installation.

These files have a simple line-oriented format, with one RR per line. Fields are separated by any combination of blanks and tab characters. Tabs are treated the same as spaces; in the following discussion the term "blank" means either a tab or a blank. A line can be either blank (and ignored), a RR, or a $INCLUDE line.

If a RR line starts with a domain name, that domain name is used to specify the location in the domain space for the record, i.e. the owner. If a RR line starts with a blank, it is loaded into the location specified by the most recent location specifier.

The location specifiers are assumed to be relative to some origin that is provided by the user of a file unless the location specifier contains the root label. This provides a convenient shorthand notation, and can also be used to prevent errors in master files from propagating into other zones. This feature is particularly useful for master files imported from other sites.

An include line begins with $INCLUDE, starting at the first line position, and is followed by a local file name and an optional offset modifier. The filename follows the appropriate local conventions. The offset is one or more labels that are added to the offset in use for the file that contained the $INCLUDE. If the offset is omitted, the included file is loaded using the

offset of the file that contained the $INCLUDE command. For example, a file being loaded at offset ARPA might contain the following lines:

```
$INCLUDE <subsys>isi.data ISI
$INCLUDE <subsys>addresses.data
```

The first line would be interpreted to direct loading of the file <subsys>isi.data at offset ISI.ARPA. The second line would be interpreted as a request to load data at offset ARPA.

Note that $INCLUDE commands do not cause data to be loaded into a different zone or tree; they are simply ways to allow data for a given zone to be organized in separate files. For example, mailbox data might be kept separately from host data using this mechanism.

Resource records are entered as a sequence of fields corresponding to the owner name, TTL, CLASS, TYPE and RDATA components. (Note that this order is different from the order used in examples and the order used in the actual RRs; the given order allows easier parsing and defaulting.)

The owner name is derived from the location specifier.

The TTL field is optional, and is expressed as a decimal number. If omitted TTL defaults to zero.

The CLASS field is also optional; if omitted the CLASS defaults to the most recent value of the CLASS field in a previous RR.

The RDATA fields depend on the CLASS and TYPE of the RR. In general, the fields that make up RDATA are expressed as decimal numbers or as domain names. Some exceptions exist, and are documented in the RDATA definitions in Appendicies 2 and 3 of this memo.

Because CLASS and TYPE fields don't contain any common identifiers, and because CLASS and TYPE fields are never decimal numbers, the parse is always unique.

Because these files are text files several special encodings are necessary to allow arbitrary data to be loaded. In particular:

. A free standing dot is used to refer to the current domain name.

@ A free standing @ is used to denote the current origin.

.. Two free standing dots represent the null domain name of the root.

\X where X is any character other than a digit (0-9), is used to quote that character so that its special meaning does not apply. For example, "\." can be used to place a dot character in a label.

\DDD where each D is a digit is the octet corresponding to the decimal number described by DDD. The resulting octet is assumed to be text and is not checked for special meaning.

( ) Parentheses are used to group data that crosses a line boundary. In effect, line terminations are not recognized within parentheses.

; Semicolon is used to start a comment; the remainder of the line is ignored.

Name server file loading example

A name server for F.ISI.ARPA , serving as an authority for the ARPA and ISI.ARPA domains, might use a boot file and two master files. The boot file initializes some non-authoritative data, and would be loaded without an origin:

```
..              9999999 IN      NS      B.ISI.ARPA
                9999999 CS      NS      UDEL.CSNET
B.ISI.ARPA      9999999 IN      A       10.3.0.52
UDEL.CSNET      9999999 CS      A       302-555-0000
```

This file loads non-authoritative data which provides the identities and addresses of root name servers. The first line contains a NS RR which is loaded at the root; the second line starts with a blank, and is loaded at the most recent location specifier, in this case the root; the third and fourth lines load RRs at B.ISI.ARPA and UDEL.CSNET, respectively. The timeouts are set to high values (9999999) to prevent this data from being discarded due to timeout.

The first master file loads authoritative data for the ARPA domain. This file is designed to be loaded with an origin of ARPA, which allows the location specifiers to omit the trailing .ARPA labels.

```
@    IN  SOA     F.ISI.ARPA       Action.E.ISI.ARPA (
                                      20     ; SERIAL
                                      3600   ; REFRESH
                                      600    ; RETRY
                                      3600000; EXPIRE
                                      60)    ; MINIMUM
         NS      F.ISI.ARPA ; F.ISI.ARPA is a name server for ARPA
         NS      A.ISI.ARPA ; A.ISI.ARPA is a name server for ARPA
MIT      NS      AI.MIT.ARPA; delegation to MIT name server
ISI      NS      F.ISI.ARPA ; delegation to ISI name server

UDEL     MD      UDEL.ARPA
         A       10.0.0.96
NBS      MD      NBS.ARPA
         A       10.0.0.19
DTI      MD      DTI.ARPA
         A       10.0.0.12

AI.MIT   A       10.2.0.6
F.ISI    A       10.2.0.52
```

The first group of lines contains the SOA record and its parameters, and identifies name servers for this zone and for delegated zones. The Action.E.ISI.ARPA field is a mailbox specification for the responsible person for the zone, and is the domain name encoding of the mail destination Action@E.ISI.ARPA. The second group specifies data for domain names within this zone. The last group has forward references for name server address resolution for AI.MIT.ARPA and F.ISI.ARPA. This data is not technically within the zone, and will only be used for additional record resolution for NS records used in referrals. However, this data is protected by the zone timeouts in the SOA, so it will persist as long as the NS references persist.

The second master file defines the ISI.ARPA environment, and is loaded with an origin of ISI.ARPA:

```
@    IN  SOA     F.ISI.ARPA       Action\.ISI.E.ISI.ARPA (
                                      20     ; SERIAL
                                      7200   ; REFRESH
                                      600    ; RETRY
                                      3600000; EXPIRE
                                      60)    ; MINIMUM
         NS      F.ISI.ARPA ; F.ISI.ARPA is a name server
A        A       10.1.0.32
         MD      A.ISI.ARPA
         MF      F.ISI.ARPA
B        A       10.3.0.52
         MD      B.ISI.ARPA
```

field in the new SOA record with the SERIAL field in the SOA record of the existing zone copy. If these values match, the zone has not been updated since the last copy and hence there is no reason to recopy the zone. In this case the name server resets the times in the existing SOA record and closes the virtual circuit to complete the operation.

If this is initial load, or the SERIAL fields were different, the name server requests a copy of the zone by sending the foreign name server an AXFR query which specifies the zone by its QCLASS and QNAME fields.

When the foreign name server receives the AXFR request, it sends each node from the zone to the requestor in a separate message. It begins with the node that contains the SOA record, walks the tree in breadth-first order, and completes the transfer by resending the node containing the SOA record.

Several error conditions are possible:

If the AXFR request cannot be matched to a SOA, the foreign name server will return a single message in response that does not contain the AXFR request. (The normal SOA query preceding the AXFR is designed to avoid this condition, but it is still possible.)

The foreign name server can detect an internal error or detect some other condition (e.g. system going down, out of resources, etc.) that forces the transfer to be aborted. If so, it sends a message with the "Server failure" condition set. If the AXFR can be immediately retried with some chance of success, it leaves the virtual open; otherwise it initiates a close.

If the foreign name server doesn't wish to perform the operation for policy reasons (i.e. the system administrator wishes to forbid zone copies), the foreign server returns a "Refused" condition.

The requestor receives these records and builds a new tree. This tree is not yet in the status table, so its data are not used to process queries. The old copy of the zone, if any, may be used to satisfy request while the transfer is in progress.

When the requestor receives the second copy of the SOA node, it compares the SERIAL field in the first copy of the SOA against the SERIAL field in the last copy of the SOA record. If these don't match, the foreign server updated its zone while the transfer was in progress. In this case the requestor repeats the AXFR request to acquire the newer version.

```
        MF      F.ISI.ARPA
F       A       10.2.0.52
        MD      F.ISI.ARPA
        MF      A.ISI.ARPA
$INCLUDE <SUBSYS>ISI-MAILBOXES.TXT
```

Where the file <SUBSYS>ISI-MAILBOXES.TXT is:

```
MOE     MB      F.ISI.ARPA
LARRY   MB      A.ISI.ARPA
CURLEY  MB      B.ISI.ARPA
STOOGES MB      B.ISI.ARPA
        MG      MOE.ISI.ARPA
        MG      LARRY.ISI.ARPA
        MG      CURLEY.ISI.ARPA
```

Note the use of the \ character in the SOA RR to specify the responsible person mailbox "Action.ISI@E.ISI.ARPA".

Name server remote zone transfer

When a name server needs to make an initial copy of a zone or test to see if a existing zone copy should be refreshed, it begins by attempting to open a virtual circuit to the foreign name server.

If this open attempt fails, and this was an initial load attempt, it schedules a retry and exits. If this was a refresh operation, the name server tests the status table to see if the maximum holding time derived from the SOA EXPIRE field has elapsed. If not, the name server schedules a retry. If the maximum holding time has expired, the name server invalidates the zone in the status table, and scans all resource records tagged with this zone number. For each record it decrements TTL fields by the length of time since the data was last refreshed. If the new TTL value is negative, the record is deleted. If the TTL value is still positive, it moves the RR to the cache tree and schedules a retry.

If the open attempt succeeds, the name server sends a query to the foreign name server in which QTYPE=SOA, QCLASS is set according to the status table information from the configuration file, and QNAME is set to the domain name of the zone of interest.

The foreign name server will return either a SOA record indicating that it has the zone or an error. If an error is detected, the virtual circuit is closed, and the failure is treated in the same way as if the open attempt failed.

If the SOA record is returned and this was a refresh, rather than an initial load of the zone, the name server compares the SERIAL

If the AXFR transfer eventually succeeds, the name server closes the virtual circuit and and creates new versions of inversion data structures for this zone. When this operation is complete, the name server acquires the main lock in write mode and then replaces any old copy of the zone and inversion data structures with new ones. The name server then releases the main lock, and can reclaim the storage used by the old copy.

If an error occurs during the AXFR transfer, the name server can copy any partial information into its cache tree if it wishes, although it will not normally do so if the zone transfer was a refresh rather than an initial load.

RESOLVER ALGORITHMS

Operations

Resolvers have a great deal of latitude in the semantics they allow in user calls. For example, a resolver might support different user calls that specify whether the returned information must be from and authoritative name server or not. Resolvers are also responsible for enforcement of any local restrictions on access, etc.

In any case, the resolver will transform the user query into a number of shared database accesses and queries to remote name servers. When a user requests a resource associated with a particular domain name, the resolver will execute the following steps:

1. The resolver first checks the local shared database, if any, for the desired information. If found, it checks the applicable timeout. If the timeout check succeeds, the information is used to satisfy the user request. If not, the resolver goes to step 2.

2. In this step, the resolver consults the shared database for the name server that most closely matches the domain name in the user query. Multiple redundant name servers may be found. The resolver goes to step 3.

3. In this step the resolver chooses one of the available name servers and sends off a query. If the query fails, it tries another name server. If all fail, an error indication is returned to the user. If a reply is received the resolver adds the returned RRs to its database and goes to step 4.

4 In this step, the resolver interprets the reply. If the reply contains the desired information, the resolver returns the information to the user. The the reply indicates that the domain name in the user query doesn't exist, then the resolver returns an error to the user. If the reply contains a transient name server failure, the resolver can either wait and retry the query or go back to step 3 and try a different name server. If the reply doesn't contain the desired information, but does contain a pointer to a closer name server, the resolver returns to step 2, where the closer name servers will be queried.

Several modifications to this algorithm are possible. A resolver may not support a local cache and instead only cache information during the course of a single user request, discarding it upon

completion. The resolver may also find that a datagram reply was truncated, and open a virtual circuit so that the complete reply can be recovered.

Inverse and completion queries must be treated in an environment-sensitive manner, because the domain system doesn't provide a method for guaranteeing that it can locate the correct information. The typical choice will be to configure a resolver to use a particular set of known name servers for inverse queries.

DOMAIN SUPPORT FOR MAIL

Introduction

Mail service is a particularly sensitive issue for users of the domain system because of the lack of a consistent system for naming mailboxes and even hosts, and the need to support continued operation of existing services. This section discusses an evolutionary approach for adding consistent domain name support for mail.

The crucial issue is deciding on the types of binding to be supported. Most mail systems specify a mail destination with a two part construct such as X@Y. The left hand side, X, is an string, often a user or account, and Y is a string, often a host. This section refers to the part on the left, i.e. X, as the local part, and refers to the part on the right, i.e. Y, as the global part.

Most existing mail systems route mail based on the global part; a mailer with mail to deliver to X@Y will decide on the host to be contacted using only Y. We refer to this type of binding as "agent binding".

For example, mail addressed to Mockapetris@ISIF is delivered to host USC-ISIF (USC-ISIF is the official name for the host specified by nickname ISIF).

More sophisticated mail systems use both the local and global parts, i.e. both X and Y to determine which host should receive the mail. These more sophisticated systems usually separate the binding of the destination to the host from the actual delivery. This allows the global part to be a generic name rather than constraining it to a single host. We refer to this type of binding as "mailbox binding".

For example, mail addressed to Mockapetris@ISI might be bound to host F.ISI.ARPA, and subsequently delivered to that host, while mail for Cohen@ISI might be bound to host B.ISI.ARPA.

The domain support for mail consists of two levels of support, corresponding to these two binding models.

The first level, agent binding, is compatible with existing ARPA Internet mail procedures and uses maps a global part onto one or more hosts that will accept the mail. This type of binding uses the MAILA QTYPE.

The second level, mailbox binding, offers extended services

that map a local part and a global part onto one or more sets of data via the MAILB QTYPE. The sets of data include hosts that will accept the mail, mailing list members (mail groups), and mailboxes for reporting errors or requests to change a mail group.

The domain system encodes the global part of a mail destination as a domain name and uses dots in the global part to separate labels in the encoded domain name. The domain system encodes the local part of a mail destination as a single label, and any dots in this part are simply copied into the label. The domain system forms a complete mail destination as the local label concatenated to the domain string for the global part. We call this a mailbox.

For example, the mailbox Mockapetris@F.ISI.ARPA has a global domain name of three labels, F.ISI.ARPA. The domain name encoding for the whole mailbox is Mockapetris.F.ISI.ARPA. The mailbox Mockapetris.cad@F.ISI.ARPA has the same domain name for the global part and a 4 label domain name for the mailbox of Mockapetris\.cad.F.ISI.ARPA (the \ is not stored in the label, its merely used to denote the "quoted" dot).

It is anticipated that the Internet system will adopt agent binding as part of the initial implementation of the domain system, and that mailbox binding will eventually become the preferred style as organizations convert their mail systems to the new style. To facilitate this approach, the domain information for these two binding styles is organized to allow a requestor to determine which types of support are available, and the information is kept in two disjoint classes.

Agent binding

In agent binding, a mail system uses the global part of the mail destination as a domain name, with dots denoting structure. The domain name is resolved using a MAILA query which return MF and MD RRs to specify the domain name of the appropriate host to receive the mail. MD (Mail delivery) RRs specify hosts that are expected to have the mailbox in question; MF (Mail forwarding) RRs specify hosts that are expected to be intermediaries willing to accept the mail for eventual forwarding. The hosts are hints, rather than definite answers, since the query is made without the full mail destination specification.

For example, mail for MOCKAPETRIS@F.ISI.ARPA would result in a query with QTYPE=MAILA and QNAME=F.ISI.ARPA, which might return two RRs:

F.ISI.ARPA MD IN F.ISI.ARPA
F.ISI.ARPA MF IN A.ISI.ARPA

The mailer would interpret these to mean that the mail agent on F.ISI.ARPA should be able to deliver the mail directly, but that A.ISI.ARPA is willing to accept the mail for probable forwarding.

Using this system, an organization could implement a system that uses organization names for global parts, rather than the usual host names, but all mail for the organization would be routed the same, regardless of its local part. Hence and organization with many hosts would expect to see many forwarding operations.

Mailbox binding

In mailbox binding, the mailer uses the entire mail destination specification to construct a domain name. The encoded domain name for the mailbox is used as the QNAME field in a QTYPE=MAILB query.

Several outcomes are possible for this query:

1. The query can return a name error indicating that the mailbox does not exist as a domain name.

   In the long term this would indicate that the specified mailbox doesn't exist. However, until the use of mailbox binding is universal, this error condition should be interpreted to mean that the organization identified by the global part does not support mailbox binding. The appropriate procedure is to revert to agent binding at this point.

2. The query can return a Mail Rename (MR) RR.

   The MR RR carries new mailbox specification in its RDATA field. The mailer should replace the old mailbox with the new one and retry the operation.

3. The query can return a MB RR.

   The MB RR carries a domain name for a host in its RDATA field. The mailer should deliver the message to that host via whatever protocol is applicable, e.g. SMTP.

4. The query can return one or more Mail Group (MG) RRs.

   This condition means that the mailbox was actually a mailing list or mail group, rather than a single mailbox. Each MG RR has a RDATA field that identifies a mailbox that is a member of

the group. The mailer should deliver a copy of the message to each member.

5. The query can return a MB RR as well as one or more MG RRs.

   This condition means the the mailbox was actually a mailing list. The mailer can either deliver the message to the host specified by the MB RR, which will in turn do the delivery to all members, or the mailer can use the MG RRs to do the expansion itself.

In any of these cases, the response may include a Mail Information (MINFO) RR. This RR is usually associated with a mail group, but is legal with a MB. The MINFO RR identifies two mailboxes. One of these identifies a responsible person for the original mailbox name. This mailbox should be used for requests to be added to a mail group, etc. The second mailbox name in the MINFO RR identifies a mailbox that should receive error messages for mail failures. This is particularly appropriate for mailing lists when errors in member names should be reported to a person other than the one who sends a message to the list. New fields may be added to this RR in the future.

Appendix 1 - Domain Name Syntax Specification

The preferred syntax of domain names is given by the following BNF rules. Adherence to this syntax will result in fewer problems with many applications that use domain names (e.g., mail, TELNET). Note that some applications use domain names containing binary information and hence do not follow this syntax.

<domain> ::= <subdomain> | " "

<subdomain> ::= <label> | <subdomain> "." <label>

<label> ::= <letter> [[<ldh-str>] <let-dig>]

<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>

<let-dig-hyp> ::= <let-dig> | "-"

<let-dig> ::= <letter> | <digit>

<letter> ::= any one of the 52 alphabetic characters A through Z in upper case and a through z in lower case

<digit> ::= any one of the ten digits 0 through 9

Note that while upper and lower case letters are allowed in domain names no significance is attached to the case. That is, two names with the same spelling but different case are to be treated as if identical.

The labels must follow the rules for ARPANET host names. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphen. There are also some restrictions on the length. Labels must be 63 characters or less.

For example, the following strings identify hosts in the ARPA Internet:

F.ISI.ARPA LINKABIT-DCN5.ARPA UCL-TAC.ARPA

Appendix 2 - Field formats and encodings

```
+-------------------------------------------------+
|                                                 |
|              *****  WARNING  *****              |
|                                                 |
|  The following formats are preliminary and      |
| are included for purposes of explanation only.  |
| In particular, new RR types will be added,      |
| and the size, position, and encoding of         |
| fields are subject to change.                   |
|                                                 |
+-------------------------------------------------+
```

TYPE values

TYPE fields are used in resource records. Note that these types are not the same as the QTYPE fields used in queries, although the functions are often similar.

TYPE value		meaning
A	1	a host address
NS	2	an authoritative name server
MD	3	a mail destination
MF	4	a mail forwarder
CNAME	5	the canonical name for an alias
SOA	6	marks the start of a zone of authority
MB	7	a mailbox domain name
MG	8	a mail group member
MR	9	a mail rename domain name
NULL	10	a null RR
WKS	11	a well known service description
PTR	12	a domain name pointer
HINFO	13	host information
MINFO	14	mailbox or mail list information

QTYPE values

QTYPE fields appear in the question part of a query. They include the values of TYPE with the following additions:

AXFR	252	A request for a transfer of an entire zone of authority
MAILB	253	A request for mailbox-related records (MB, MG or MR)
MAILA	254	A request for mail agent RRs (MD and MF)
*	255	A request for all records

CLASS values

CLASS fields appear in resource records

CLASS value meaning

IN	1	the ARPA Internet
CS	2	the computer science network (CSNET)

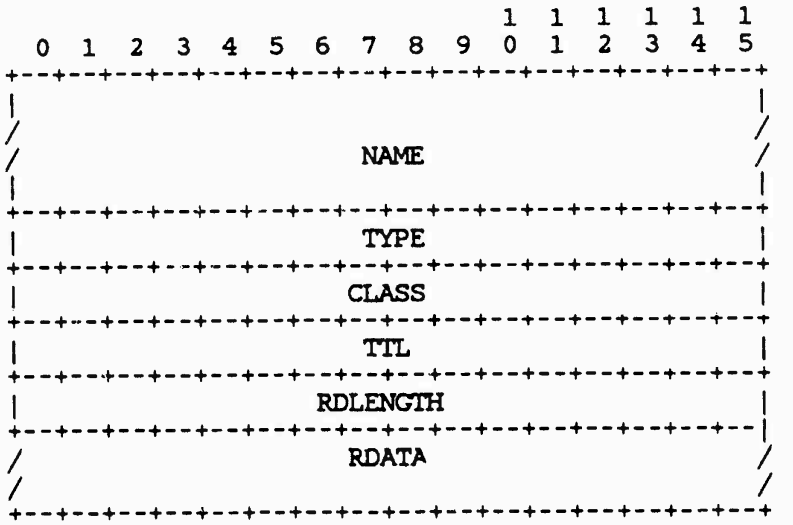
QCLASS values

QCLASS fields appear in the question section of a query. They include the values of CLASS with the following additions:

*	255 any class

Standard resource record formats

All RRs have the same top level format shown below:



where:

NAME - a compressed domain name to which this resource record pertains.

TYPE - two octets containing one of the RR type codes defined in Appendix 2. This field specifies the meaning of the data in the RDATA field.

CLASS - two octets which specifies the class of the data in the RDATA field.

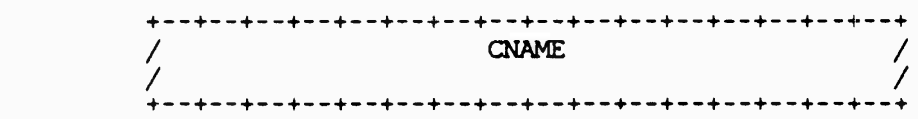
TTL - a 16 bit signed integer that specifies the time interval that the resource record may be cached before the source of the information should again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. For example, SOA records are always distributed with a zero TTL to prohibit caching. Zero values can also be used for extremely volatile data.

RDLENGTH- an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

RDATA - a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record.

The format of the RDATA field is standard for all classes for the RR types NS, MD, MF, CNAME, SOA, MB, MG, MR, PTR, HINFO, MINFO and NULL. These formats are shown below together with the appropriate additional section RR processing.

CNAME RDATA format

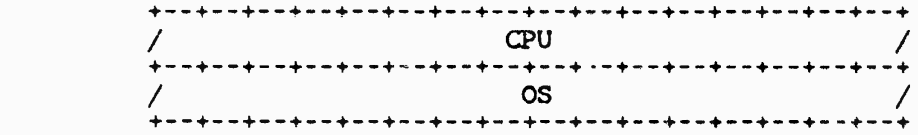


where:

CNAME - A compressed domain name which specifies that the domain name of the RR is an alias for a canonical name specified by CNAME.

CNAME records cause no additional section processing. The RDATA section of a CNAME line in a master file is a standard printed domain name.

HINFO RDATA format



where:

CPU - A character string which specifies the CPU type. The character string is represented as a single octet length followed by that number of characters. The following standard strings are defined:.

PDP-11/70	C/30	C/70	VAX-11/780
H-316	H-516	DEC-2060	DEC-1090T
ALTO	IBM-PC	IBM-PC/XT	PERQ
IBM-360/67	IBM-370/145		

OS - A character string which specifies the operating system type. The character string is represented as a single octet

length followed by that number of characters. The following standard types are defined:.

ASP	AUGUST	BKY	CCP
DOS/360	ELF	EPOS	EXEC-8
GCOS	GPOS	ITS	INTERCOM
KRONOS	MCP	MOS	MPX-RT
MULTICS	MVT	NOS	NOS/BE
OS/MVS	OS/MVT	RIG	RSX11
RSX11M	RT11	SCOPE	SIGNAL
SINTRAN	TENEX	TOPS10	TOPS20
TSS	UNIX	VM/370	VM/CMS
VMS	WAITS		

HINFO records cause no additional section processing.

HINFO records are used to acquire general information about a host. The main use is for protocols such as FTP that can use special procedures when talking between machines or operating systems of the same type.

MB RDATA format

where:

MADNAME - A compressed domain name which specifies a host which has the specified mailbox.

MB records cause additional section processing which looks up an A type record corresponding to MADNAME. The RDATA section of a MB line in a master file is a standard printed domain name.

MD RDATA format

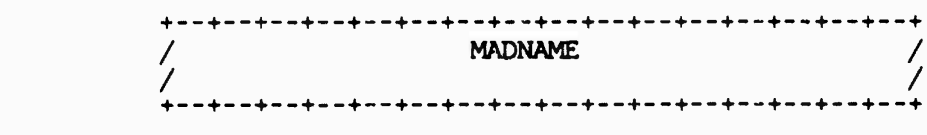
where:

MADNAME - A compressed domain name which specifies a host which

has a mail agent for the domain which should be able to deliver mail for the domain.

MD records cause additional section processing which looks up an A type record corresponding to MADNAME. The RDATA section of a MD line in a master file is a standard printed domain name.

MF RDATA format

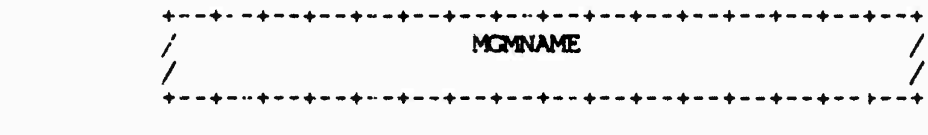


where:

MADNAME - A compressed domain name which specifies a host which has a mail agent for the domain which will accept mail for forwarding to the domain.

MF records cause additional section processing which looks up an A type record corresponding to MADNAME. The RDATA section of a MF line in a master file is a standard printed domain name.

MG RDATA format

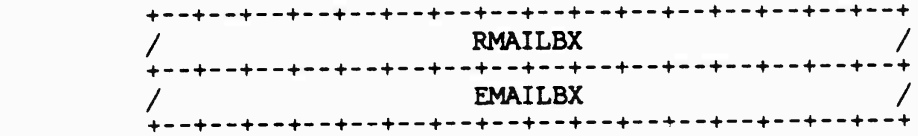


where:

MGMNAME - A compressed domain name which specifies a mailbox which is a member of the mail group specified by the domain name.

MF records cause no additional section processing. The RDATA section of a MF line in a master file is a standard printed domain name.

MINFO RDATA format



where:

RMAILBX - A compressed domain name which specifies a mailbox which is responsible for the mailing list or mailbox. If this domain name names the root, the owner of the MINFO RR is responsible for itself. Note that many existing mailing lists use a mailbox X-request for the RMAILBX field of mailing list X, e.g. Msgroup-request for Msgroup. This field provides a more general mechanism.

EMAILBX - A compressed domain name which specifies a mailbox which is to receive error messages related to the mailing list or mailbox specified by the owner of the MINFO RR (similar to the ERRORS-TO: field which has been proposed). If this domain name names the root, errors should be returned to the sender of the message.

MINFO records cause no additional section processing. Although these records can be associated with a simple mailbox, they are usually used with a mailing list. The MINFO section of a MF line in a master file is a standard printed domain name.

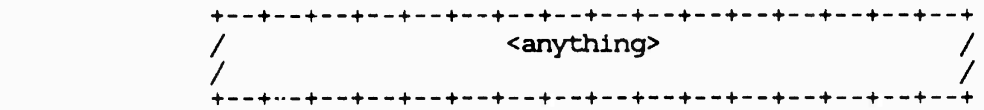
MR RDATA format

where:

NEWNAME - A compressed domain name which specifies a mailbox which is the proper rename of the specified mailbox.

MR records cause no additional section processing. The RDATA section of a MR line in a master file is a standard printed domain name.

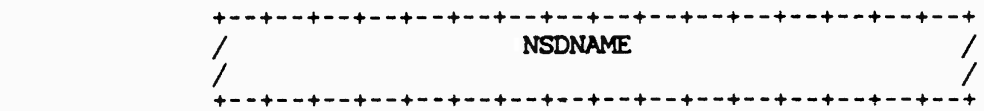
NULL RDATA format



Anything at all may be in the RDATA field so long as it is 65535 octets or less.

NULL records cause no additional section processing. NULL RRs are not allowed in master files.

NS RDATA format

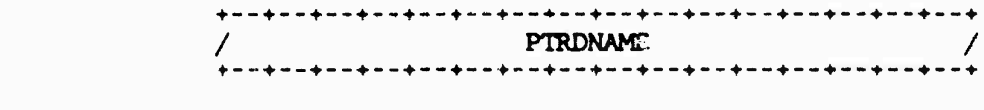


where:

NSDNAME - A compressed domain name which specifies a host which has a name server for the domain.

NS records cause both the usual additional section processing to locate a type A record, and a special search of the zone in which they reside. The RDATA section of a NS line in a master file is a standard printed domain name.

PTR RDATA format

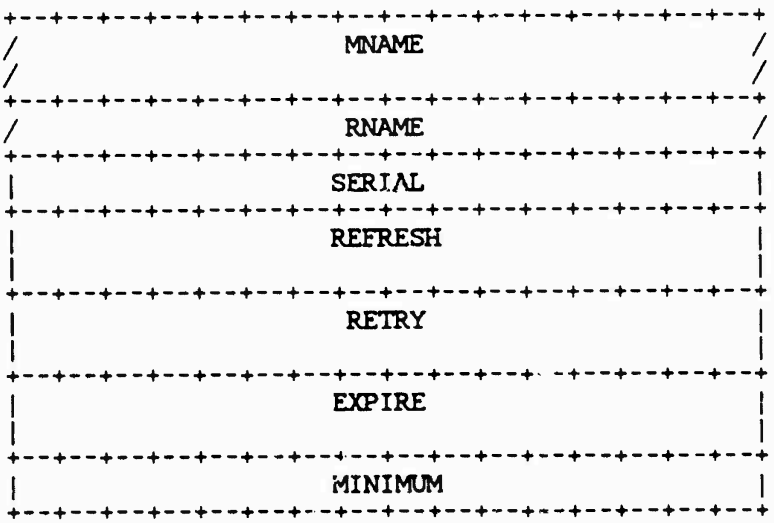


where:

PTRDNAME - A compressed domain name which points to some location in the domain name space.

PTR records cause no additional section processing. These RRs are used in special domains to point to some other location in the domain space. These records are simple data, and don't imply any special processing similar to that performed by CNAME, which identifies aliases. Appendix 3 discusses the use of these records in the ARPA Internet address domain.

SOA RDATA format



where:

MNAME - The domain name of the name server that was the original source of data for this zone.

RNAME - A domain name which specifies the mailbox of the person responsible for this zone.

SERIAL - The unsigned 16 bit version number of the of the original copy of the zone. This value wraps and should be compared using sequence space arithmetic.

REFRESH - The unsigned 32 bit time interval before the zone should be refreshed.

RETRY - The unsigned 32 bit time interval that should elapse before a failed refresh should be retried.

EXPIRE - A 32 bit time value that specifies the upper limit on the time interval that can elapse before the zone is no longer authoritative.

MINIMUM - The unsigned 16 bit minimum TTL field that should be exported with any RR from this zone (other than the SOA itself).

SOA records cause no additional section processing. The RDATA

section of a SOA line in a master file is a standard printed domain name for MNAME, a standard X@Y mailbox specification for RNAME, and decimal numbers for the remaining parameters.

All times are in units of seconds.

Most of these fields are pertinent only for name server maintenance operations. However, MINIMUM is used in all query operations that retrieve RRs from a zone. Whenever a RR is sent in a response to a query, the TTL field is set to the maximum of the TTL field from the RR and the MINIMUM field in the appropriate SOA. Thus MINIMUM is a lower bound on the TTL field for all RRs in a zone. RRs in a zone are never discarded due to timeout unless the whole zone is deleted. This prevents partial copies of zones.

Appendix 3 - Internet specific field formats and operations

Message transport

The Internet supports name server access using TCP [10] on server port 53 (decimal) as well as datagram access using UDP [11] on UDP port 53 (decimal). Messages sent over TCP virtual circuits are preceded by an unsigned 16 bit length field which describes the length of the message, excluding the length field itself.

```
+-------------------------------------------------+
|                                                 |
|              ***** WARNING *****                |
|                                                 |
|  The following formats are preliminary and      |
| are included for purposes of explanation only.  |
| In particular, new RR types will be added,      |
| and the size, position, and encoding of         |
| fields are subject to change.                   |
|                                                 |
+-------------------------------------------------+
```

A RDATA format

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ADDRESS                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

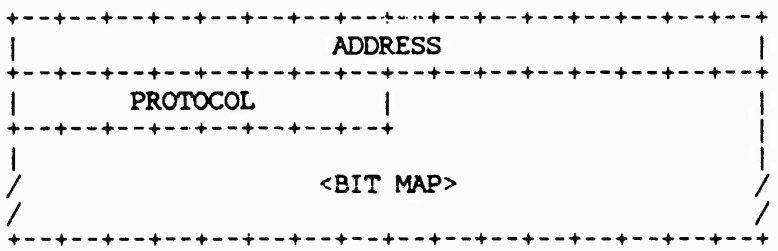
where:

ADDRESS - A 32 bit ARPA internet address

Hosts that have multiple ARPA Internet addresses will have multiple A records.

A records cause no additional section processing. The RDATA section of an A line in a master file is an Internet address expressed as four decimal numbers separated by dots without any imbedded spaces (e.g., "10.2.0.52" or "192.0.5.6").

WKS RDATA format



where:

ADDRESS - An 32 bit ARPA Internet address

PROTOCOL - An 8 bit IP protocol number

<BIT MAP> - A variable length bit map. The bit map must be a multiple of 8 bits long.

The WKS record is used to describe the well known services supported by a particular protocol on a particular internet address. The PROTOCOL field specifies an IP protocol number, and the bit map has one bit per port of the specified protocol. The first bit corresponds to port 0, the second to port 1, etc. If less than 256 bits are present, the remainder are assumed to be zero. The appropriate values for ports and protocols are specified in [13].

For example, if PROTOCOL=TCP (6), the 26th bit corresponds to TCP port 25 (SMTP). If this bit is set, a SMTP server should be listening on TCP port 25; if zero, SMTP service is not supported on the specified address.

The anticipated use of WKS RRs is to provide availability information for servers for TCP and UDP. If a server supports both TCP and UDP, or has multiple Internet addresses, then multiple WKS RRs are used.

WKS RRs cause no additional section processing. The RDATA section of a WKS record consists of a decimal protocol number followed by mnemonic identifiers which specify bits to be set to 1.

IN-ADDR special domain

The ARPA internet uses a special domain to support gateway location and ARPA Internet address to host mapping. The intent of this domain is to allow queries to locate all gateways on a

particular network in the ARPA Internet, and also to provide a guaranteed method to perform host address to host name mapping.

Note that both of these services are similar to functions that could be performed by inverse queries; the difference is that this part of the domain name space is structured according to address, and hence can guarantee that the appropriate data can be located without an exhaustive search of the domain space. It is anticipated that the special tree will be used by ARPA Internet resolvers for all gateway location services, but that address to name resolution will be performed by first trying the inverse query on the local name server database followed by a query in the special space if the inverse query fails.

The domain is a top level domain called IN-ADDR whose substructure follows the ARPA Internet addressing structure.

Domain names in the IN-ADDR domain are defined to have up to four labels in addition to the IN-ADDR label. Each label is a character string which expresses a decimal value in the range 0-255 (with leading zeros omitted except in the case of a zero octet which is represented by a single zero). These labels correspond to the 4 octets of an ARPA Internet address.

Host addresses are represented by domain names that have all four labels specified. Thus data for ARPA Internet address 10.2.0.52 is located at domain name 52.0.2.10.IN-ADDR. The reversal, though awkward to read, allows zones to follow the natural grouping of hosts within networks. For example, 10.IN-ADDR can be a zone containing data for the ARPANET, while 26.IN-ADDR can be a separate zone for MILNET. Address nodes are used to hold pointers to primary host names in the normal domain space.

Network addresses correspond to some of the non-terminal nodes in the IN-ADDR tree, since ARPA Internet network numbers are either 1, 2, or 3 octets. Network nodes are used to hold pointers to primary host names (which happen to be gateways) in the normal domain space. Since a gateway is, by definition, on more than one network, it will typically have two or more network nodes that point at the gateway. Gateways will also have host level pointers at their fully qualified addresses.

Both the gateway pointers at network nodes and the normal host pointers at full address nodes use the PTR RR to point back to the primary domain names of the corresponding hosts.

For example, part of the IN-ADDR domain will contain information about the ISI to MILNET and MIT gateways, and hosts F.ISI.ARPA and MULTICS.MIT.ARPA. Assuming that ISI gateway has addresses

10.2.0.22 and 26.0.0.103, and a name MILNET-GW.ISI.ARPA, and the MIT gateway has addresses 10.0.0.77 and 18.10.0.4 and a name GW.MIT.ARPA, the domain database would contain:

```
10.IN-ADDR            PTR  IN MILNET-GW.ISI.ARPA
10.IN-ADDR            PTR  IN GW.MIT.ARPA
18.IN-ADDR            PTR  IN GW.MIT.ARPA
26.IN-ADDR            PTR  IN MILNET-GW.ISI.ARPA
22.0.2.10.IN-ADDR     PTR  IN MILNET-GW.ISI.ARPA
103.0.0.26.IN-ADDR    PTR  IN MILNET-GW.ISI.ARPA
77.0.0.10.IN-ADDR     PTR  IN GW.MIT.ARPA
4.0.10.18.IN-ADDR     PTR  IN GW.MIT.ARPA
52.0.2.10.IN-ADDR     PTR  IN F.ISI.ARPA
6.0.0.10.IN-ADDR      PTR  IN MULTICS.MIT.ARPA
```

Thus a program which wanted to locate gateways on net 10 would originate a query of the form QTYPE=PTR, QCLASS=IN, QNAME=10.IN-ADDR. It would receive two RRs in response:

```
10.IN-ADDR            PTR  IN MILNET-GW.ISI.ARPA
10.IN-ADDR            PTR  IN GW.MIT.ARPA
```

The program could then originate QTYPE=A, QCLASS=IN queries for MILNET-GW.ISI.ARPA and GW.MIT.ARPA to discover the ARPA Internet addresses of these gateways.

A resolver which wanted to find the host name corresponding to ARPA Internet host address 10.0.0.6 might first try an inverse query on the local name server, but find that this information wasn't available. It could then try a query of the form QTYPE=PTR, QCLASS=IN, QNAME=6.0.0.10.IN-ADDR, and would receive:

```
6.0.0.10.IN-ADDR      PTR  IN MULTICS.MIT.ARPA
```

Several cautions apply to the use of these services:

Since the IN-ADDR special domain and the normal domain for a particular host or gateway will be in different zones, the possibility exists that that the data may be inconsistent.

Gateways will often have two names in separate domains, only one of which can be primary.

Systems that use the domain database to initialize their routing tables must start with enough gateway information to guarantee that they can access the appropriate name server.

The gateway data only reflects the existence of a gateway in a

manner equivalent to the current HOSTS.TXT file. It doesn't replace the dynamic availability information from GGP or EGP.

REFERENCES and BIBLIOGRAPHY

[1] E. Feinler, K. Harrenstien, Z. Su, and V. White, "DOD Internet Host Table Specification", RFC 810, Network Information Center, SRI International, March 1982.

[2] J. Postel, "Computer Mail Meeting Notes", RFC 805, USC/Information Sciences Institute, February 1982.

[3] Z. Su, and J. Postel, "The Domain Naming Convention for Internet User Applications", RFC 819, Network Information Center, SRI International, August 1982.

[4] Z. Su, "A Distributed System for Internet Name Service", RFC 830, Network Information Center, SRI International, October 1982.

[5] K. Harrenstien, and V. White, "NICNAME/WHOIS", RFC 812, Network Information Center, SRI International, March 1982.

[6] M. Solomon, L. Landweber, and D. Neuhengen, "The CSNET Name Server", Computer Networks, vol 6, nr 3, July 1982.

[7] K. Harrenstien, "NAME/FINGER", RFC 742, Network Information Center, SRI International, December 1977.

[8] J. Postel, "Internet Name Server", IEN 116, USC/Information Sciences Institute, August 1979.

[9] K. Harrenstien, V. White, and E. Feinler, "Hostnames Server", RFC 811, Network Information Center, SRI International, March 1982.

[10] J. Postel, "Transmission Control Protocol", RFC 793, USC/Information Sciences Institute, September 1981.

[11] J. Postel, "User Datagram Protocol", RFC 768, USC/Information Sciences Institute, August 1980.

[12] J. Postel, "Simple Mail Transfer Protocol", RFC 821, USC/Information Sciences Institute, August 1980.

[13] J. Reynolds, and J. Postel, "Assigned Numbers", RFC 870, USC/Information Sciences Institute, October 1983.

[14] P. Mockapetris, "Domain names - Concepts and Facilities," RFC 882, USC/Information Sciences Institute, November 1983.

INDEX

where <response key> is a keyword indicating the nature of the response, and the rest of the response is interpreted in the context of the key.

NOTE: Care should be taken to interpret the nature of the reply (e.g, single record or multiple record), so that no confusion about the state of the reply results. An "ALL" request will likely return several hundred or more records of all types, whereas "HNAME" or "HADDR" will usually return one HOST record.

COMMAND/RESPONSE KEYS

The currently defined command keywords are listed below. NOTE: Because the server and the features available will evolve with time, the HELP command should be used to obtain the most recent summary of implemented features, changes, or new commands.

Keyword	Response
HELP	This information.
VERSION	"VERSION: <string>" where <string> will be different for each version of the host table.
HNAME <hostname>	One or more matching host table entries.
HADDR <hostaddr>	One or more matching host table entries.
ALL	The entire host table.
ALL-OLD	The entire host table without domain style names.
DOMAINS	The entire top-level domain table (domains only).
ALL-DOM	Both the entire domain table and the host table.
ALL-INGWAY	All known gateways in TENEX/TOPS-20 INTERNET.GATEWAYS format.

Remember that the server accepts only a single command line and returns only a single response before closing the connection. HNAME and HADDR are useful for looking up a specific host by name or address; VERSION can be used by automated processes to see whether a "new" version of the host table exists without having to transfer the

Network Working Group
Request for Comments: 953
Obsoletes: RFC 811

K. Harrenstien (SRI)
M. Stahl (SRI)
E. Feinler (SRI)
October 1985

HOSTNAME SERVER

STATUS OF THIS MEMO

This RFC is the official specification of the Hostname Server Protocol. This edition of the specification includes minor revisions to RFC 811 which brings it up to date. Distribution of this memo is unlimited.

INTRODUCTION

The NIC Internet Hostname Server is a TCP-based host information program and protocol running on the SRI-NIC machine. It is one of a series of internet name services maintained by the DDN Network Information Center (NIC) at SRI International on behalf of the Defense Communications Agency (DCA). The function of this particular server is to deliver machine-readable name/address information describing networks, gateways, hosts, and eventually domains, within the internet environment. As currently implemented, the server provides the information outlined in the DoD Internet Host Table Specification [See RFC-952]. For a discussion of future developments see also RFC-921 concerning the Domain Name System.

PROTOCOL

To access this server from a program, establish a TCP connection to port 101 (decimal) at the service host, SRI-NIC.ARPA (26.0.0.73 or 10.0.0.51). Send the information request (a single line), and read the resulting response. The connection is closed by the server upon completion of the response, so only one request can be made for each connection.

QUERY/RESPONSE FORMAT

The name server accepts simple text query requests of the form

<command key> <argument(s)> [<options>]

where square brackets ("[]") indicate an optional field. The command key is a keyword indicating the nature of the request. The defined keys are explained below.

The response, on the other hand, is of the form

<response key> : <rest of response>

whole table. Note, however, that the returned version string is only guaranteed to be unique to each version, and nothing should currently be assumed about its format.

Response Keys:

ERR	entry not found, nature of error follows
NET	entry found, rest of entry follows
GATEWAY	entry found, rest of entry follows
HOST	entry found, rest of entry follows
DOMAIN	entry found, rest of entry follows
BEGIN	followed by multiple entries
END	done with BEGIN block of entries

More keywords will be added as new needs are recognized. A more detailed description of the allowed requests/responses follows.

QUERY/RESPONSE EXAMPLES

1. HNAME Query - Given a name, find the entry or entries that match the name. For example:

HNAME SRI-NIC.ARPA <CRLF>

where <CRLF> is a carriage return/ linefeed, and 'SRI-NIC.ARPA' is a host name

The likely response is:

```
HOST : 26.0.0.73, 10.0.0.51 : SRI-NIC.ARPA,SRI-NIC,NIC :
       DEC-2060 : TOPS20 : TCP/TELNET,TCP/SMTP,TCP/TIME,TCP/FTP,
       TCP/ECHO,ICMP :
```

A response may stretch across more than one line. Continuation lines always begin with at least one space.

2. HADDR Query - Given an internet address (as specified in RFC 796) find the entry or entries that match that address. For example:

HADDR 26.0.0.73 <CRLF>

where <CRLF> is a carriage return/ linefeed, and '26.0.0.73' is a host address.

The likely response is the same as for the previous HNAME request.

3. ALL Query - Deliver the entire internet host table in a machine-readable form. For example:

ALL <CRLF> ;where <CRLF> is a carriage return/linefeed

The likely response is the keyword 'BEGIN' followed by a colon ':', followed by the entire internet host table in the format specified in RFC-952, followed by 'END:'.

ERROR HANDLING

ERR Reply - may occur on any query, and should be permitted in any access program using the name server. Errors are of the form

ERR : <code> : <string> :
as in
ERR : NAMNFD : Name not found :

The error code is a unique descriptor, limited to 8 characters in length for any given error. It may be used by the access program to identify the error and, in some cases, to handle it automatically. The string is an accompanying message for a given error for that case where the access program simply logs the error message. Current codes and their associated interpretations are

NAMNFD	Name not found; name not in table
ADRNFD	Address not found; address not in table
ILLCOM	Illegal command; command key not recognized
TMPSYS	Temporary system failure, try again later

REFERENCES

1. Harrenstien, K., Stahl, M., and Feinler, E., "Official DoD Internet Host Table Specification," RFC-952, DDN Network Information Center, SRI International, October 1985.

2. Pickens, J., Feinler, E., and Mathis, J., "The NIC Name Server," A Datagram-based Information Utility, RFC-756, Network Information Center, SRI International, July 1979.

3. Postel, J., "Address Mappings," RFC-796, Information Sciences Institute, University of Southern California, Marina del Rey, September 1981.

4. Postel, J., "Domain Name System Implementation Schedule", RFC-921, Information Sciences Institute, University of Southern California, Marina del Rey, October 1984.

RFC 953
Hostname Server

October 1985

IEN 133

The TFTP Protocol

January 29, 1980

Karen R. Sollins

Summary

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each nonterminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

Acknowledgements

The protocol was originally designed by Noel Chiappa, and was redesigned by him, Bob Baldwin and Dave Clark, with comments from Steve Szymanski. The original version of this document was written by Bob Baldwin. The current version of the document includes modifications suggested by Noel Chiappa, Dave Clark, Liza Martin and the author. The acknowledgement and retransmission scheme was inspired by TCP, and the error mechanism was suggested by PARC's EFTP abort message.

1: Purpose

TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It is built on top of the Internet User Datagram protocol (UDP or Datagram) [2] so it may be used to move files between machines on different networks. It is designed to be small and easy to implement. Therefore, it lacks most of the features of a regular FTP. The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8 bit bytes of data.

Three modes of transfer are currently supported: netascii (1); binary, raw 8 bit bytes; mail, netascii characters sent to a user rather than a file. Additional modes can be defined by pairs of cooperating hosts.

2: Overview of the Protocol

Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be

(1) This is ascii as defined in "USA Standard Code for Information Interchange" [1] with the modifications specified in "Telnet Protocol Specification" [3]. Note that it is 8 bit ascii. The term "netascii" will be used throughout this document to mean this particular version of ascii.

acknowledged by an acknowledgment packet before the next packet can be sent. A packet of less than 512 bytes signals termination of a transfer. If a packet gets lost in the network, the intended recipient will timeout and may retransmit his last packet (which may be data or an acknowledgment), thus causing the sender of the lost packet to retransmit that lost packet. The sender has to keep just one packet on hand for retransmission, since the lock step acknowledgment guarantees that all older packets have been received. Notice that both machines involved in a transfer are considered senders and receivers. One sends data and receives acknowledgments, the other sends acknowledgments and receives data.

Most errors cause termination of the connection. An error is signalled by sending an error packet. This packet is not acknowledged, and not retransmitted (i.e., a TFTP server or user may terminate after sending an error message), so the other end of the connection may not get it. Therefore timeouts are used to detect such a termination when the error packet has been lost. Errors are caused by three types of events: not being able to satisfy the request (e.g., file not found, or access violation), receiving a packet which cannot be explained by a delay or duplication in the network (e.g. an incorrectly formed packet), and losing access to a necessary resource (e.g., disc full, or source file truncated during transfer).

TFTP recognizes only one type of error that does not cause termination, the source port of a received packet being incorrect. In

this case an error packet is sent to the originating host. See the section on the Initial Connection Protocol for more details.

This protocol is very restrictive, but that makes it easier to implement. For example, the fixed length blocks make allocation straight forward, and the lock step acknowledgement provides flow control and eliminates the need to reassemble files.

3: Relation to other Protocols

As mentioned TFTP is designed to be implemented on top of the Datagram protocol. Since Datagram is implemented on the Internet protocol, packets will have an Internet header, a Datagram header, and a TFTP header. Additionally, the packets may have a header (LNI, ARPA header, etc.) to allow them through the local transport medium. As shown in Figure 1, the order of the contents of a packet will be local medium header, if used, Internet header, Datagram header, TFTP header, followed by the remainder of the TFTP packet. (This may or may not be data depending on the type of packet as specified in the TFTP header.) TFTP does not specify any of the values in the Internet header.

The source and destination port fields of the Datagram header (its format is given in the appendix) are used by TFTP and the length field reflects the size of the TFTP packet. The transfer identifiers (TID's) used by TFTP are passed to the Datagram layer to be used as ports. Therefore for they must be between 0 and 65,535. The initialization of TID's is discussed in the section on initial connection protocol.

The TFTP header consists of a 2 byte opcode field which indicates the packet's type (e.g., DATA, ERROR, etc.) These opcodes and the formats of the various types of packets are discussed further in the section on TFTP packets.

Figure 1. Order of Headers

4: Initial Connection Protocol

A transfer is established by sending a request (WRQ to write onto a foreign file system, or RRQ to read from it), and receiving a positive reply, an acknowledgment packet for write, or the first data packet for read. In general an acknowledgment packet will contain the block number of the data packet being acknowledged. Each data packet has associated with it a block number; block numbers are consecutive and begin with one. Since the positive response to a write request is an acknowledgment packet, in this special case the block number will be zero. (Normally, since an acknowledgment packet is acknowledging a data packet, the acknowledgment packet will contain the block number of the data packet being acknowledged.) If the reply is an error packet, then the request is denied for the reason stated in the error packet.

In order to create a connection, TID's to be used for the duration of the connection are chosen by the two ends of that connection. The TID's chosen for a connection should be randomly chosen, so that the

probability that the same number is chosen twice in immediate succession is very low. Every packet has associated with it two TID's, the source TID and the destination TID. A requesting host chooses its source TID as described above, and sends its initial request to the known TID 69 (105 octal) on the serving host. The response to the request, under normal operation, uses a TID chosen by the server as its source TID and the TID chosen for the previous message by the requestor as its destination TID. The two chosen TID's are then used for the remainder of the transfer.

As an example, the following shows the steps used to establish a connection to write a file. Note that WRQ, ACK, and DATA are the names of the write request, acknowledgment, and data types of packets respectively. The Appendix contains a similar example for reading a file.

1. Host A sends a "WRQ" to host B with
   source= A's TID, destination= 69.
2. Host B sends a "ACK" (with block number= 0) to host A with
   source= B's TID, destination= A's TID.
3. Host A sends a "DATA" (with block number= 1) to host B with
   source= A's TID, destination= B's TID.
4. Host B sends a "ACK" (with block number= 1) to host A with
   source= B's TID, destination= A's TID.

In step three, and in all succeeding steps, the hosts should make sure that the source TID matches the value that was agreed on in step 2.

If it doesn't match, an error packet should be sent to the originator, but the connection should not be aborted. The following example demonstates the problem this and the randomly chosen TID's are trying to solve.

Host A sends a request to host B. Somewhere in the network, the request packet is duplicated, and as a result two acknowledgments are returned to host A, with different TID's chosen on host B in repsonse to the two requests. When the first response arrives, host A continues the connection. When the second response to the request arrives, it should be rejected, but there is no reason to terminate the first connection. Therefore, if different TID's are chosen on host B and host A checks the source TID's of the messages it receives, the first connection can be maintained while the second is rejected.

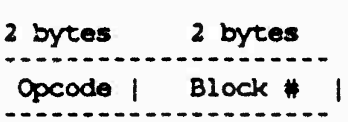
5: TFTP Packets

TFTP supports five types of packets, all of which have been mentioned above:

opcode	operation
1	Read request (RRQ)
2	Write request (WRQ)
3	Acknowledgment (ACK)
4	Data (DATA)
5	Error (ERROR)

Data is actually transferred in DATA packets depicted in Figure 3. DATA packets (opcode = 4) have a block number and data field. The block numbers on data packets begin with one and increase by one for each new block of data. This restriction allows the program to use a single number to discriminate between new packets and duplicates. The data field is from zero to 512 bytes long. If it is 512 bytes long, the block is not the last block of data; if it is from zero to 511 bytes long, it signals the last data packet. (See the section on Normal Termination for details.)

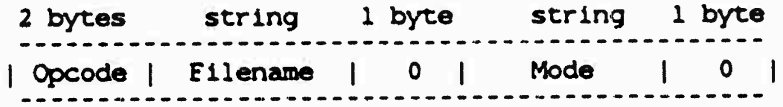
Figure 4. ACK packet



All packets other than those used for termination are acknowledged individually. Sending a DATA packet is an acknowledgment for the ACK packet of the previous DATA packet. The WRQ and DATA packets are acknowledged by ACK or ERROR packets, while RRQ and ACK packets are acknowledged by DATA or ERROR packets. Figure 4 depicts an ACK packet; the opcode is 3. The block number in an ACK echoes the block number of the DATA packet being acknowledged. A WRQ is acknowledged with an ACK packet having a block number of zero.

The TFTP header of a packet contains the opcode associated with that packet.

Figure 2. RRQ/WRQ



RRQ and WRQ packets (opcodes 1 and 2 respectively) have the format shown in Figure 2. The file name is a sequence of bytes in netascii terminated by a zero byte. The mode field contains the string "netascii", "binary", or "mail" in netascii indicating the three modes defined in the protocol. A host which receives netascii mode data must translate the data to its own format. Presumably, every host will translate its character set to and from netascii. Binary mode allows the two communicating hosts to impose their own interpretation on the data being transmitted; between similar machines binary mode can be used to avoid the conversion overhead. If a host receives a binary file and then returns it, the returned file must be identical to the file it received. Mail mode uses the name of a mail recipient in place of a file and must begin with a WRQ. Otherwise it is identical to netascii mode.

Figure 3. DATA

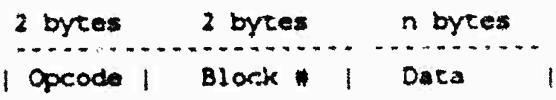
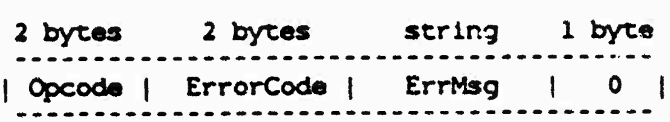


Figure 5. ERROR packet



An ERROR packet (opcode 5) takes the form depicted in Figure 5. An ERROR packet can be the acknowledgment of any other type of packet. The error code is a small integer indicating the nature of the error. A table of its values and meanings is given in the appendix. The error message is intended for human consumption, and should be in netascii. Like all other strings, it is terminated with a zero byte.

6: Normal Termination

The end of a transfer is marked by a DATA packet that contains between 0 and 511 bytes of data (i.e. Datagram length < 516). This packet is acknowledged by an ACK packet like all other DATA packets. The final ACK packet is never retransmitted; the host acknowledging the final DATA packet may terminate its side of the connection on sending the final ACK. On the other hand, the host sending the last DATA must retransmit it until the packet is acknowledged or the sending host times out. If the response is an ACK, the transmission was completed successfully. If it is an ERROR (unknown transfer ID), or the sender of the data times out and is not prepared to retransmit any more, the transfer may still have been completed successfully, after which the acknowledger may have experienced a problem. It is also possible in

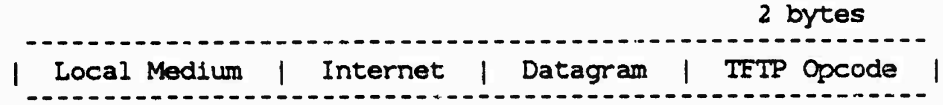
this case that the transfer was unsuccessful. In any case, the connection has been closed.

7: Premature Termination

If a request can not be granted, or some error occurs during the transfer, then an ERROR packet (opcode 5) is sent. This is only a courtesy since it will not be retransmitted or acknowledged, so it may never be received. Timeouts must also be used to detect errors.

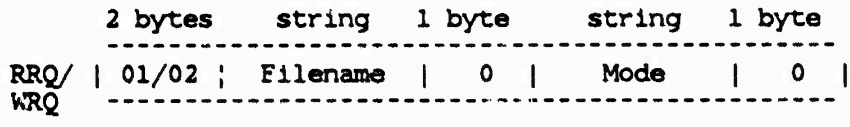
APPENDIX

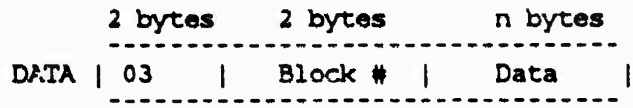
Order of Headers

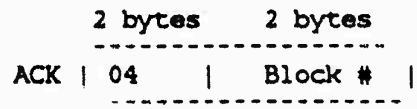


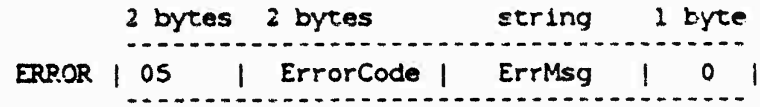
TFTP Formats

Type Op # Format without header









Initial Connection Protocol for reading a file

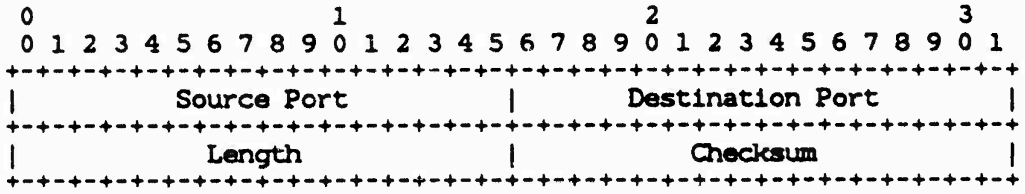
1. Host A sends a "RRQ" to host B with
   source= A's TID, destination= 69.
2. Host B sends a "DATA" (with block number= 1) to host A with
   source= B's TID, destination= A's TID.
3. Host A sends an "ACK" (with block number= 1) to host B with
   source= A's TID, destination= B's TID.

Error Codes

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disc full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.

Internet User Datagram Header

Format



Values of Fields

Source Port	Picked by originator of packet.
Dest. Port	Picked by destination machine (69 for RRQ or WRQ).
Length	Number of bytes in packet after Datagram header.
Checksum	Reference 2 describes rules for computing checksum. Field contains zero if unused.

Note: TFTP passes transfer identifiers (TID's) to the Internet User Datagram protocol to be used as the source and destination ports.

References

1. USA Standard Code for Information Interchange, USASI X3.4-1968.

2. Postel, Jon., "User Datagram Protocol," IEN 88, May 2, 1979.

3. "Telnet Protocol Specification," RFC552, NIC 18639, August, 1973.

Network Working Group
Request for Comments: 913

Mark K. Lottor
MIT
September 1984

Simple File Transfer Protocol

STATUS OF THIS MEMO

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

INTRODUCTION

SFTP is a simple file transfer protocol. It fills the need of people wanting a protocol that is more useful than TFTP but easier to implement (and less powerful) than FTP. SFTP supports user access control, file transfers, directory listing, directory changing, file renaming and deleting.

SFTP can be implemented with any reliable 8-bit byte stream oriented protocol, this document describes its TCP specification. SFTP uses only one TCP connection; whereas TFTP implements a connection over UDP, and FTP uses two TCP connections (one using the TELNET protocol).

THE PROTOCOL

SFTP is used by opening a TCP connection to the remote hosts' SFTP port (115 decimal). You then send SFTP commands and wait for replies. SFTP commands sent to the remote server are always 4 ASCII letters (of any case) followed by a space, the argument(s), and a <NULL>. The argument can sometimes be null in which case the command is just 4 characters followed by <NULL>. Replies from the server are always a response character followed immediately by an ASCII message string terminated by a <NULL>. A reply can also be just a response character and a <NULL>.

<command> : = <cmd> [<SPACE> <args>] <NULL>

<cmd> : = USER ! ACCT ! PASS ! TYPE ! LIST ! CDIR
KILL ! NAME ! DONE ! RETR ! STOR

<response> : = <response-code> [<message>] <NULL>

<response-code> : = + | - | | !

<message> can contain <CRLF>

Commands that can be sent to the server are listed below. The server

replies to each command with one of the possible response codes listed under each message. Along with the response, the server should optionally return a message explaining the error in more detail. Example message texts are listed but do not have to be followed. All characters used in messages are ASCII 7-bit with the high-order bit zero, in an 8 bit field.

The response codes and their meanings:

+ Success.

- Error.

  An error occurred while processing your command.

  Number.

  The number-sign is followed immediately by ASCII digits representing a decimal number.

! Logged in.

  You have sent enough information to be able to log yourself in. This is also used to mean you have sent enough information to connect to a directory.

To use SFTP you first open a connection to the remote SFTP server. The server replies by sending either a positive or negative greeting, such as:

+MIT-XX SFTP Service

  (the first word should be the host name)

-MIT-XX Out to Lunch

If the server send back a '-' response it will also close the connection, otherwise you must now send a USER command.

USER user-id

Your userid on the remote system.

The reply to this command will be one of:

!<user-id> logged in

Meaning you don't need an account or password or you specified a user-id not needing them.

+User-id valid, send account and password

-Invalid user-id, try again

If the remote system does not have user-id's then you should send an identification such as your personal name or host name as the argument, and the remote system would reply with '+'.

ACCT account

The account you want to use (usually used for billing) on the remote system.

Valid replies are:

! Account valid, logged-in

Account was ok or not needed. Skip the password.

+Account valid, send password

Account ok or not needed. Send your password next.

-Invalid account, try again

PASS password

Your password on the remote system.

Valid replies are:

! Logged in

Password is ok and you can begin file transfers.

+Send account

Password ok but you haven't specified the account.

Wrong password, try again

You cannot specify any of the following commands until you receive a '!' response from the remote system.

TYPE { A | B | C }

The mapping of the stored file to the transmission byte stream is controlled by the type. The default is binary if the type is not specified.

A - ASCII

The ASCII bytes are taken from the file in the source system, transmitted over the connection, and stored in the file in the destination system.

The data is the 7-bit ASCII codes, transmitted in the low-order 7 bits of 8-bit bytes. The high-order bit of the transmission byte must be zero, and need not be stored in the file.

The data is "NETASCII" and is to follow the same rules as data sent on Telnet connections. The key requirement here is that the local end of line is to be converted to the pair of ASCII characters CR and LF when transmitted on the connection.

For example, TOPS-20 machines have 36-bit words. On TOPS-20 machines, The standard way of labeling the bits is 0 through 35 from high-order to low-order. On TOPS-20 the normal way of storing ASCII data is to use 5 7-bit bytes per word. In ASCII mode, the bytes transmitted would be [0-6], [7-13], [14-20], [21-27], [28-34], (bit 35 would not be transmitted), each of these 7-bit quantities would be transmitted as the low-order 7 bits of an 8-bit byte (with the high-order bit zero).

For example, one disk page of a TOPS-20 file is 512 36-bit words. But using only 35 bits per word for 7-bit bytes, a page is 17920 bits or 2560 bytes.

B - BINARY

The 8-bit bytes are taken from the file in the source system, transmitted over the connection, and stored in the file in the destination system.

The data is in 8-bit units. In systems with word sizes which are not a multiple of 8, some bits of the word will not be transmitted.

For example, TOPS-20 machines have 36-bit words. In binary mode, the bytes transmitted would be [0-7], [8-15], [16-23], [24-31], (bits 32-35 would not be transmitted).

For example, one disk page of a TOPS-20 file is 512 36-bit words. But using only 32 bits per word for 8-bit bytes, a page is 16384 bits or 2048 bytes.

C - CONTINUOUS

The bits are taken from the file in the source system continuously, ignoring word boundaries, and sent over the connection packed into 8-bit bytes. The destination system stores the bits received into the file continuously, ignoring word boundaries.

For systems on machines with a word size that is a multiple of 8 bits, the implementation of binary and continuous modes should be identical.

For example, TOPS-20 machines have 36-bit words. In continuous mode, the bytes transmitted would be [first word, bits 0-7], [first word, bits 8-15], [first word, bits 16-23], [first word, bits 24-31], [first word, bits 32-35 + second word, bits 0-3], [second word, bits 4-11], [second word, bits 12-19], [second word, bits 20-27], [second word, bits 28-35], then the pattern repeats.

For example, one disk page of a TOPS-20 file is 512 36-bit words. This is 18432 bits or 2304 8-bit bytes.

Replies are:

+Using (Ascii | Binary | Continuous) mode

-Type not valid

LIST { F | V } directory-path

A null directory-path will return the current connected directory listing.

F specifies a standard formatted directory listing.

An error reply should be a '-' followed by the error message from the remote systems directory command. A directory listing is a '+' followed immediately by the current directory path specification and a <CRLF>. Following the directory path is a single line for each file in the directory. Each line is just the file name followed by <CRLF>. The listing is terminated with a <NULL> after the last <CRLF>.

V specifies a verbose directory listing.

An error returns '-' as above. A verbose directory listing is a '+' followed immediately by the current directory path specification and a <CRLF>. It is then followed by one line per file in the directory (a line ending in <CRLF>). The line returned for each file can be of any format. Possible information to return would be the file name, size, protection, last write date, and name of last writer.

CDIR new-directory

This will change the current working directory on the remote host to the argument passed.

Replies are:

!Changed working dir to <new-directory>

-Can't connect to directory because: (reason)

+directory ok, send account/password

If the server replies with '+' you should then send an ACCT or PASS command. The server will wait for ACCT or PASS commands until it returns a '-' or '!' response.

Replies to ACCT could be:

!Changed working dir to <new-directory>

+account ok, send password

-invalid account

Replies to PASS could be:

!Changed working dir to <new-directory>

+password ok, send account

-invalid password

KILL file-spec

This will delete the file from the remote system.

Replies are:

+<file-spec> deleted

-Not deleted because (reason)

NAME old-file-spec

Renames the old-file-spec to be new-file-spec on the remote system.

Replies:

+File exists

-Can't find <old-file-spec>

NAME command is aborted, don't send TOBE.

If you receive a '+' you then send:

TOBE new-file-spec

The server replies with:

+<old-file-spec> renamed to <new-file-spec>

-File wasn't renamed because (reason)

DONE

Tells the remote system you are done.

The remote system replies:

+(the message may be charge/accounting info)

and then both systems close the connection.

RETR file-spec

Requests that the remote system send the specified file.

Receiving a '-' from the server should abort the RETR command and the server will wait for another command.

The reply from the remote system is:

<number-of-bytes-that-will-be-sent> (as ascii digits)

-File doesn't exist

You then reply to the remote system with:

SEND (ok, waiting for file)

The file is then sent as a stream of exactly the number of 8-bit bytes specified. When all bytes are received control passes back to you (the remote system is waiting for the next command). If you don't receive a byte within a reasonable amount of time you should abort the file transfer by closing the connection.

STOP (You don't have enough space to store file)

Replies could be:

+ok, RETR aborted

You are then ready to send another command to the remote host.

STOR { NEW | OLD | APP } file-spec

Tells the remote system to receive the following file and save it under that name.

Receiving a '-' should abort the STOR command sequence and the server should wait for the next command.

NEW specifies it should create a new generation of the file and not delete the existing one.

Replies could be:

+File exists, will create new generation of file

+File does not exist, will create new file

-File exists, but system doesn't support generations

OLD specifies it should write over the existing file, if any, or else create a new file with the specified name.

Replies could be:

+Will write over old file

+Will create new file

(OLD should always return a '+')

APP specifies that what you send should be appended to the file on the remote site. If the file doesn't exist it will be created.

Replies could be:

+Will append to file

+Will create file

(APP should always return a '+')

You then send:

SIZE <number-of-bytes-in-file> (as ASCII digits)

where number-of-bytes-in-file

is the exact number of 8-bit bytes you will be sending.

The remote system replies:

+ok, waiting for file

You then send the file as exactly the number of bytes specified above.

When you are done the remote system should reply:

+Saved <file-spec>

-Couldn't save because (reason)

-Not enough room, don't send it

This aborts the STOR sequence, the server is waiting for your next command.

You are then ready to send another command to the remote host.

AN EXAMPLE

An example file transfer. 'S' is the sender, the user process. 'R' is the reply from the remote server. Remember all server replies are terminated with <NULL>. If the reply is more than one line each line ends with a <CRLF>.

```
R: (listening for connection)
S: (opens connection to R)
R: +MIT-XX SFTP Service
S: USER MKL
R: +MKL ok, send password
S: PASS foo
R: ! MKL logged in
S: LIST F PS: <MKL>
R: +PS: <MKL>
   Small.File
   Large.File
S: LIST V
R: +PS: <MKL>
   Small.File  1        69(7)  P775240  2-Aug-84 20:08  MKL
   Large.File  100  255999(8)  P770000  9-Dec-84 06:04  MKL
S: RETR SMALL.FILE
R:  69
S: SEND
R: This is a small file, the file is sent without
   a terminating null.
S: DONE
R: +MIT-XX closing connection
```

RFC 913 September 1984
Simple File Transfer Protocol

EDITORS NOTE

Mark Lotter receives full credit for all the good ideas in this memo. As RFC editor, i have made an number of format changes, a few wording changes, and one or two technical changes (mostly in the TYPEs). I accept full responsibility for any flaws i may have introduced.

A draft form of this memo was circulated for comments. I will attempt to list the issues raised and summarize the pros and cons, and resolution for each.

ASCII Commands vs Binary Operation Codes

The ASCII command style is easier to debug, the extra programming cost in minimal, the extra transmission cost is trivial.

Binary operation codes are more efficient, and a few days of debugging should not out weigh years of use.

Resolution: I have kept the ASCII Commands.

Additional Modes

Pro: For some machines you can't send all the bits in a word using this protocol. There should be some additional mode to allow it.

Con: Forget it, this is supposed to be SIMPLE file transfer. If you need those complex modes use real FTP.

Resolution: I have added the Continuous mode.

CRLF Conversion

Pro: In ASCII type, convert the local end of line indicator to CRLF on the way out of the host and onto the network.

Con: If you require that you have to look at the bytes as you send them, otherwise you can just send them. Most of the time both sides will have the same end of line convention anyway. If someone needs a conversion it can be done with a TECO macro separately.

Resolution: I have required CRLF conversion in ASCII type. If you have the same kind of machines and the same end of line convention you can avoid the extra cost of conversion by using the binary or continuous type.

TCP Urgent

Pro: Use TCP Urgent to abort a transfer, instead of aborting the connection. Then one could retry the file, or try a different file without having to login again.

Con: That would couple SFTP to TCP too much. SFTP is supposed to be able to be work over any reliable 8-bit data stream.

Resolution: I have not made use of TCP Urgent.

Random Access

Pro: Wouldn't it be nice if (WIBNIF) SFTP had a way of accessing parts of a file?

Con: Forget it, this is supposed to be SIMPLE file transfer. If you need random access use real FTP (oops, real FTP doesn't have random access either -- invent another protocol?).

Resolution: I have not made any provision for Random Access.

-- jon postel.

Network Working Group
Request for Comments: 862

J. Postel
ISI
May 1983

Echo Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement an Echo Protocol are expected to adopt and implement this standard.

A very useful debugging and measurement tool is an echo service. An echo service simply sends back to the originating source any data it receives.

TCP Based Echo Service

One echo service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 7. Once a connection is established any data received is sent back. This continues until the calling user terminates the connection.

UDP Based Echo Service

Another echo service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 7. When a datagram is received, the data from it is sent back in an answering datagram.

Network Working Group
Request for Comments: 863

J. Postel
ISI
May 1983

Discard Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Discard Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a discard service. A discard service simply throws away any data it receives.

TCP Based Discard Service

One discard service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 9. Once a connection is established any data received is thrown away. No response is sent. This continues until the calling user terminates the connection.

UDP Based Discard Service

Another discard service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 9. When a datagram is received, it is thrown away. No response is sent.

Network Working Group
Request for Comments: 867

J. Postel
ISI
May 1983

Daytime Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Daytime Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a daytime service. A daytime service simply sends a the current date and time as a character string without regard to the input.

TCP Based Daytime Service

One daytime service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 13. Once a connection is established the current date and time is sent out the connection as a ascii character string (and any data received is thrown away). The service closes the connection after sending the quote.

UDP Based Daytime Service

Another daytime service service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 13. When a datagram is received, an answering datagram is sent containing the current date and time as a ASCII character string (the data in the received datagram is ignored).

Daytime Syntax

There is no specific syntax for the daytime. It is recommended that it be limited to the ASCII printing characters, space, carriage return, and line feed. The daytime should be just one line.

One popular syntax is:

Weekday, Month Day, Year Time-Zone

Example:

Tuesday, February 22, 1982 17:37:43-PST

Another popular syntax is that used in SMTP:

dd mmm yy hh:mm:ss zzz

Example:

02 FEB 82 07:59:01 PST

NOTE: For machine useful time use the Time Protocol (RFC-868).

Network Working Group
Request for Comments: 868

J. Postel - ISI
K. Harrenstien - SRI
May 1983

Time Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Time Protocol are expected to adopt and implement this standard.

This protocol provides a site-independent, machine readable date and time. The Time service sends back to the originating source the time in seconds since midnight on January first 1900.

One motivation arises from the fact that not all systems have a date/time clock, and all are subject to occasional human or machine error. The use of time-servers makes it possible to quickly confirm or correct a system's idea of the time, by making a brief poll of several independent sites on the network.

This protocol may be used either above the Transmission Control Protocol (TCP) or above the User Datagram Protocol (UDP).

When used via TCP the time service works as follows:

S: Listen on port 37 (45 octal).

U: Connect to port 37.

S: Send the time as a 32 bit binary number.

U: Receive the time.

U: Close the connection.

S: Close the connection.

The server listens for a connection on port 37. When the connection is established, the server returns a 32-bit time value and closes the connection. If the server is unable to determine the time at its site, it should either refuse the connection or close it without sending anything.

When used via UDP the time service works as follows:

S: Listen on port 37 (45 octal).

U: Send an empty datagram to port 37.

S: Receive the empty datagram.

S: Send a datagram containing the time as a 32 bit binary number.

U: Receive the time datagram.

The server listens for a datagram on port 37. When a datagram arrives, the server returns a datagram containing the 32-bit time value. If the server is unable to determine the time at its site, it should discard the arriving datagram and make no reply.

The Time

The time is the number of seconds since 00:00 (midnight) 1 January 1900 GMT, such that the time 1 is 12:00:01 am on 1 January 1900 GMT; this base will serve until the year 2036.

For example:

the time 2,208,988,800 corresponds to 00:00 1 Jan 1970 GMT,

2,398,291,200 corresponds to 00:00 1 Jan 1976 GMT,

2,524,521,600 corresponds to 00:00 1 Jan 1980 GMT,

2,629,584,000 corresponds to 00:00 1 May 1983 GMT,

and -1,297,728,000 corresponds to 00:00 17 Nov 1858 GMT.

Network Working Group
Request for Comments: 864

J. Postel
ISI
May 1983

Character Generator Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Character Generator Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a character generator service. A character generator service simply sends data without regard to the input.

TCP Based Character Generator Service

One character generator service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 19. Once a connection is established a stream of data is sent out the connection (and any data received is thrown away). This continues until the calling user terminates the connection.

It is fairly likely that users of this service will abruptly decide that they have had enough and abort the TCP connection, instead of carefully closing it. The service should be prepared for either the carfull close or the rude abort.

The data flow over the connection is limited by the normal TCP flow control mechanisms, so there is no concern about the service sending data faster than the user can process it.

UDP Based Character Generator Service

Another character generator service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 19. When a datagram is received, an answering datagram is sent containing a random number (between 0 and 512) of characters (the data in the received datagram is ignored).

There is no history or state information associated with the UDP version of this service, so there is no continuity of data from one answering datagram to another.

The service only send one datagram in response to each received datagram, so there is no concern about the service sending data faster than the user can process it.

RFC 864 May 1983
Character Generator Protocol

Data Syntax

The data may be anything. It is recommended that a recognizable pattern be used in tha data.

One popular pattern is 72 chraracter lines of the ASCII printing characters. There are 95 printing characters in the ASCII character set. Sort the characters into an ordered sequence and number the characters from 0 through 94. Think of the sequence as a ring so that character number 0 follows character number 94. On the first line (line 0) put the characters numbered 0 through 71. On the next line (line 1) put the characters numbered 1 through 72. And so on. On line N, put characters (0+N mod 95) through (71+N mod 95). End each line with carriage return and line feed.

RFC 864 May 1983
Character Generator Protocol

Example

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefgh
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghi
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghij
$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijk
%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijkl
&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklm
'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmn
()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
)*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnop
*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopq
+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqr
,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrs
-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrst
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstu
/0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuv
0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwx
23456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxy
3456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{
56789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|
6789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}
789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
89:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
9:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !
:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"
;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#
<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$
=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%
>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&
?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()
BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*
CDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+
DEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,
EFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-
FGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-.
GHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0
IJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./01
JKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./012
KLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0123
LMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./01234
MNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./012345
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0123456
```

Network Working Group
Request for Comments: 865

J. Postel
ISI
May 1983

Quote of the Day Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Quote of the Day Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a quote of the day service. A quote of the day service simply sends a short message without regard to the input.

TCP Based Character Generator Service

One quote of the day service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 17. Once a connection is established a short message is sent out the connection (and any data received is thrown away). The service closes the connection after sending the quote.

UDP Based Character Generator Service

Another quote of the day service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 17. When a datagram is received, an answering datagram is sent containing a quote (the data in the received datagram is ignored).

Quote Syntax

There is no specific syntax for the quote. It is recommended that it be limited to the ASCII printing characters, space, carriage return, and line feed. The quote may be just one or up to several lines, but it should be less than 512 characters.

Network Working Group
Request for Comments: 866

J. Postel
ISI
May 1983

Active Users

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement an Active Users Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is an active users service. An active users service simply sends a list of the currently active users on the host without regard to the input.

An active user is one logged in, such as listed in SYSTAT or WHO.

TCP Based Active Users Service

One active users service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 11. Once a connection is established a list of the currently active users is sent out the connection (and any data received is thrown away). The service closes the connection after sending the list.

UDP Based Active Users Service

Another active users service service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 11. When a datagram is received, an answering datagram is sent containing a list of the currently active users (the data in the received datagram is ignored).

If the list does not fit in one datagram then send a sequence of datagrams but don't break the information for a user (a line) across a datagram. The user side should wait for a timeout for all datagrams to arrive. Note that they are not necessarily in order.

User List Syntax

There is no specific syntax for the user list. It is recommended that it be limited to the ASCII printing characters, space, carriage return, and line feed. Each user should be listed on a separate line.

NWG/RFC# 742
Network Working Group
Request for Comments: 742
NIC: 42758

KLH 30-Dec-77 08:31 42758
K. Harrenstien
SRI-KL
30 December 1977

NAME/FINGER

Introduction

This note describes the Name/Finger protocol. This is a simple protocol which provides an interface to the Name and Finger programs at several network sites. These programs return a friendly, human-oriented status report on either the system at the moment or a particular person in depth. Currently only the SAIL (SU-AI), SRI (SRI-(KA/KL)), and ITS (MIT-(AI/ML/MC/DMS)) sites support this protocol, but there are other systems with similar programs that could easily be made servers; there is no required format and the protocol consists mostly of specifying a single "command line".

To use via the network:

ICP to socket 117 (octal, 79. decimal) and establish two 8-bit connections.

Send a single "command line", ending with <CRLF>.

Receive information which will vary depending on the above line and the particular system. The server closes its connections as soon as this output is finished.

The command line:

Systems may differ in their interpretations of this line. However, the basic scheme is straightforward: if the line is null (i.e. just a <CRLF> is sent) then the server should return a "default" report which lists all people using the system at that moment. If on the other hand a user name is specified (e.g. FOO<CRLF>) then the response should concern only that particular user, whether logged in or not.

Both ITS and SAIL sites allow several names to be included on the line, separated by commas; but the syntax for some servers can be slightly more elaborate. For example, if "/W" (called the "Whois switch") also appears on the line given to an ITS server, much fuller descriptions are returned. The complete documentation may be found at any time in the files ".INFO.;NAME ORDER" on MIT-AI, "FINGER.LES[UP,DOC]" on SU-AI, and "<DOCUMENTATION>FINGER.DOC" on

NWG/RFC# 742 KLH 30-Dec-77 08:31 42758
Name/Finger

SRI-KL, all freely accessible by FTP (with the exception of SRI-KL, where TOPS-20 requires the "anonymous" login convention).

Allowable "names" in the command line should of course include "user names" or "login names" as defined by the system, but it is also reasonable to understand last names or even full names as well. If a name is ambiguous, all possible derivations should be returned in some fashion; SAIL will simply list the possible names and no more, whereas an ITS server will furnish the full standard information for each possibility.

Response to null command line - "default" listing:

This is a request for a list of all online users, much like a TOPS-10 or TENEX "systat". To fulfill the basic intent of the Name/Finger programs, the returned list should include at least the full names of each user and the physical locations of their terminals insofar as they can be determined. Including the job name and idle time (number of minutes since last typein, or since last job activity) is also reasonable and useful. The appendix has examples which demonstrate how this information can be formatted.

Response to non-null command line - "name" listing:

For in-depth status of a specified user, there are two main cases. If the user is logged in, a line or two is returned in the same format as that for the "default" listing, but showing only that user. If not logged in, things become more interesting. Furnishing the full name and time of last logout is the expected thing to do, but there is also a "plan" feature, wherein a user may leave a short message that will be included in the response to such requests. This is easily implemented by (for example) having the program look for a specially named text file on the user's directory or some common area. See the examples for typical "plans".

Implementation miscellany:

Anyone wishing to implement such a server is encouraged to get in touch with the maintainers of NAME by sending a message to BUG-NAME @ MIT-AI; apart from offering advice and help, a list of all sites with such servers is kept there. It is also suggested that any existing programs performing similar functions locally (i.e. not as net servers) be extended to allow specification of other sites, or names at other sites. For example, on ITS systems one can say ":NAME<cr>" for a local default listing, or ":NAME @SAIL<cr>" for SAIL's default listing, or ":NAME Foo@MC<cr>" to ask MIT-MC about Foo's status, etc.

NWG/RFC# 742 KLH 30-Dec-77 08:31 42758
Name/Finger

It should be noted that connecting directly to the server from a TIP or an equally narrow-minded TELNET-protocol user program can result in meaningless attempts at option negotiation being sent to the server, which will foul up the command line interpretation unless the server knows enough to filter out IAC's and perhaps even respond negatively (IAC WON'T) to all option commands received. This is a convenience but is not at all required, since normally the user side is just an extended NAME/FINGER type program.

And finally a little background:

The FINGER program at SAIL, written by Les Earnest, was the inspiration for the NAME program on ITS. Earl Killian at MIT and Brian Harvey at SAIL were jointly responsible for implementing the protocol just described, and Greg Hinchliffe has recently brought up a similar server for SRI-KA and SRI-KL.

NWG/RFC# 742 KLH 30-Dec-77 08:31 42758
Appendix - Examples

EXAMPLES

Note: it is possible for some lines of the actual output to exceed 80 chars in length. The handling of such lines is of course dependant on the particular user program; in these examples, lines have been truncated to 72 chars for greater clarity.

Three examples with a null command line:

Site: MIT-AI
Command line:

```
-User-    --Full name--          Jobnam Idle TTY -Console location-
XGP     O Xerox Graphics Printer XGPSPL      T24 Datapoint Near XGP (9TH)
FFM     U Steven J. Kudlak       HACTRN      T41 Net site CMU-10A
KLH     + Ken Harrenstien        F           T42 Net site SRI-KL
___013  -  Not Logged In         HACTRN 1:26.T43 DSSR UNIX x3-6048 (MIT-*
CWH     U Carl W. Hoffman        E         4.T50 919 Very Small Data Bas*
CARL    A Carl Hewitt            HACTRN 5:03.T52 813 Hewitt x5873
APD     M Alexander Doohovskoy   XGP    1:52.T54 912 9th Floor Lounge x6*
JJK     T James Koschella        E           T55 824 Hollerbach, Levin, *
KEN     L Kenneth Kahn           E           T56 925 Moon (Tycho under) *
```

Site: SAIL
Command line:

```
     Person           Job Jobnam Idle    Terminal
DAN Dan Sleator       46 MACLSP      DM-3          150/1200 modem 415 49*
DEK Don Knuth          3 E        3. tv-55  205    Library
                      20 PI       2  TV-55  205    Library
ES  Gene Salamin      44 SD MC       TV-40  223a   Farmwald
JJ  Jerrold Ginsparg  11 TELNET     DM-0          150/1200 modem 415 49*
JMC John McCarthy      1 FINGER   . detached
                      12 E        2. IML-15        McCarthy's house
RD  Randy Davis       42 AID      7  TV-52  203    Allen
LES Les Earnest       23 TEMPS    2. DM-1          150/1200 modem 415 49*
ME  Martin Frost      17 E        3  tv-46  220    Filman, Frost
                      31 E           TV-46  220    Filman, Frost
PAM Paul Martin        9 E           TV-106 251C   King, Levy, Martin
ROD Rod Brooks        37 MACLSP   3  TV-117 250C
RWG Bill Gosper       30 SD MC       TV-34  230e   Robinson
                                     TV-67  213    Kant, McCune, Steinbe*
RWW Richard Weyhrauch 39 E           TV-42  214    Weyhrauch
SYS system files       6 FINGER      PTY122        job 5 Arpanet site AI*
```

NWG/RFC# 742 KLH 30-Dec-77 08:31 42758
Appendix - Examples

Site: SRI-KL
Command line:

Thursday, 15-Dec-77 01:21:24-PST System up 3 Days, 22:20:52 28 Jobs
Drum 0% Load avs 0.26 0.23 0.31 14 Act, 10 Idle, 8 Det

User	Personal Name	Job	Subsys	15m%	TTY	Room	Console Location
BLEAN	Bob Blean	37	EXEC	0.0	41	K2007	Blean
KLH	Ken Harrenstien	83	TELNET	1.6	12	J2023	Spaceport
KREMERS	Jan Kremers	48	TECO	0.0	121	Dialup	326-7005 (300 Ba*
MAINT	Digital Equipment	54	SNDMSG	0.5	43	K2035	Melling
MCCLURG	Jim McClurg	40	EXEC	0.0	26	PKT	
MMCM	Michael McMahon	31	EXEC	1.5	122	Dialup	326-7006 (300 Ba*
MOORE	J Moore	52	TV	0.2	124	Dialup	326-7008 (300 Ba*
PATTIS	Richard Pattis	19	LISP	0.8	11	ARC	
PETERSO	Norman Peterson	33	EXEC	25:12	234		(RAND-TIP)
STONE	Duane Stone	34	TELNET	3:51	240		(RADC-TIP)
		27	EXEC	7:11	232		(SRI-KL)
TORRES	Israel Torres	64	BSYS	0.0	76	K2079	TI by tape drives
		68	EXEC	1:15	104	K2029	Operators' Office

NWG/RFC# 742 KLH 30-Dec-77 08:31 42758
Appendix - Examples

Examples with names specified:

```
Site: MIT-AI
Command line: klh

KLH     + Ken Harrenstien          Last logout 10/16/77 13:02:11  No plan.


Site: MIT-MC
Command line: cbf

CBF     M Charles Frankston        Not logged in.  Plan:
  I'll be visiting another planet til about December 15.  If anyone
  wants to get a hold of me transmit on some fundamental wavelength
  (like the radius of the hydrogen atom).


Site: MIT-MC
Command line: smith

BRIAN   A Brian C. Smith           Last logout 11/24/77 08:02:24  No plan.
DBS     T David B. Smith           Last logout 12/03/77 11:24:01  No plan.
BPS     T Byron Paul Smith         Not logged in.  No plan.
GRS     U Gary R. Smith            Last logout 12/12/77 18:43:19  No plan.
JOS     S Julius Orion III Smith   Last logout 11/29/77 06:18:18  No plan.
$PETE   M PETER G. SMITH,          Not logged in.  No plan.
IAN     L Ian C. Smith             Not logged in.  No plan.
AJS     D Arnold J. Smith          Last logout 12/09/77 14:31:11  No plan.


Site: SU-AI
Command line: smith

"SMITH" is ambiguous:
RS  Bob Smith
DAV Dave Smith
JOS Julius Smith
LCS Leland Smith
```

NWG/RFC# 742 KLH 30-Dec-77 08:31 42758
Appendix - Examples

Site: SU-AI
Command line: jbr

```
     Person          Job Jobnam Idle Line  Room   Location
JBR Jeff Rubin        16 COPY    27. TV-43  222    Rubin
                                     TV-104 233    hand-eye table
```

Site: SU-AI
Command line: bh

```
     Person              Last logout
BH  Brian Harvey      22:49 on 14 Dec 1977.  Plan:
    ^008-Oct-77  2156   BH  ^Y12257 (1-Jul-78)
Weekdays during the day I'm usually unreachable; I'm either at S.F.
State or at Benjamin Franklin JHS in San Francisco, but neither place
is recommended for leaving messages.  Evenings and weekends I'm
generally home (55) 751-1762 unless I'm at SAIL.  I log in daily from
home.
```

Site: SRI-KL
Command line: greg

GREG (Greg Hinchliffe) is on the system:

```
Job Subsys  # Siz  Runtime     1m%  15m%  TTY  Room   Console Location
62  EXEC    1   0  0:00:10.6         0.8  235         (SUMEX-AIM)

Last login: Mon 12-Dec-77, 15:05, from SUMEX-AIM (Host #56.)
GREG has no new mail, last read on Mon 12-Dec-77 15:10
```

Network Working Group
Request for Comments: 954
Obsoletes: RFC 812

K. Harrenstien (SRI)
M. Stahl (SRI)
E. Feinler (SRI)
October 1985

NICNAME/WHOIS

STATUS OF THIS MEMO

This RFC is the official specification of the NICNAME/WHOIS protocol. This memo describes the protocol and the service. This is an update of RFC 812. Distribution of this memo is unlimited.

INTRODUCTION

The NICNAME/WHOIS Server is a TCP transaction based query/response server, running on the SRI-NIC machine (26.0.0.73 or 10.0.0.51), that provides netwide directory service to internet users. It is one of a series of internet name services maintained by the DDN Network Information Center (NIC) at SRI International on behalf of the Defense Communications Agency (DCA). The server is accessible across the Internet from user programs running on local hosts, and it delivers the full name, U.S. mailing address, telephone number, and network mailbox for DDN users who are registered in the NIC database.

This server, together with the corresponding WHOIS Database can also deliver online look-up of individuals or their online mailboxes, network organizations, DDN nodes and associated hosts, and TAC telephone numbers. The service is designed to be user-friendly and the information is delivered in human-readable format. DCA strongly encourages network hosts to provide their users with access to this network service.

WHO SHOULD BE IN THE DATABASE

DCA requests that each individual with a directory on an ARPANET or MILNET host, who is capable of passing traffic across the DoD Internet, be registered in the NIC WHOIS Database. MILNET TAC users must be registered in the database. To register, send via electronic mail to REGISTRAR@SRI-NIC.ARPA your full name, middle initial, U.S. mailing address (including mail stop and full explanation of abbreviations and acronyms), ZIP code, telephone (including Autovon and FTS, if available), and one network mailbox. Contact the DDN Network Information Center, REGISTRAR@SRI-NIC.ARPA or (800) 235-3155, for assistance with registration.

RFC 954 October 1985
NICNAME/WHOIS

PROTOCOL

To access the NICNAME/WHOIS server:

Connect to the SRI-NIC service host at TCP service port 43 (decimal).

Send a single "command line", ending with <CRLF> (ASCII CR and LF).

Receive information in response to the command line. The server closes its connection as soon as the output is finished.

EXISTING USER PROGRAMS

NICNAME is the global name for the user program, although many sites have chosen to use the more familiar name of "WHOIS". There are versions of the NICNAME user program for TENEX, TOPS-20, and UNIX. The TENEX and TOPS-20 programs are written in assembly language (FAIL/MACRO), and the UNIX version is written in C. They are easy to invoke, taking one argument which is passed directly to the NICNAME server at SRI-NIC. Contact NIC@SRI-NIC.ARPA for copies of the program.

COMMAND LINES AND REPLIES

A command line is normally a single name specification. Note that the specification formats will evolve with time; the best way to obtain the most recent documentation on name specifications is to give the server a command line consisting of "?<CRLF>" (that is, a question-mark alone as the name specification). The response from the NICNAME server will list all possible formats that can be used. The responses are not currently intended to be machine-readable; the information is meant to be passed back directly to a human user. The following three examples illustrate the use of NICNAME as of October 1985.

Command line: ?
Response:

Please enter a name or a NIC handle, such as "Smith" or "SRI-NIC". Starting with a period forces a name-only search; starting with exclamation point forces handle-only. Examples:

```
   Smith      [looks for name or handle SMITH]
   !SRI-NIC   [looks for handle SRI-NIC only]
   .Smith, John
              [looks for name JOHN SMITH only]
```

Adding "..." to the argument will match anything from that point, e.g. "ZU..." will match ZUL, ZUM, etc.

To search for mailboxes, use one of these forms:

```
   Smith@     [looks for mailboxes with username SMITH]
   @Host      [looks for mailboxes on HOST]
   Smith@Host
              [Looks for mailboxes with username SMITH on HOST]
```

To obtain the entire membership list of a group or organization, or a list of all authorized users of a host, precede the name of the host or organization by an asterisk, i.e. *SRI-NIC. [CAUTION: If there are a lot of members, this will take a long time!] You may use exclamation point and asterisk, or a period and asterisk together.

Command line: fischer
Response:

```
Fischer, Charles (CF17)      fischer@UWISC          (608) 262-1204
Fischer, Herman (HF)         HFischer@USC-ECLB      (818) 902-5139
Fischer, Jeffery H. (JHF1)   FISCHER@LL-XN          (617) 863-5500
                                                    ext 4403 or 4689
Fischer, Kenneth (KF8)       SAC.SIUBO@USC-ISIE     (402) 294-5161
                                                    (AV)  271-5161
Fischer, Marty (MF28)        MFISCHER@DCA-EMS       (703) 437-2344
Fischer, Michael J. (MJF)    FISCHER@YALE           (203) 436-0744
Fischer, Nancy C. (NANCY)    FISCHER@SRI-NIC        (415) 859-2539
Fischer, Richard A. (RAF4)   Fisher Richa@LLL-MFE   (415) 422-5032
```

To single out any individual entry, repeat the command using the argument "!HANDLE" instead of "NAME", where the handle is in parentheses following the name.

Command line: !nancy
Response:

Fischer, Nancy C. (NANCY) FISCHER@SRI-NIC SRI International
Telecommunication Sciences Center
333 Ravenswood Avenue, EJ289
Menlo Park, California 94025
Phone: (415) 859-2539
MILNET TAC user

BIBLIOGRAPHY

1. Harrenstien, K., and White, V., "NICNAME/WHOIS," RFC-812, Network Information Center, SRI International, March 1982.

2. Harrenstien, K., "NAME/FINGER," RFC-742, Network Information Center, SRI International, December 1977.

Network Working Group
RFC 569/ NET-USING Memo 1
NIC # 18972

Mike Padlipsky
NET-USING
October 15, 1973

NETED: A Common Editor for the ARPA Network

BACKGROUND

At the recent Resource Sharing Workshop, there was a somewhat surprising degree of consensus on what I had anticipated would the least popular aspect of the my "Unified User-Level Protocol" proposal: A number of the attendees agreed without argument that it would be a good thing to have "the same" context editor available on all Servers -- where "the same" refers, of course, to the user interface. We even agreed that "NETED" seemed to be a plausible common name. In view of the fact that the rest of the proposal didn't seem to capture anybody's imagination, though, it seemed to be a useful notion to separate out the common editor and make it the subject of a stand-alone proposal.

My resolve to come up with the following was further strengthened at the the organizing meeting of the Network User Interest Group, which followed the Workshop. Being primarily concerned with user issues, this group was downright enthusiastic about the prospect of a common editor. Indeed, this proposal has been reviewed by the group and is endorsed by it.

REASONS

The need for a common editor might well be obvious to many readers. They are encouraged to skip this section, which is for the benefit of those who don't already see the light.

In the first place, it's almost axiomatic that to use a time-sharing system, you have to be able to create files (/"datasets"/"segments"). Even if you're only using the Network to send "mail", you'd still like to be able to create a file separately, so as to be able to edit it before sending. And if you want to write a program -- or even make a "runoff" source file -- you simply must be able to use some editor command on the system at hand.

Unfortunately, there are even more editors than there are systems; and each one has it own conventions and peculiarities. So "Network users" (who use several Servers, as opposed to those who use the Network only to access a particular system all the time) are faced with the unpleasant chore of developing several sets of incompatible reflexes if they want to get along. This can certainly be done. It has been by a number of members of the Network Working Group.

The real kicker, however, comes when we consider the needs of those users -- who are coming into the Network community in ever-increasing numbers -- who are not professional programmers. They just want to get some work done, "on the Net" (that is, irrespective of which operating system they might be talking to). They are likely to be appalled rather than amused by having to learn a dozen ways of getting to first base. Therefore, it seems clear than not only do we need a common editor, but we also need a simple common editor.

CHOICES

Simplicity is not the only criterion for rejecting the apparently "obvious" choice of either TECCO or QED. (That it is a strong factor is indicated by the old test of "Consider explaining it to a naive secretary -- now consider explaining it to a corporation president.") Perhaps even worse is the problem of "dialects". That is, features vary across implementations, and settling on a common set of features (or dialect) is likely to be a very hard task, for programmers tend to get very fond of their familiar goodies. Besides, both TECO and QED have their own strong (/fanatic) advocates, who's probably never be willing to settle for the other one. Further, not every system has both, and implementing the other is a fairly large job even if the NWG could agree on which (and how much).

At any rate, the difficulties seem overwhelming when it comes to choosing a high-powered editor as the common editor. Therefore, I tried to think of a nice low-powered editor, and it suddenly occurred to me that I not only knew of one, but it was even fairly well documented (!). The editor in question is known on Multics as "eds" (the same member of the "ed" family of editors which started on CTSS), a line-oriented context editor (no "regular expressions", but also no line numbers). It is used as an extended example of programming in the Multics environment in Chapter 4 of the Multics Programmers' Manual, which gives an annotated PL/I listing of the actual working program. It is simple to learn and should be quite easy to implement, PL/I version serves as a detailed model with only equivalent system calls and choice of language to worry about. I urge its adoption as the common Network editor, to be known on all participating Servers as "NETED" and/or "neted".

DOCUMENTATION

In view of the fact that if "eds"/NETED is adopted only perhaps a dozen members of the NWG will actually have to implement one, it seems wasteful to distributed some 30 pages of the MPM to everyone -- especially since most of the parties concerned have access to an MPM already. (Another problem solved by not including it here is that of whether I'd be violating copyright by doing so.) The exact reference is pp. 24-54 of Chapter 4 of Part I of the Multics Programmer's Manual.

Anybody who needs a copy can let me know. Although the emphasis in the document is, naturally enough, on the Multics-specific aspects, I believe that the listing is clear enough to serve as a model to implementors without any great difficulty. If we do get to the implementation stage, I'll be glad to try to explain any non-obvious system calls, either individually or in a follow-up memo. But even though we "initiate" where you "open", or we " call los_$read_ptr" where you "IOT TTY" (or something), it shouldn't cause much trouble. For that matter, some implementers might prefer to ignore the existing program and simply work from the function specifications (below).

LIMITATIONS

It became abundantly clear during the course of the review of this document by the User Interest Group that the limitations of NETED must be acknowledged (even insisted upon) and explained here. In the first place, it must be emphasized that it is not being proposed as "THE" Network editor. Rather, it is an insistently simple-minded editor for two major reasons: 1) it is meant for use mainly by non-professional programmers, and 2) more important still, it is meant to be extremely easy to implement. Therefore, it seems far more important to go with the published program, with all its warts, than to specify the addition of new, undebugged features. The idea is to make it implementable in man-days by an average to average-plus programmer instead of in man-weeks by a superstar programmer.

In the second place, the very act of adding new features is fraught with peril. To take some examples from the comments I received during the review phase: In the first draft, I inadvertently failed to document the mechanism by which the ability to "go backwards" (i.e., to reverse the direction of the current-line pointer described below) is actuated. Several reviewers argued strongly for the inclusion of such a mechanism; but, not knowing it was already "in" the code I argued -- successfully -- for leaving it out, on the grounds that we should stick to what's in the existing code, which is known to work as published. Even what to call such a new request would have been debatable -- should it be "-" and become the only non-alphabetic name? should it be "b" for "bottom"? should "n" (for "next") become "+"? And so on. Although this particular issue turned out be a false alarm, I've left it in to emphasize the sort of pitfalls we can get into by haggling over particular "features". Another familiar feature is some sort of "read" request so that the file name need not be specified as an argument to the command. Then, of course, one would also want a "create" or "make" request. And perhaps a file delete request? It keeps going on and on. The point, I think, is that if we allow ourselves to go into "tinker mode" we could spend as many years striving to achieve consensus on what features to add as we've spent on Telnet or FTP ... and still not please everyone. Therefore, I urge the NWG to accept the contention that having a working model to use as

a pattern is more important than any particular additional features (even though I myself find "=" for "what's the current line's number?" annoying to live without).

RESPONSES

For the benefit of those who don't want to plow through the functional spec, this seems to be a good spot to indicate what appropriate responses to this proposal would be. Ideally, I'd like to hear from a responsible system programmer at, say, TENEX, CCN, UCSD, UCSB, AMES-67, one of the DEC 10/50 Hosts, and from any of the experimental Servers who happen to be interested, that they think it's a fine idea and why don't I log in next week to try their NETEDs. Next most desirable would be agreement in principle followed by specific inquiries about "eds". I would hope that haggling over specific features wouldn't occur (as we're not trying to do a definitive editor, just an easy, commonly implemented one based on an existing implementation), but unfortunately I can't legislate such haggles out of existence. At the very least, I'd hope to either hear or see reasoned arguments why it's not worth doing. As usual, phone, mail "mail" ("map.cn" in sndmsg, or "map cn" in "mail" on Multics) or RFC's are the assumed media for responding.

USAGE

(The following is intended to serve double-duty, as both a functional spec now and -- with the addition of some examples -- a "users' manual" later. So if it seems to "tutorial", I'm sorry. And if it doesn't seem tutorial enough -- assuming the addition of examples -- please let me know.)

As is typical of "context editors," the NETED command is used both for creating new files and for altering already existing files -- where "files" are named collections of character encoded data in the storage hierarchy of a time-sharing system. Consequently, NETED operates in two distinct "modes" -- called "input mode" and "edit mode".

When NETED is used to create a file (that is, when it is invoked from command level with an argument which specifies the name of a file which does not already exist in the user's "working directory"), it is automatically in input mode. It will announce this fact by outputting a message along the lines of "File soandso not found. Input." Until you take explicit action to leave input mode, everything you type will go into the specified file. (Actually, it goes into a "working copy" of the file, and into the real file only when you indicate a desire to have that happen.) To leave input mode, type a line consisting of only a period and the appropriate new-line character: ".<NL>", where <NL> is whatever it takes to cause a Telnet New-Line to be generated from your terminal

After leaving input mode, you are in edit mode. Here, you may issue various "requests" which will allow you to alter the contents of the (working) file, re-enter input mode if you wish, and eventually cause the file to be stored. Note that edit mode is entered automatically if the argument you supplied to NETED specified an existing file. Regardless of how it was entered, being in edit mode is confirmed by NETED's outputting a message of the form "Edit". Editing is performed relative to (conceptual) pointer which specifies the current line, and many requests pertain to either moving the pointer or changing the contents of the current line. (When edit mode is entered from input mode, the pointer is at the last line input; when entered from command level, the pointer is at the "top" of the file.)

NETED's edit mode requests follow, in order intended to be helpful. Two important reminders: the requests may only be issued from edit mode, and each one "is a line" (i.e., terminates in a new line / carriage return / linefeed is appropriate to the User Telnet being employed). SYNTAX NOTE: If the request takes an argument, there must be at least one space (blank) between request's name and the argument.

1. n m

For unsigned m, the n(ext) request causes the pointer to be moved "down" m lines. If m is negative, the pointer is moved "up" m lines. If m is not specified, the pointer is moved one line. If the end of the file is reached, an "End of file reached by n m" message is output by NETED; the pointer is left "after" the last line.

2. l string

The l(ocate) request causes the pointer to be moved to the net line containing the character string "string" (which may contain blanks); the line is output. If no match is found, a message of the form "End of file reached by l string" will be output (and the pointer will have returned to the top of the file). The search will not wrap around the end of the file; however, if the string was above the starting position of the pointer, a repetition of the locate request will find it, in view of the fact that the pointer would have been moved to the top of the file. To find any occurrence of the string -- rather than the next occurrence -- it is necessary to move the pointer to the top of the file before doing the locate (see following request).

3. t

Move the pointer to the top of the file.

4. b

Move the pointer to the bottom of the file and enter input mode.

5. "."

Leave the pointer where it is and enter input mode. (First new line goes after current old line.)

6. i string

The i(nsert) request cause a line consisting of string (which will probably contain blanks) to be inserted after the current line. The pointer is moved to the new line. Edit mode is not left.

7. r string

The r(eplace) request causes a line consisting of string (probably containing blanks) to replace the current line.

8. p m

The p(rint) request causes the current line and the succeeding m - 1 lines to be output. If m is not specified, only the current line will be output. End of file considerations are the same as with "n".

9. c /s1/s2/ m g

The c(hange) request is quite powerful, although perhaps a bit complex to new users. In the line being pointed at, the string of characters s1 is replaced by the string of characters s2. If s1 is void, s2 will be inserted at the beginning of the line; if s2 is void, s1 will be deleted from the line. Any character not appearing within either character string may be used in place of the slash (/) as a delimiter. If a number, m, is present, the request will affect m lines, starting with the one being pointed at. All lines in which a change was made are printed. The pointer is left at the last line scanned. If the letter "g" is absent (after the final delimiter) only the first occurrence of s1 within a line will be changed. If "g" (for "global") is present, all occurrences of s1 within a line will be changed. (If s1 is void, "g" has no effect.) NOTE WELL: blanks in both strings are significant and must be counted exactly. End of file considerations are the same as with "n".

10 d m

The d(elete) request causes m lines, including the current one, to be deleted from the working copy of the file. If m is not specified, only the current line is deleted. The pointer is left at a null line above the first undeleted line. End of file considerations are the same as with "n".

11. w

Write out the working copy into the storage hierarchy but remain in NETED. (Useful for those who fear crashes and don't want to lose all the work performed.)

12. save

Write out the working copy into the storage hierarchy and exit from NETED.

Additional specs:

a. On Multics, type-ahead is permitted. This approach is recommended for all versions of NETED, but is of course not required as various Servers' NCP Implementations may prohibit it; however:

b. If an error is detected, the offending line is output, and pending typeahead (if any) must be discarded (to guard against the possibility of the pending request's being predicated on the success of erroneous request).

c. The command is not reinvokable, in the sense that work is lost if you "quit" out of it via the Telnet Interrupt Process command or its equivalent; indeed, quitting out is the general method of negating large amounts of incorrect work and retaining the original file intact.

(When the time comes, I'll be glad to furnish examples for the users' manual version; but for now, that's all there is.)

NOTE

It really does work, unsophisticated though it may be. I think that it's sufficient to get new users going, and necessary to give them a fighting chance. It would even be of utility within the NWG, for those of us who don't like having to learn new editors all the time. If anybody wants to try it, I'll make a version available to "anonymous users" (see the Multics section in the Resource Notebook if you don't already know how to get in our sampling account), under the name "neted". (*) (If you do try it, please delete files when done with them.)

(*) Knowledgeable Multics users with their own accounts can instead link to >udd>cn>map>neted. It is also there under the names "eds" if you want to save typing a couple of characters.

Network Working Group
Request for Comments: 887

M. Accetta
Carnegie-Mellon University
December 1983

RESOURCE LOCATION PROTOCOL

This note describes a resource location protocol for use in the ARPA Internet. It is most useful on networks employing technologies which support some method of broadcast addressing, however it may also be used on other types of networks. For maximum benefit, all hosts which provide significant resources or services to other hosts on the Internet should implement this protocol. Hosts failing to implement the Resource Location Protocol risk being ignored by other hosts which are attempting to locate resources on the Internet. This RFC specifies a draft standard for the ARPA Internet community.

The Resource Location Protocol (RLP) utilizes the User Datagram Protocol (UDP) [1] which in turn calls on the Internet Protocol (IP) [3] to deliver its datagrams. See Appendix A and [6] for the appropriate port and protocol number assignments.

Unless otherwise indicated, all numeric quantities in this document are decimal numbers.

1. Introduction

From time to time, Internet hosts are faced with the problem of determining where on the Internet some particular network service or resource is being provided. For example, this situation will arise when a host needs to send a packet destined for some external network to a gateway on its directly connected network and does not know of any gateways. In another case, a host may need to translate a domain name to an Internet address and not know of any name servers which it can ask to perform the translation. In these situations a host may use the Resource Location Protocol to determine this information.

In almost all cases the resource location problem is simply a matter of finding the IP address of some one (usually any) host, either on the directly connected network or elsewhere on the Internet, which understands a given protocol. Most frequently, the querying host itself understands the protocol in question. Typically (as in the case of locating a name server), the querying host subsequently intends to employ that protocol to communicate with the located host once its address is known (e.g. to request name to address translations). Less frequently, the querying host itself does not necessarily understand the protocol in question. Instead (as in the case of locating a gateway), it is simply attempting to find some other host which does (e.g. to determine an appropriate place to forward a packet which cannot be delivered locally).

2. Resource Naming

Although the resource location problem can, in most cases, be reduced to the problem of finding a host which implements a given Internet based protocol, locating only a particular lowest level Internet protocol (i.e. one assigned a protocol number for transport using IP) is not completely sufficient. Many significant network services and resources are provided through higher level protocols which merely utilize the various lower level protocols for their own transport purposes (e.g. the FTP protocol [2] employs the TCP protocol [4] for its lower level transport). Conceptually, this protocol nesting may even be carried out to arbitrary levels.

Consequently, the Resource Location Protocol names a resource by the protocol number assigned to its lowest level Internet transport protocol and by a variable length protocol/resource specific identifier. For example, the UDP based Echo Protocol can be named by its assigned protocol number (17) and its assigned 16-bit "well-known" port number (7). Alternatively, the Internet Control Message Protocol [5] (lacking any higher level client protocols) would be named simply by its assigned protocol number (1) and an empty protocol specific identifier. On the other hand, some as yet undefined resource protocol (provided via say TCP), might be named by the assigned protocol number (6), its 16-bit "well-known" TCP port number, and then some additional fixed or variable length identifier specific to that TCP port.

In general, the components of the protocol/resource specific identifier are defined to be the "natural" quantities used to successively de-multiplex the protocol at each next highest protocol level. See section 5 for some sample assignments.

3. Protocol Summary

The Resource Location Protocol is a simple request/reply procedure. The querying host constructs a list of resources which it would like to locate and sends a request message on the network. A request message may be sent either to a particular IP address and host or, on networks which provide broadcast address capability, to the IP address which refers to all hosts on that network (see [7]). For example, a host attempting to locate a domain name server might construct a request containing the resource name [17, 53] (referring to the Domain Name Server protocol provided at "well-known" UDP port 53) and then broadcast that request on its local network.

Each receiving host examines the list of resources named in the request packet, determines which of the resources it provides, and returns a reply message to the querying host in confirmation. The receiving host determines whether or not it provides a resource by successive decomposition of the resource name until either the name is exhausted or it encounters a component which is not supported. In the previous

example, each host on the network receiving the broadcast request would examine the resource name by first consulting its tables to determine if it provided UDP service. If this was successful, it would then examine the UDP port component of the name and consult its UDP table to determine if it provided service on UDP port 53. At this point the name would be exhausted and if both checks were successful the host would return a reply message to the querying host indicating support for that resource.

3.1. Request Messages

RLP provides two basic types of request messages which may be transmitted by a querying host. The first type requires any host receiving the request message to return a reply message only if it provides at least one of the resources named in the request list. The second type requires any host receiving the message to always return a reply message even if it provides none of the resources named in the request list.

These two types of request messages are:

<Who-Provides?>
This type requires any host receiving the message to return an appropriate reply message which names all of the resources from the request list which it provides. If the receiving host provides none of the named resources, no reply may be returned.

<Do-You-Provide?>
This type is identical to the <Who-Provides?> message but with the extra requirement that a reply must always be returned. When a receiving host provides none of the requested resources, it simply returns an empty reply list. This empty reply list allows the querying host to immediately detect that the confirming host provides none of the named resources without having to timeout after repeatedly retransmitting the request.

The <Who-Provides?> request message is most typically used when broadcasting requests to an entire IP network. The <Do-You-Provide?> request message, on the other hand, is most typically used when confirming that a particular host does or does not provide one or more specific resources. It may not be broadcast (since such a request would flood the querying host with reply messages from all hosts on the network).

In addition to the two basic types of request messages, an additional two variant types of request messages may also be transmitted by a querying host. These message types provide a "third-party" resource location capability. They differ from the basic message types by

providing space for an additional qualifier with each listed resource to identify "third-party" hosts which the confirming host believes may provide the resource. As before, the first type requires any host receiving the request message to return a reply message only if it knows of some host which provides at least one of the resources named in the request list. The second type requires any host receiving the message to always return a reply message even if it knows of no hosts which provide any of the resources named in the request list.

These two remaining types of request messages are:

<Who-Anywhere-Provides?>
This message parallels the <Who-Provides?> message with the "third-party" variant described above. The confirming host is required to return at least its own IP address (if it provides the named resource) as well as the IP addresses of any other hosts it believes may provide the named resource. The confirming host though, may never return an IP address for a resource which is the same as an IP address listed with the resource name in the request message. In this case it must treat the resource as if it was unsupported at that IP address and omit it from any reply list.

<Does-Anyone-Provide?>
This message parallels the <Do-You-Provide?> message again with the "third-party" variant described above. As before, the confirming host is required to return its own IP address as well as the IP addresses of any other hosts it believes may provide the named resource and is prohibited from returning the same IP address in the reply resource specifier as was listed in the request resource specifier. As in the <Do-You-Provide?> case and for the same reason, this message also may not be broadcast.

These variant request messages permit "smart" hosts to supply resource location information for networks without broadcast capability (provided that all hosts on the network always "know" the address of one or more such "smart" hosts). They also permit resource location information for services which are not provided anywhere on a directly connected network to be provided by "smart" gateways which have perhaps queried other networks to which they are attached or have somehow otherwise acquired the information.

The restriction against returning the same IP address in a reply as was specified in the request provides a primitive mechanism for excluding certain known addresses from consideration in a reply (see section 5, example 3). It may also be used to override the receiving host's preference for its own IP address in "third-party" replies when this is required.

3.2. Reply Messages

Each of the types of request messages has an associated type of reply message. The basic reply message type is returned in response to both <Who-Provides?> and <Do-You-Provide?> request messages and supplies information about the resources provided by the confirming host. The other reply message type is the "third-party" variant returned in response to both <Who-Anywhere-Provides?> and <Does-Anyone-Provide?> request messages and supplies information about resources provided by hosts known to the confirming host.

These two types of reply messages are:

<I-Provide>
This reply message contains a list of exactly those resources from the request list which the confirming host provides. These resources must occur in the reply list in precisely the same order as they were listed in the request message.

<They-Provide>
This reply message similarly contains a list of exactly those resources from the request list (appropriately qualified with IP addresses) which the confirming host provides or believes another host provides. These resources again must occur in the reply list in precisely the same order as they were listed in the request message.

Neither type of reply message may be broadcast.

A querying host which receives a <They-Provide> reply message from a confirming host on behalf of a third host is not required to unquestioningly rely on the indirectly provided information. This information should usually be regarded simply as a hint. In most cases, the querying host should transmit a specific <Do-You-Provide?> request to the third host and confirm that the resource is indeed provided at that IP address before proceeding.

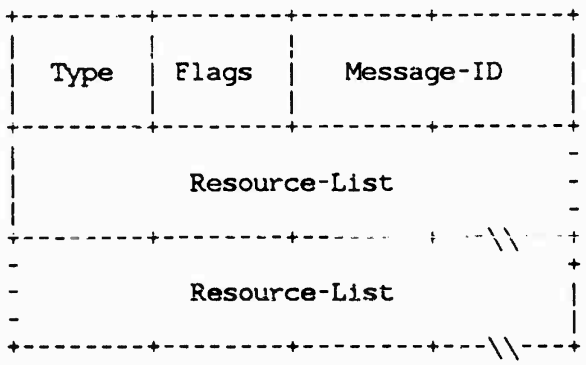
4. Message Formats

RLP messages are encapsulated in UDP packets to take advantage of the multiplexing capability provided by the UDP source and destination ports and the extra reliability provided by the UDP checksum. Request messages are sent from a convenient source port on the querying host to the assigned RLP destination port of a receiving host. Reply messages are returned from the assigned RLP source port of the confirming host to the appropriate destination port of the querying host as determined by the source port in the request message.

The format of the various RLP messages is described in the following diagrams. All numeric quantities which occupy more than one octet are

stored in the messages from the high order octet to the low order octet as per the usual Internet protocol standard. All packet diagrams indicate the octets of the message from left to right and then top to bottom as they occur in the data portion of the encapsulating UDP packet.

Each RLP packet has the general format



where

<Type>
is a single octet which identifies the message type. The currently defined types are:

0 <Who-Provides?>
1 <Do-You-Provide?>
2 <Who-Anywhere-Provides?>
3 <Does-Anyone-Provide?>
4 <I-Provide>
5 <They-Provide>
6-255 Reserved.

<Flags>
is a single octet specifying possible modifications to the standard interpretation of <Type>. Bits in this field are numbered from left to right (from most significant to least significant) beginning with bit 1. The currently defined flag bits are:

Bit 1 <Local-Only>. Requires that any reply message generated in response to a request message with this flag bit set may only name IP addresses which are on the same IP network as the sender of the request message. This flag also requires that multi-homed hosts answering broadcast <Who-Provides?> requests use the appropriate local network IP source address in the returned reply. This bit must be zero in reply messages.

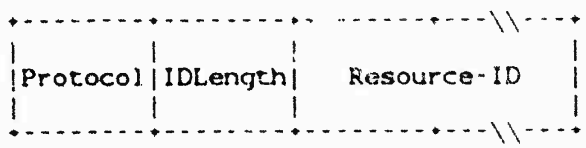
Bits 2-8 Reserved. Must be zero.

<Message-ID>
is a two octet (16-bit) value which identifies the request message. It is used simply to aid in matching requests with replies. The sending host should initialize this field to some convenient value when constructing a request message. The receiving host must return this same value in the <Message-ID> field of any reply message generated in response to that request.

<Resource-List>
is the list of resources being queried or for which location information is being supplied. This list is a sequence of octets beginning at the octet following the <Message-ID> and extending through the end of the UDP packet. The format of this field is explained more fully in the following section. The size of this list is implicitly specified by the length of the encapsulating UDP datagram.

4.1. Resource Lists

A <Resource-List> consists of zero or more resource specifiers. Each resource specifier is simply a sequence of octets. All resource specifiers have a common resource name initial format



where

<Protocol>
is the protocol number assigned to the lowest level Internet protocol utilized for accessing the resource.

<IDLength>
is the length of the resource identifier associated with this <Protocol>. This length may be a fixed or variable value depending on the particular resource. It is included so that specifiers which refer to resources which a host may not provide can be skipped over without needing to know the specific structure of the particular resource identifier. If the <Protocol> has no associated natural identifier, this length is 0.

<Resource-ID>
is the qualifying identifier used to further refine the resource being queried. If the <IDLength> field was 0, then this field is empty and occupies no space in the resource specifier.

In addition, resource specifiers in all <Who-Anywhere-Provides?>, <Does-Anyone-Provide?> and <They-Provide> messages also contain an additional qualifier following the <Protocol-ID>. This qualifier has the format

where

<IPLength>
is the number of IP addresses containing in the following <IP-Address-List> (the <IP-Address-List> field thus occupies the last 4*<IPLength> octets in its resource specifier). In request messages, this is the maximum number of qualifying addresses which may be included in the corresponding reply resource specifier. Although not particularly useful, it may be 0 and in that case provides no space for qualifying the resource name with IP addresses in the returned specifier. In reply messages, this is the number of qualifying addresses known to provide the resource. It may not exceed the number specified in the corresponding request specifier. This field may not be 0 in a reply message unless it was supplied as 0 in the request message and the confirming host would have returned one or more IP addresses had any space been provided.

<IP-Address-List>
is a list of four-octet IP addresses used to qualify the resource specifier with respect to those particular addresses. In reply messages, these are the IP addresses of the confirming host (when appropriate) and the addresses of any other hosts known to provide that resource (subject to the list length limitations). In request messages, these are the IP addresses of hosts for which resource information may not be returned. In such messages, these addresses should normally be initialized to some "harmless" value (such as the address of the querying host) unless it is intended to specifically exclude the supplied addresses from consideration in any reply messages.

The receiving host determines if it provides any of the resources named in the request list by successively decomposing each resource name. The first level of decomposition is the Internet protocol number which can presumably be looked up in a table to determine if that protocol is supported on the host. Subsequent decompositions are based on previous components until one of three events occur:

1. the current component identifies some aspect of the previous components which the host does not support,

2. the resource name from the request list is exhausted, or

3. the resource name from the request list is not exhausted but the host does not expect any further components in the name given the previous components

In case 1, the receiving host has determined that it does not provide the named resource. The resource specifier may not be included in any reply message returned.

In case 2, the receiving host has determined that it does indeed provide the named resource (note: this may occur even if the receiving host

would have expected the resource name to contain more components than were actually present). The resource specifier must be included (modulo IP address prohibitions) in any reply message returned.

In case 3, the receiving host has determined that it does not completely provide the named resource since name components remain which it does not understand (this might occur with specializations of or extensions to a known protocol which are not universally recognized). The resource specifier may not be included in any reply message returned.

5. Sample Usage

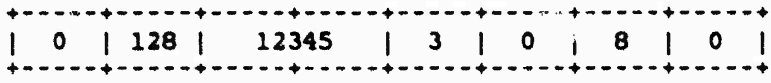
The following scenarios illustrate some typical uses of RLP. In all cases the indicated messages are encapsulated in a UDP datagram with the appropriate source and destination port numbers, message length, and checksum. This datagram is further encapsulated in an IP datagram with the appropriate source address of the sending host and destination address (either broadcast or individual) for the receiving host.

All numeric protocol examples are as specified in the appropriate protocol description documents listed in the references.

1. Suppose a freshly rebooted host H wishes to find some gateway on its directly connected network to which it can send its first external packet. It then broadcasts the request

<Who-Provides?> <Flags>=<Local-Only> <Message-ID>=12345
<Resource-List>={[GGP], [EGP]}

encoded as the 8 octet message

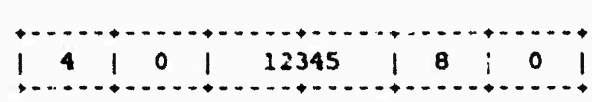


on its local network.

- Gateway G1 (which understands EGP) receives the request and returns the reply

<I-Provide> <Flags>=none <Message-ID>=12345
<Resource-List>={[EGP]}

encoded as the 6 octet message



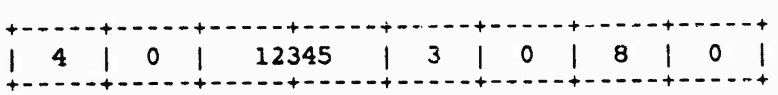
to host H which then remembers that gateway G1 may be used

to route traffic to the rest of the Internet.

- At the same time, gateway G2 (which understands both GGP and EGP) might also receive the request and return the reply

<I-Provide> <Flags>=none <Message-ID>=12345
<Resource-List>={[GGP], [EGP]}

encoded as the 8 octet message

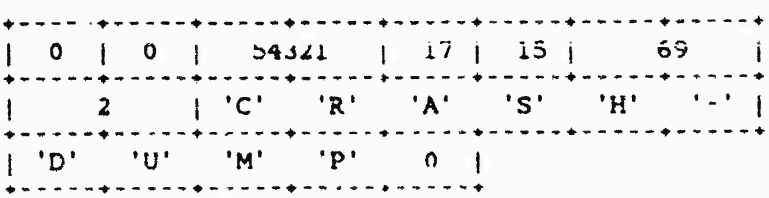


to host H which might then also add gateway G2 to its list if it chooses.

2. Assume instead that host H is a stand-alone system which has just encountered some fatal software error and wishes to locate a crash dump server to save its state before reloading. Suppose that the crash dump protocol on the host's local network is implemented using the Trivial File Transfer Protocol (TFTP) [8]. Furthermore, suppose that the special file name "CRASH-DUMP" is used to indicate crash dump processing (e.g. the server might locally generate a unique file name to hold each dump that it receives from a host). Then host H might broadcast the request

<Who-Provides?> <Flags>=none <Message-ID>=54321
<Resource-List>={[UDP, TFTP, WRQ, "CRASH-DUMP"]}

encoded as the 21 octet message

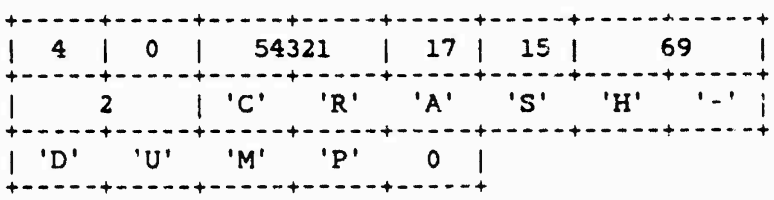


to its local network (note that the file name component is explicitly terminated by a null so as not to exclude future further specialization of the crash dump protocol).

- Host C (which supports this specialization of the TFTP protocol) receives the request and returns the reply

<I-Provide> <Flags>=none <Message-ID>=54321
<Resource-List>={[UDP, TFTP, WRQ, "CRASH-DUMP"]}

encoded as the 21 octet message



to host H which may then proceed to send its crash dump to host C and reload.

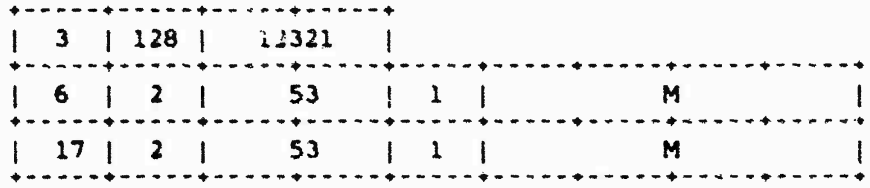
- Host D (which provides TFTP service but not the crash dump specialization), however, might receive the request and determine that it provides no support for the resource (since the resource name contains components following the UDP port number which it does not understand). It would therefore return no reply to host H.

3. Finally, suppose host M wishes to locate some domain name translation server (either UDP or TCP based) anywhere on the Internet. Furthermore, suppose that host M is on a IP network which does not provide broadcast address capabilities and that host R is a "known" resource location server for that network.

   First, since host M prefers to find a domain name server on its own locally connected network if possible, it sends the request

```
<Does-Anyone-Provide?> <Flags>=<Local-Only>
    <Message-ID>=12321 <Resource-List>=
  {[TCP, <DOMAIN-NAME-SERVER-PORT>] {M},
   [UDP, <DOMAIN-NAME-SERVER-PORT>] {M}}
```

   encoded as the 22 octet message



   to host R.

   Host R receives the request and consults its tables for any hosts known to support either variety of domain name service. It finds entries indicating that both hosts S and T provide UDP

based domain name service but that neither host is on the same IP network as host H. It must then send the negative confirmation reply

```
<They-Provide> <Flags>=none <Message-ID>=12321
               <Resource-List>={}
```

encoded as the 4 octet message

5	0	12321

back to host M.

Host M, receiving this reply, might now abandon any hope of finding a server on its own network, reformat its request to permit any host address, and resend

```
<Does-Anyone-Provide?> <Flags>=none <Message-ID>=12322
                       <Resource-List>=
              {[TCP, <DOMAIN-NAME-SERVER-PORT>] {M},
               [UDP, <DOMAIN-NAME-SERVER-PORT>] {M}}
```

encoded as the 22 octet message

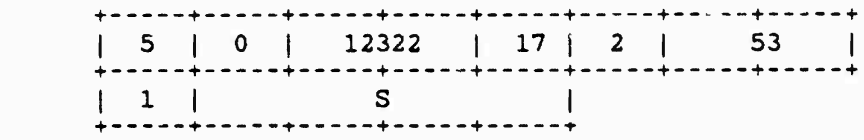
3	0	12322						
6	2	53	1	M				
17	2	53	1	M				

again to host R.

Host R receives this new request and is no longer constrained to return only local addresses. However, since only space for a single qualifying IP address was provided in each request resource specifier, it may not immediately return both addresses. Instead, it is forced to return only the first address and replies

```
<They-Provide> <Flags>=none <Message-ID>=12322
<Resource-List>={[UDP, <DOMAIN-NAME-SERVER-PORT>] {S}}
```

encoded as the 13 octet message

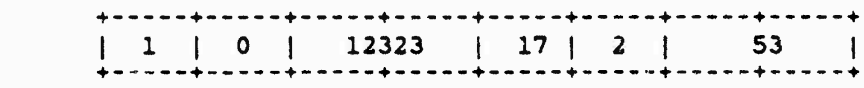


to Host M.

Host M receives the reply and (being the suspicious sort) decides to confirm it with host S. It then sends

<Do-You-Provide?> <Flags>=none <Message-ID>=12323
<Resource-List>={[UDP, <DOMAIN-NAME-SERVER-PORT>]}

encoded as the 8 octet message



to host S and receives back from host S the reply

<I-Provide> <Flags>=none <Message-ID>=12323
<Resource-List>={}

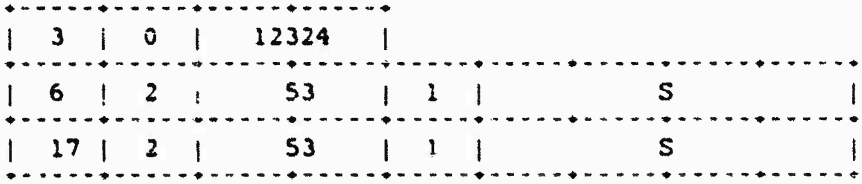
encoded as the 4 octet message

denying any support for UDP based domain name service.

In desperation host M again queries host R, this time excluding host S from consideration, and sends the request

<Does-Anyone-Provide?> <Flags>=none <Message-ID>=12324
<Resource-List>=
{[TCP, <DOMAIN-NAME-SERVER-PORT>] {S},
[UDP, <DOMAIN-NAME-SERVER-PORT>] {S}}

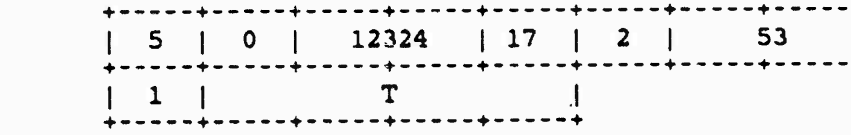
encoded as the 22 octet message



and this time receives the reply

<They-Provide> <Flags>=none <Message-ID>=12324
<Resource-List>={[UDP, <DOMAIN-NAME-SERVER-PORT>] {T}}

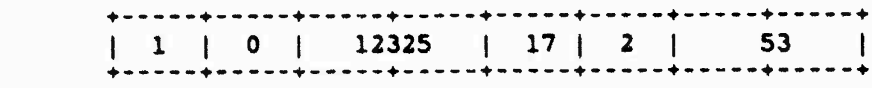
encoded as the 13 octet message



from host R which of course host M again insists on confirming by sending the request

<Do-You-Provide?> <Flags>=none <Message-ID>=12325
<Resource-List>=
{[UDP, <DOMAIN-NAME-SERVER-PORT>]}

encoded as the 8 octet message



to host T and finally receives confirmation from host T with the reply

<I-Provide> <Flags>=none <Message-ID>=12325
<Resource-List>={[UDP, <DOMAIN-NAME-SERVER-PORT>]}

encoded as the 8 octet message

that it indeed provides domain name translation service at UDP port 53.

A. Assigned Numbers

The "well-known" UDP port number for the Resource Location Protocol is 39 (47 octal).

REFERENCES

[1] Postel, J.
User Datagram Protocol.
RFC 768, USC/Information Sciences Institute, August, 1980.

[2] Postel, J.
File Transfer Protocol.
RFC 765, USC/Information Sciences Institute, June, 1980.

[3] Postel, J.
Internet Protocol - DARPA Internet Program Protocol Specification.
RFC 791, USC/Information Sciences Institute, September, 1981.

[4] Postel, J.
Transmission Control Protocol- DARPA Internet Program Protocol Specification.
RFC 793, USC/Information Sciences Institute, September, 1981.

[5] Postel, J.
Internet Control Message Protocol - DARPA Internet Program Protocol Specification.
RFC 792, USC/Information Sciences Institute, September, 1981.

[6] Reynolds, J., and J. Postel.
Assigned Numbers.
RFC 870, USC/Information Sciences Institute, October, 1983.

[7] Gurwitz, R., and R. Hinden.
IP - Local Area Network Addressing Issues.
IEN 212, Bolt Beranek and Newman, September, 1982.

[8] Sollins, K.
The TFTP Protocol (revision 2).
RFC 783, MIT/Laboratory for Computer Science, June, 1981.

(Oct. 16, 1972)
RFC 407 NIC 12112

Robert Bressler, MIT-DMCG
Richard Guida, MIT-DMCG
Alex McKenzie, BBN-NET

Obsoletes RFC 360

REMOTE JOB ENTRY PROTOCOL

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

REMOTE JOB ENTRY PROTOCOL

INTRODUCTION

Remote job entry is the mechanism whereby a user at one location causes a batch-processing job to be run at some other location. This protocol specifies the Network standard procedures for such a user to communicate over the Network with a remote batch-processing server, causing that server to retrieve a job-input file, process the job, and deliver the job's output file(s) to a remote location. The protocol uses a TELNET connection (to a special standardized logger, not socket 1) for all control communication between the user and the server RJE processes. The server-site then uses the File Transfer Protocol to retrieve the job-input file and to deliver the output file(s).

There are two types of users: direct users (persons) and user processes. The direct user communicates from an interactive terminal attached to a TIP or any host. This user may cause the input and/or output to be retrieved/sent on a specific socket at the specified host (such as for card readers or printers on a TIP), or the user may have the files transferred by file-id using File Transfer Protocol. The other type of user is a RJE User-process in one remote host communicating with the RJE Server-process in another host. This type of user ultimately receives its instructions from a human user, but through some unspecified indirect means. The command and response streams of this protocol are designed to be readily used and interpreted by both the human user and the user process.

A particular user site may choose to establish the TELNET control connection for each logical job or may leave the control connection open for extended periods. If the control connection is left open, then multiple job-files may be directed to be retrieved or optionally (to servers that are able to determine the end of one logical job by the input stream and form several jobs out of one input file) one continuous retrieval may be done (as from a TIP card reader). This then forms a "hot" card reader to a particular server with the TELNET connection serving as a "job monitor". Since the output is always transferred job at a time per connection to the output socket, the output from this "hot" reader would appear when ready as if to a "hot" printer. Another possibility for more complex hosts is to attach an RJE User-process to a card reader and take instructions from a lead control card, causing an RJE control TELNET to be opened to the appropriate host with appropriate log-on and input retrieval commands. This card reader would appear to the human user as a Network "hot" card reader. The details of this RJE User-process are beyond the scope of this protocol.

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

GENERAL SPECIFICATIONS

User

A human user at a real terminal or a process that supplies the command control stream causing a job to be submitted remotely will be termed the User. The procedure by which a process user receives its instructions is beyond the scope of this protocol.

User TELNET

The User communicates its commands over the Network in Network Virtual Terminal code through a User TELNET process in the User's Host. This User TELNET process initiates its activity via ICP to the standard "RJE Logger" socket (socket 5) at the desired RJE-server Host.

RJE-Server TELNET

The RJE-server process receives its command stream from and sends its response stream to the TELNET channel through an RJE-server TELNET process in the server host. This process must listen for the ICP on the "RJE Logger" socket (and cause appropriate ICP socket shifting).

TELNET Connection

The command and response streams for the RJE mechanism are via a TELNET-like connection to a special socket with full specifications according to the current NWG TELNET protocol.

RJE-Server

The RJE-Server process resides in the Host which is providing Remote Batch Job Entry service. This process receives input from the RJE-server TELNET, controls access through the "log-on" procedure, retrieves input job files, queues jobs for execution by the batch system, responds to status inquiries, and transmits job output files when available.

User FTP

All input and output files are transferred under control of the RJE-server process at its initiative. These files may be directly transferred via Request-for-connection to a specific Host/socket or they may be transferred via File Transfer Protocol. If the latter method is used, then the RJE-server acts through its local User FTP process to cause the transfer. This process initiates

activity by an active Request-for-connection to the "FTP Logger" in the foreign host.

Server FTP

This process in a remote host (remote from the RJE-server) listens for an ICP from the User FTP and then acts upon the commands from the User FTP causing the appropriate file transfer.

FTP

When File Transfer Protocol is used for RJE files, the standard FTP mechanism is used as fully specified by the current NWG FTProtocol.

RJE Command Language

The RJE system is controlled by a command stream from the User over the TELNET connection specifying the user's identity (log-on), the source of the job input file, the disposition of the job's output files, enquiring about job status, altering job status or output disposition. Additional commands affecting output disposition are includable in the job input file. This command language is explicitly specified in a following section of this protocol.

RJE Command Replies

Every command input from the User via TELNET calls for a response message from the RJE-server to the User over the TELNET connection. Certain other conditions also require a response message. These messages are formatted in a standardized manner to facilitate interpretation by both human Users and User processes. A following section of this protocol specifies the response messages.

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

RJE COMMANDS OVER TELNET CONNECTION

GENERAL CONVENTIONS

1. Each of the commands will be contained in one input line terminated by the standard TELNET "crlf". The line may be of any length desired by the user (explicitly, not restricted to a physical terminal line width). The characters "cr" and "lf" will be ignored by the RJE-server except in the explicit order "crlf" and may be used as needed for local terminal control.

2. All commands will begin with a recognized command name and may then contain recognized syntactic element strings and free-form variable strings (for user-id, file-ids, etc.). Recognized words consist of alphanumeric strings (letters and digits) or punctuation. Recognized alphanumeric string elements must be separated from each other and from unrecognizable strings by at least one blank or a syntacticly permitted punctuation. Other blanks may be used freely as desired before or after any syntactic element ("blank" is understood here to mean ASCII SPACE (octal 040); formally: <blank>::= <blank><ASCII SPACE> | <ASCII SPACE> ; thus, a sequence of SPACES is also permissible in place of <blank>, although there is no syntactic necessity for there to be more than one). The "=" after the command name in all commands except OUT and CHANGE is optional.

3. Recognized alphanumeric strings may contain upper case letters or lower case letters in any mixture without syntactic differentiation. Unrecognizable strings will be used exactly as presented with full differentiation of upper and lower case input, unless the host finally using the string defines otherwise.

4. There are two types of Unrecognizable strings: final and imbedded. Final strings appear as the last syntactic element of a command and are parsed as beginning with the next non-blank character of the input stream and continuing to the last non-blank character before the "crlf".

Imbedded strings include "job-id" and "job-file-id" in the OUT, CHANGE, and ALTER commands. At present these fields will be left undelimited since they must only be recognizable by the server host which hopefully can recognize its own job-ids and file-names.

SYNTAX

The following command descriptions are given in a BNF syntax. Names within angle brackets are non-terminal syntactic elements which are expanded in succeeding syntactic equations. Each equation has the

PASS

Pass = <password>

This command immediately follows a USER command and completes the "log-on" procedure. Although a particular Server may not require a password and has already indicated "log-on ok" after the USER command, every Server must permit a PASS command (and possibly ignore it) and acknowledge it with a "log-on ok" if the log-on is completed.

BYE

BYE

This command terminates a USER and requests the RJE server to close the TELNET connection. If input transfer is not in progress, the TELNET connection may be closed immediately; if input is in progress, the connection should remain open for result response and then be closed. During the interim, a new USER command (and no other command) is acceptable.

An unexpected close on the TELNET connection will cause the server to take the effective action of an ABORT and a BYE.

INID/INPASS

INID = <user-id>
INPASS = <password>

The specified user-id and password will be sent in the File Transfer request to retrieve the input file. These parameters are not used by the Server in any other way. If this command does not appear, then the USER/PASS parameters are used.

INPATH/INPUT

INPATH = <file-id>
INPUT = <file-id>
INPUT

NOTE: The following syntax will be used for output as well.

<file-id>::= <host-socket> | <host-file>
<host-socket>::= <host>,<socket><attributes> |
<socket><attributes>
no <host> part implies the User-site host
<host>::= <integer>
<socket>::= <integer>

defined name on the left of the ::= and a set of alternative definitions, separated by vertical lines "|", on the right.

REINITIALIZE

REINIT

This command puts the user into a state identical to the state immediately after a successful connection to the RJE-server, prior to having sent any commands over the TELNET connection. The effective action taken is that of an ABORT and a flushing of all INPUT, OUTPUT and ID information. Naturally, the user is still responsible for any usage charges incurred prior to his REINIT command. The TELNET connection is not affected in any way.

USER

User = <user-id>

This command must be the first command over a new TELNET connection. As such, it initiates a "logon" sequence. The response to this command is one of the following:

1. User code in error.
2. Enter password (if user code ok).
3. Log-on ok, proceed (if no password requested).

Another USER command may be sent by the User at any time to change Users. Further input will then be charged to the new user. A server may refuse to honor a new user command if it is not able to process it in its current state (during input file transfer, for example), but the protocol permits the USER command at any time without altering previous activity. An incorrect subsequent USER command or its following PASS command are to be ignored with error response, leaving the original User logged-in.

It is permissable for a server to close the TELNET connection if the initial USER/PASS commands are not completed within a server specified time period. It is not required or implied that the "logged-on" User's user-id be the one used for file transfer or job execution, but only identifies the submitter of the command stream. Servers will establish their own rules relating user-id with the job-execution-user for Job or Output alteration commands.

Successful "log-on" always clears any previous Input or Output default parameters (INID, etc.).

```
<integer>::= D<decimal-integer> | O<octal-integer> |
             H<hexadecimal-integer>
<host-file>::= <host><attributes>/<pathname>
<attributes>::= <empty> | :<transmission><code>
<transmission>::= <empty> | T | A | N
      <empty> implies default which is N for Input files
              and A for Output files
      T       specifies TELNET-like coding with embedded
              "crlf" for new-line, "ff" for new-page
      N       specifies FTP blocked transfer with record
              marks but without other carriage-control
      A       specifies FTP blocked records with ASA
              carriage-control
              (column 1 of image is forms control)
<code>::= <empty> | E
      <empty> specifies NVT ASCII code
      E specifies EBCDIC
<pathname>::= <any string recognized by the FTP Server at
              the site of the file>
```

The <file-id> syntax is the general RJE mechanism for specifying a particular file source or destination for input or output. If the <host-socket> form is used then direct transfer will be made by the RJE-Server to the named socket using the specified <attributes>. If the <host-file> form is used then the RJE-server will call upon its local FTP-user process to do the actual transfer. The data stream in this mode is either TELNET-like ASCII or blocked records (which may use column 1 for ASA carriage-control). Although A mode is permitted on input (column 1 is deleted) the usual mode is the default N. The output supplies carriage-control in the first character of each record ("blank" = single-space, "1" = new-page, etc.), while the optional N mode transfers the data only (as to a card punch, etc.).

The <pathname> is an arbitrary Unrecognized string which is saved by RJE-server and sent back over FTP to the FTP-server to retrieve or store the appropriate files.

INPATH or INPUT commands first store the specified <file-id> if one is supplied, and then the INPUT command initiates input. The INPATH name may be used to specify a file-id for later input and the INPUT command without file-id will cause input to initiate over a previously specified file-id. An INPUT "crlf" command with no previous <file-id> specified is illegal.

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

ABORT

ABORT

This command aborts any input retrieval in progress, discards already received records, and closes the retrieval connection. Note: ABORT with parameters is an Output Transmission control (see below).

OUTUSER/OUTPASS

OUTUSER = <user-id>
OUTPASS = <password>

The specified user-id and password will be sent in the File Transfer request to send the output file(s). These parameters are not used by the Server in any other way. If this command does not appear, then the USER/PASS parameters are used.

OUT

OUT <out-file> = <disp>

<out-file>::= <empty> | <job-file-id>
<empty> implies the primary print file of the job
<job-file-id>::= <string representing a specific output file from the job as recognized by the Server>
<disp>::= <empty><file-id> | (H) | (S)<file-id>|(D)
<empty> specifies Transmit then discard
(H) specifies Hold-only, do not transmit
(S) specifies Transmit and Save
(D) specifies discard without transmitting
Note: Parentheses are part of the above elements.

<file-id>::= (same as for INPUT command)

This command specifies the disposition of output file(s) produced by the job. Unspecified files will be Hold-only by default. The OUTUSER, OUTPASS, and OUT commands must be specified before the INPUT command to be effective. These commands will affect any following jobs submitted by this USER over this RJE-TELNET connection. A particular job may override these commands by NET control cards on the front of the input file.

Once output disposition is specified by this OUT command or by a NET OUT card, the information is kept with the job until final output disposition, and is modifiable by the CHANGE command.

On occasion, the server may find that the destination for the output is "busy" (i.e., RFC to either Server-FTP or specified socket is refused), or that the host which should receive the output is dead. In these cases, the server should wait several minutes and then try to transmit again.

OUTPUT RE-ROUTE

CHANGE <job-id><blank><out-file> = <disp>

This command changes the output disposition supplied with the job at submission. The <job-id> is assumed recognizable by the RJE-server, who may verify if this USER is authorized to modify the specified job. After the job is identified, the other information has the same syntax and semantics as the original OUT command. CHANGE command may be specified for a job-file-id which was not mentioned at submission time and has the same effect as an original OUT command.

OUTPUT CONTROLS DURING TRANSMISSION

<command><blank><count><blank><what>

<command>::= RESTART | RECOVER | BACK | SKIP |
ABORT | HOLD

These commands specify (respectively):

Restart the transmission (new RFC, etc.)
Recover restarts transmission from last FTP
Restart-marker-reply
(see FTP).
Back up the output "count" blocks
Skip the output forward "count" blocks
Abort the output, discarding it
Abort the output, but Hold it

<count>::= <empty> | <integer>
<empty> implies 1 where defined
<what>::= @<file-id> | <job-id><job-file-id>
<disp>::= (same as for OUT command)
<file-id>::= (same as for INPUT command)
<integer>::= (same as for INPUT command)
<job-id>::= <server recognized job identifier which was supplied
at INP completion by the server>

<job-file-id>::= <server recognized file identifier or if missing
then the prime printer output of the specified
job>

This collection of commands will modify the transmission of output in progress or recently aborted. If output transmission is cut-off before completion, then the RJE-server will either try to resend the entire file if the file's <disp> was Transmit-and-discard or will Hold the file for further User control if the <disp> was (S) transmit-and-Save. Either during transmission, during the Save part of a transmit-and-Save, or for a Hold-only file, the above commands may be used to control the transmission. The @<file-id> form of <what> is permitted only if transmission is actually in progress.

If the file's state is inconsistent with the command, then the command is illegal and ignored with reply.

STATUS

STATUS <job-id>
STATUS <job-id><blank><job-file-id>

These commands request the status of the RJE-server, a particular job, or the transmission of an output or input file, respectively. The information content of the Status reply is site dependent.

CANCEL/ALTER

CANCEL <job-id>
ALTER <job-id><blank><site dependent options>

These commands change the course of a submitted job. CANCEL specifies that the job is to be immediately terminated and any output discarded. ALTER provides for system dependent options such as changing job priority, process limits, Teminate without Cancel, etc.

OP

OP (any string)

The specified string is to be displayed to the Server site operator when any following job is initiated from the batch queue of the Server. This command usually appears in the input file as a NET OP control card, but may be a TELNET command. It is cancelled as an all-jobs command by an OP "crlf" command (no text supplied).

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

RJE CONTROL CARDS IN THE INPUT FILE

Certain RJE commands may be specified by control cards in the front of the input file. If these controls appear, they take precedence over the same command given thru the RJE-TELNET connection and affect only this specific job. All these RJE control cards must appear as the first records of the job's input-file. They all contain the control word NET in columns 1 through 3. Scanning for these controls stops when the first card without NET in col 1-3 is encountered.

The control commands appear in individual records and are terminated by the end-of-record (usually an 80 column card-image). Continuation is permitted onto the next record by the appearance of NET+ in columns 1-4 of the next record. Column 5 of the next record immediately follows the last character of the previous record.

NET OUTUSER = <user-id>
NET OUTPASS = <password>
NET OUT <out-file> = <disp>
NET OP <any string>

See the corresponding TELNET command for details. One option permitted by the NET OUTUSER and NET OUT controls not possible from the TELNET connection is specification of different OUTUSERs for different OUTS, since the TELNET stored and supplies only an initial OUTUSER, but the controls may change OUTUSERs before each OUT control is encountered.

RJE USE OF FILE TRANSFER PROTOCOL

Most non-TIP files will be transferred to or from the RJE-server through the FTP process. RJE-server will call upon its local FTP-user supplying the Host, File-pathname, User-id, Password, and Mode of the desired transfer. FTP-user will then connect to its FTP-server counterpart in the specified host and set up a transfer path. Data will then flow through the RJE-FTP interface in the Server, over the Network, from/to the foreign FTP-server and then from/to the specified File-pathname in the foreign host's file storage space. On output files, the file-pathname may be recognized by the foreign host as directions to a printer or the file may simply be stored; a User-RJE-process can supply an output <file-id> by default which is recognized by its own Server-FTP as routing to a printer.

Although many specifics of the RJE-Server/User-FTP interface are going to be site dependent, there are several FTP options which will be used in a standard way by RJE-Servers:

1. A new FTP connection will be initiated for each file to be transferred. The connection will be opened with the RJE User supplied User-id (OUTUSER or INUSER) and Password.

2. The data bytesize will be 8 bits.

3. The FTP Type, Structure, and Mode parameters are determined by the RJE transfer direction (I/O), and the <transmission> and <code> options supplied by the User:

I/O	<TRANS>	<CODE>	FTP-TYPE	FTP-STRUCTURE	FTP-MODE
I*	N	-	A	R	B
I	N	E	E	R	B
I	T	-	A	F	S
I	T	E	E	F	S
I	A	-	P	R	B
I	A	E	F	R	B
O*	A	-	P	R	B
O	A	E	F	R	B
O	N	-	A	R	B
O	N	E	E	R	B
O	T	-	A	F	S
O	T	E	E	F	S

(*indicates default)

4. The service commands used will be Retrieve for input and Append (with create) for output. The FTP pathname will be the <pathname> supplied by the RJE User.

5. On output in B form, the User-FTP at the RJE-Server site will send Restart-markers at periodic intervals (like every 100 lines, or so), and will remember the latest Restart-marker-reply with the file. If the file transfer is not completed and the <disp> is (S) then the file will be held pending User intervention. The User may then use the RECOVER command to cause a FTP restart at the last remembered Restart-marker-reply.

6. The FTP Abort command will be used for the RJE ABORT and CANCEL commands.

7. For transfers where the FTP-MODE is defined as B, the user FTP may optionally attempt to use H mode.

The specific form of the FTP commands used by an RJE-Server site, and the order in which they are used will not be specified in this protocol.

Errors encountered by FTP fall into three categories: a) access errors or no storage space error; b) command format errors; and c) transfer failure errors. Since the commands are created by the RJE-Server process, an error is a programming problem and should be logged for attention and the situation handled as safely as possible. Transmission failure or access failure on input cause an effective ABORT and user notification. Transmission failure on output causes RESTART or Save depending on <disp> (see OUT command). Access failure on output is a problem since the User may not be accessible. A status response should be queued for him, should he happen to inquire; a <disp> = (S) file should be Held; and a <disp> = <empty> transmit-and-discard file should be temporarily held and then discarded if not claimed. "Temporarily" is understood here to mean at least several days, since particularly in the case of jobs which generate voluminous output at great expense to the User, he should be given every chance to retrieve his rightful output. Servers may elect, however, to charge the User for the file-storage space occupied by the held output.

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

REPLIES OVER THE TELNET CONNECTION

Each action of the RJE-server, including entry of each TELNET command, is noted over the TELNET connection to the User. These RJE-server replies are formatted for Human or Process interpretation. They consist of a leading 3-digit numeric code followed by a blank followed by a text explanation of the message. The numeric codes are assigned by groups for future expansion to hopefully cover other protocols besides RJE (like FTP). The numeric code is designed for ease of interpretation by processes. The three digits of the code are interpreted as follows:

The first digit specified the "type" of response indicated:

000

These "replies" are purely informative, and are issued voluntarily by the Server to inform a User of some state of the server's system.

100

Replies to a specific status inquiry. These replies serve as both information and an acknowledgment of the status request.

200

Positive acknowledgment of some previous command/request. The reply 200 is a generalized "ok" for commands which require no other comment. Other 2xx replies are specified for specific successful actions.

300

Incomplete information supplied so far. No major problem, but activity cannot proceed with the input specified.

400

Unsuccessful reply. A request was correctly specified, but could not be correctly completed. Further attempts will require User commands.

500

Incorrect or illegal command. The command or its parameters were invalid or incomplete from a syntactic view, or the command is inconsistent with a previous command. The command in question has been totally ignored.

600-900

Reserved for expansion

The second digit specifies the general subject to which the response refers:

x00-x29

General purpose replies, not assignable to other subjects.

x30

Primary access. These replies refer to the attempt to "log-on" to a Server service (RJE, FTP, etc.).

x40

Secondary access. The primary Server is commenting on its ability to access a secondary service (RJE must log-on to a remote FTP service).

x50

FTP results.

x60

RJE results.

x70-x99

Reserved for expansion.

The final digit specifies a particular message type. Since the code is designed for an automaton process to interpret, it is not necessary for every variation of a reply to have a unique number, only that the basic meaning have a unique number. The text of a reply can explain the specific reason for the reply to a human User.

Each TELNET line (ended by "crlf") from the Server is intended to be a complete reply message. If it is necessary to continue the text of a reply onto following lines, then those continuation replies contain the special reply code of three blanks.

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

The assigned reply codes relating to RJE are:

000 General information message (time of day, etc.)
030 Server availability information
050 FTP commentary or user information
060 RJE or Batch system commentary or information
100 System status reply
150 File status reply
151 Directory listing reply
160 RJE system general status reply
161 RJE job status reply
200 Last command received ok
201 An ABORT has terminated activity, as requested
202 ABORT request ignored, no activity in progress
203 The requested Transmission Control has taken effect
204 A REINIT command has been executed, as requested
230 Log-on completed
231 Log-off completed, goodbye.
232 Log-off noted, will complete when transfer done
240 File transfer has started
250 FTP File transfer started ok
251 FTP Restart-marker-reply
Text is: MARK yyyy = mmmm
where yyyy is data stream marker value (yours)
and mmmm is receiver's equivalent mark (mine)
252 FTP transfer completed ok
253 Rename completed
254 Delete completed
260 Job <job-id> accepted for processing
261 Job <job-id> completed, awaiting output transfer
262 Job <job-id> Cancelled as requested
263 Job <job-id> Altered as requested to state <status>
264 Job <job-id>,<job-file-id> transmission in progress
300 Connection greeting message, awaiting input
301 Current command not completed (may be sent after
suitable delay, if not "crlf")
330 Enter password (may be sent with hide-your input mode)
360 INPUT has never specified an INPATH
400 This service is not implemented
401 This service is not accepting log-on now, goodbye.
430 Log-on time or tries exceeded, goodbye.
431 Log-on unsuccessful, user and/or password invalid
432 User not valid for this service
434 Log-out forced by operator action, please phone site
435 Log-out forced by system problem
436 Service shutting down, goodbye
440 RJE could not log-on to remote FTP for input transfer
441 RJE could not access the specified input file thru FTP
442 RJE could not establish <host-socket> input connection

443 RJE could not log-on to remote FTP for output delivery
444 RJE could not access file space given for output
445 RJE could not establish <host-socket> output connection
450 FTP: The named file does not exist (or access denied)
451 FTP: The named file space not accessable by YOU
452 FTP: Transfer not completed, data connection closed
453 FTP: Transfer not completed, insufficient storage space
460 Job input not completed, ABORT performed
461 Job format not acceptable for processing, Cancelled
462 Job previously accepted has mysteriously been lost
463 Job previously accepted did not complete
464 Job-id referenced by STATUS, CANCEL, ALTER, CHANGE, or Transmission Control is not known (or access denied)
465 Request Alteration is not permitted for the specified job
466 Un-deliverable, un-claimed output for <job-id> discarded
467 Requested REINIT not accomplished
500 Last command line completely unrecognized
501 Syntax of the last command is incorrect
502 Last command incomplete, parameters missing
503 Last command invalid, illegal parameter combination
504 Last command invalid, action not possible at this time
505 Last command conflicts illegally with previous command(s)
506 Requested action not implemented by this Server
507 Job <job-id> last command line completely unrecognized
508 Job <job-id> syntax of the last command is incorrect
509 Job <job-id> last command incomplete, parameters missing
510 Job <job-id> last command invalid, illegal parameter combination
511 Job <job-id> last command invalid, action impossible at this time
512 Job <job-id> last command conflicts illegally with previous command(s)

SEQUENCING OF COMMANDS AND REPLIES

The communication between the User and Server is intended to be an alternating dialogue. As such, the User issues an RJE command and the Server responds with a prompt primary reply. The User should wait for this initial success or failure response before sending further commands.

A second type of reply is sent by Server asynchronously with respect to User commands. These replies report on the progress of a job submission caused by the INPUT command and as such are secondary replies to that command.

The final class of Server "replies" are strictly informational and may arrive at any time. These "replies" are listed below as spontaneous.

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

COMMAND-REPLY CORRESPONDENCE TABLE

COMMAND	SUCCESS	FAILURE
REINIT	204	467,500-505
USER	230,330	430-432,500-505
PASS	230	430-432,500-505
BYE	231,232	500-505
INID	200	500-505
INPASS	200	500-505
INPATH	200	500-505
INPUT	240	360,440-442,500-505
sec. input retrieval	260	460,461
sec. job execution	261	462,463
sec. output transmission	-	443-445,466
ABORT (input)	201,202	500-505
OUTUSER	200	500-505
OUTPASS	200	500-505
OUT	200	500-505
CHANGE	200	500-505
RESTART/RECOVER/BACK /SKIP/ABORT (output)/HOLD	203	464,500-506
STATUS	1xx,264	460-465,500-505
CANCEL	262	464,500-506
ALTER	263	464,465,500-506
OP	200	500-505
Spontaneous	0xx,300,301	434-436

Note: For commands appearing on cards, a separate set of error codes is provided (507-512). Since these error replies are "asynchronously" sent, and thus could cause some confusion if the user is in the process of submitting a new job after the present one, the error replies must identify which job has the faulty card(s).

TYPICAL RJE SCENARIOS

TIP USER WANTING HOT CARD READER TO HOSTX

1. TIP user opens TELNET connection to HOSTX socket 5

2. Commands sent over TELNET to RJE

   USER=myself
   PASS=dorwssap
   OUT=H70002
   INPUT=H50003

3. RJE-server connects to the TIP's device 5 and begins reading. When end-of-job card is recognized, the job is queued to run. The connection to the card reader is still open for more input as another job.

4. The first job finishes. A connection to the TIP's device 7 is established by RJE-server and the output is sent as an NVT stream.

5. Continue at any time with another deck at step 3.

TIP WITH JOB-AT-A-TIME CARD READER

1. thru 4) the same but User closes Reader after the deck

2. The output finishes and the printer connection closes.

3. INPUT may be typed any time after step 3 finishes and another job will be entered starting at 3.

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

HOSTA USER RUNS JOB AT HOSTC, INPUT FROM HOSTB

1. User TELNET connects to HOSTC socket 5 for RJE

   USER=roundabout
   PASS=aaabbbc
   OUTUSER=roundab1
   OUT=:E/.sysprinter
   OUT puncher = (S)HOSTB:NE/my.savepunch
   INUSER=rounder
   INPASS=x.x.x
   INPUT=HOSTB:E/my.jobinput

2. The RJE-server has FTP retrieve the input from HOSTB using User-id of "rounder" and Password of "x.x.x" for file named "my.jobinput".

3. The job finishes. RJE-server uses FTP to send two files: the print output is sent to HOSTA in EBCDIC with ASA carriage control to file ".sysprinter" while the file known as "puncher" is sent to HOSTB in EBCDIC without carriage-control to file "my.savepunch".

4. When the outputs finish, RJE-server at HOSTC discards the print file but retains the "puncher" file.

5. The User who has signed out after job submission has gotten his output and checked his file "my.savepunch" at HOSTB. He deletes the saved copy at HOSTC by re-calling RJE at HOSTC.

   USER=roundabout
   PASS=aaabbbcc
   ABORT job 123 puncher
         or
   CHANGE job 123 puncher = (D)

RFC 740
NETRJS Protocol

RTB 42423 22 Nov 77

Network Working Group
Request for Comments: 740
NIC: 42423
Obsoletes: 189, 599

R. Braden
UCLA-CCN
22 November 1977

NETRJS PROTOCOL

A. Introduction

NETRJS, a private protocol for remote job entry service, was defined and implemented by the UCLA Campus Computing Network (CCN) for batch job submission to an IBM 360 Model 91. CCN's NETRJS server allows a remote user, or a daemon process working in behalf of a user, to access CCN's RJS ("Remote Job Service") subsystem. RJS provides remote job entry service to real remote batch (card reader/line printer) terminals over direct communications lines as well as to the ARPANET.

A batch user at a remote host needs a NETRJS user process to communicate with the NETRJS server at the batch host. An active NETRJS user process simulates a "Virtual Remote Batch Terminal", or "VRBT".

A VRBT may have virtual card readers, printers, and punches. In addition, every VRBT has a virtual remote operator console. Using a virtual card reader, a Network user can transmit a stream of card images comprising one or more batch jobs, complete with job control language ("JCL"), to the batch server host. The NETRJS server will cause these jobs to be spooled into the batch system to be executed according to their priority. NETRJS will automatically return the print and/or punch output images which are created by these jobs to the virtual printer and/or card punch at the VRBT from which the job was submitted. The batch user can wait for his output, or he can signoff and signon again later to receive it.

To initiate a NETRJS session, the user process must execute a standard ICP to a fixed socket at the server. The result is to establish a full-duplex Telnet connection for the virtual remote operator console, allowing the VRBT to signon to RJS. The virtual remote operator console can then be used to issue commands to NETRJS and to receive status, confirmation, and error messages from the

server. The most important remote operator commands are summarized in Appendix D.

Different VRBT's are distinguished by 8-character terminal id's, which are assigned by the server site to individual batch users or user groups.

B. Connections and Protocols

The protocol uses up to five connections between the user and server processes. The operator console uses a a full-duplex Telnet connection. The data transfer streams for the virtual card reader, printer, and punch each use a separate simplex connection under a data transfer protocol defined in Appendix A. This document will use the term "channel" for one of these simplex data transfer connections and will designate a connection "input" or "output" with reference to the server.

A particular data transfer channel needs to be open only while it is in use, and different channels may be used sequentially or simultaneously. CCN's NETRJS server will support simultaneous operation of a virtual card reader, a virtual printer, and a virtual punch (in addition to the operator console) on the same VRBT process. The NETRJS protocol could easily be extended to any number of simultaneously-operating virtual card readers, printers, and punches.

The NETRJS server takes a passive role in opening the data channels: the server only "listens" for an RFC from the user process. NETRJS is defined with an 8-bit byte size on all data channels.

Some implementations of NETRJS user processes are daemons, operating as background processes to submit jobs from a list of user requests; other implementations are interactive processes executed directly under terminal control by remote users. In the latter case, the VRBT process generally multiplexes the user terminal between NETRJS, i.e., acting as the remote operator console, and entering local commands to control the VRBT. Local VRBT commands allow selection of the files containing job streams to be sent to the server as well as files to receive job output from the server. Other local commands would cause the VRBT to open data transfer channels to the NETRJS server and to close these channels to free buffer space or abort transmission.

The user process has a choice of three ICP sockets, to select the character set of the VRBT -- ASCII-68, ASCII-63, or EBCDIC. The server will make the corresponding translation of the data in the card reader and printer channels. (In the CCN implementation of NETRJS, an EBCDIC VRBT will transmit and receive, without

translation, "transparent" streams of 8-bit bytes, since CCN is an EBCDIC installation). The punch stream will always be transparent, outputting "binary decks" of 80-byte records untranslated. The operator console connections always use Network ASCII, as defined by the Telnet protocol.

The NETRJS protocol provides data compression, replacing repeated blanks or other characters by repeat counts. However, when the terminal id is assigned, a particular network VRBT may be specified to use no data compression. In this case, NETRJS will simply truncate trailing blanks and send records in a simple "op code-length-data" form, called "truncated format" (see Appendix A).

C. Starting and Terminating a Session

The remote user establishes a connection to the NETRJS server by executing an ICP to the contact socket 71 (decimal) for EBCDIC, socket 73 (decimal) for ASCII-68, or to socket 75 (decimal) for ASCII-63. A successful ICP results in a pair of connections which are in fact the NETRJS operator console connections. NETRJS will send a READY message over the operator output connection.

The user (process) must now enter a valid NETRJS signon command ("SIGNON terminal-id") through the virtual remote operator console. RJS will normally acknowledge signon with a console message; however, if there is no available NETRJS server port, NETRJS will indicate refusal by closing both operator connections. If the user fails to enter a valid signon within 3 minutes, NETRJS will close the operator connections. If the VRBT attempts to open data transfer channels before the signon command is accepted, the data transfer channels will be refused with an error message to the VRBT operator console.

Suppose that S is the even number sent in the ICP; then the NETRJS connections have sockets at the server with fixed relation to S, as shown in the following table:

Channel	Server Socket	User Socket
Remote Operator Console Input	S	U + 3 Telnet
Remote Operator Console Output	S + 1	U + 2 Telnet
Data Transfer - Card Reader #1	S + 2	any odd number
Data Transfer - Printer #1	S + 3	any even number
Data Transfer - Punch #1	S + 5	any even number

Once the VRBT has issued a valid signon, it can open data transfer channels and initiate input and output operations as explained in the following sections. To terminate the session, the VRBT may close all

connections. Alternatively, it may enter a SIGNOFF command through the virtual remote operator console. Receiving a SIGNOFF, NETRJS will wait until the current job output streams are complete and then itself terminate the session by closing all connections.

D. Input Operations

A job stream for submission to the NETRJS server is a series of logical records, each of which is a card image of at most 80 characters. The user can submit a "stack" of successive jobs through the card reader channel with no end-of-job indication between jobs; NETRJS is able to parse the JCL sufficiently to recognize the beginning of each job.

To submit a batch job or stack of jobs for execution, the user process must first open the card reader channel by issuing an Init for foreign socket S+2 and the appropriate local socket. NETRJS, which is listening on socket S+2, will return an RTS command to open the channel. When the channel is open, the user can begin sending his job stream using the protocol defined in Apendix A. For each job successfully spooled, NETRJS will send a confirming message to the remote operator console.

At the end of the job stack, the user process must send an End-of-Data transaction to initiate processing of the last job. NETRJS will then close the channel (to avoid holding buffer space unnecessarily). At any time during the session, the user process can re-open the card reader channel and transmit another job stack. It can also terminate the session and signon later to get the output.

If the user process leaves the channel open for 5 minutes without sending any bits, the server will abort (close) the channel. The user process can abort the card reader channel at any time by closing the channel; NETRJS will then discard the last partially spooled job. If NETRJS finds an error (e.g., transaction sequence number error or a dropped bit), it will abort the channel by closing the channel prematurely, and also inform the user process that this job was discarded (thus solving the race condition between End-of-Data and aborting). The user process should retransmit only those jobs in the stack that have not been completely spooled.

If the user's process, NCP, or host, or the Network itself fails during input, RJS will discard the job being transmitted. A message informing the user that this job was discarded will be generated and sent to him the next time he signs on. On the other hand, those jobs whose receipt have been acknowledged on the operator's console will not be affected by the failure, but will be executed by the server.

E. Output Operations

The VRBT may wait to set up a virtual printer or punch and open its channel until a STATUS message from NETRJS indicates output is ready; or it may leave the output channel(s) open during the entire session, ready to receive output whenever it becomes available. The VRBT can also control which one of several available jobs is to be returned by entering appropriate operator commands.

To be prepared to receive printer (or punch) output from its jobs, the VRBT issues an Init for foreign socket S+3 or S+5 for printer or punch output, respectively. NETRJS is listening on these sockets and should immediately return an STR. However, it is possible that because of a buffer shortage, NETRJS will refuse the connection by returning a CLS; in this case, try again later.

When NETRJS has job output for a particular virtual terminal and a corresponding open output channel, it will send the output as a series of logical records using the protocol in Appendix A. The first record will consist of the job name (8 characters) followed by a comma and then the ID string from the JOB card, if any. In the printer stream, the first column of each record after the first will be an ASA carriage control character (see Appendix C). A virtual printer in NETRJS has 254 columns, exclusive of carriage control; NETRJS will send up to 255 characters of a logical record it finds in a SYSOUT data set. If the user wishes to reject or fold records longer than some smaller record size, he can do so in his VRBT process.

NETRJS will send an End-of-Data transaction and then close an output channel at the end of the output for each complete batch job; the remote site must then send a new RFC to start output for another job. This gives the remote site a chance to allocate a new file for each job without breaking the output within a job.

If the batch user wants to cancel (or backspace or defer) the output of a particular job, he can enter appropriate NETRJS commands on the operator input channel (see Appendix D).

If NETRJS encounters a permanent I/O error in reading the disk data set, it will notify the user via his console, skip forward to the next set of system messages or SYSOUT data set in the same job, and continue. If the user process stops accepting bits for 5 minutes, the server will abort the channel. In any case, the user will receive notification of termination of output data transfer for each job via a remote console message.

If the user detects an error in the stream, he can issue a Backspace (BSP) command from his console to repeat the last "page" of output, or a Restart (RST) command to repeat from the last SYSOUT data set or the beginning of the job, or he can abort the channel by closing his socket. If he aborts the channel, NETRJS will simulate a Backspace command, and when the user re-opens the channel the job will begin transmission again from an earlier point in the same data set. This is true even if the user terminates the current session first and reopens the channnel in a later session; RJS saves the state of every incomplete output stream. However, before re-opening the channel he can defer this job for later output, restart it at the beginning, or cancel its output (see Appendix D). Note that aborting the channel is only effective if NETRJS has not yet sent the End-of-Data transaction.

If the user's process, NCP, or host or the Network itself fails during an output operation, NETRJS will act as if the channel had been aborted and the user signed off. NETRJS will discard the output of a job only after receiving the RFNM from the last data transfer message (containing an End-of-Data). In no case should a NETRJS user lose output from a batch job.

APPENDIX A

Data Transfer Protocol in NETRJS

1. Introduction

The records in the data transfer channels (for virtual card reader, printer, and punch) are generally grouped into transactions preceded by headers. The transaction header includes a sequence number and the length of the transaction. Network byte size must be 8 bits in these data streams.

A transaction is the unit of buffering within the server software, and is limited to 880 8-bit bytes. Transactions can be as short as one record; however, those sites which are concerned with efficiency should send transactions as close as possible to the 880 byte limit.

There is no necessary connection between physical message boundaries and transactions ("logical messages"); the NCP can break a transaction arbitrarily into physical messages. The CCN server starts each transaction at the beginning of a new physical message, but this is not a requirement of the protocol.

Each logical record within a transaction begins with an "op code" byte which contains the channel identification, so its value is unique to each channel but constant within a channel. This choice provides the receiver with a convenient way to verify bit-synchronization, and it also allows an extension in the future to true "multi-leaving" (i.e., multiplexing all channels within one connection in each direction).

The only provisions for transmission error detection in the current NETRJS protocol are (1) the "op code" byte to verify bit synchronization and (2) the transaction sequence number. Under the NETRJS protocol, a data transfer error must abort the entire transmission; there is no provision for restart.

2. Meta-Notation

The following description of the NETRJS data transfer protocol uses a formal notation derived from that proposed in RFC 31 by Bobrow and Sutherland. The notation consists of a series of productions for bit string variables. Each variable name which represents a fixed length field is followed by the length in bits (e.g., SEQNUMB(16)). Numbers enclosed in quotes are decimal, unless qualified by a leading X meaning hex. Since each hex digit is 4 bits, the length is not shown explicitly in hex numbers. For example, '255'(8) and X'FF' both represent a string of 8 one bits.

The meta-syntactic operators are:

| | :alternative string

[ ] :optional string

( ) :grouping

+ :catenation of bit strings

The numerical value of a bit string (interpreted as an integer) is symbolized by a lower case identifier preceding the string expression and separated by a colon. For example, in "i:FIELD(8)", i symbolizes the numeric value of the 8 bit string FIELD.

Finally, we use Bobrow and Sutherland's symbolism for iteration of a sub-string: (STRING-EXPRESSION = n); denotes n occurrences of STRING-EXPRESSION, implicitly catenated together. Here any n greater or equal to 0 is assumed unless n is explicitly restricted.

3. Protocol Definition

STREAM ::= (TRANSACTION = n) + [END-OF-DATA]

That is, STREAM, the entire sequence of data on a particular open channel, is a sequence of n TRANSACTIONS followed by an END-OF-DATA marker (omitted if the sender aborts the channel).

TRANSACTION ::= THEAD(72) + (RECORD = r) + ('0'(1) = f)

That is, a transaction consists of a 72 bit header, r records, and f filler bits; it may not exceed 880*8 bits.

```
THEAD ::= X'FF'+f:FILLER(8)+SEQNUMB(16)+LENGTH(32)+X'00'
```

Transactions are to be consecutively numbered in the SEQNUMB field, starting with 0 in the first transaction after the channel is (re-) opened. The 32 bit LENGTH field gives the total length in bits of the r RECORD's which follow. For convenience, the using site may add f additional filler bits at the end of the transaction to reach a convenient word boundary on his machine; the value f is transmitted in the FILLER field of THEAD.

```
RECORD ::= COMPRESSED | TRUNCATED
```

RJS will accept intermixed RECORD's which are COMPRESSED or TRUNCATED in an input stream. RJS will send one or the other format in the printer and punch streams to a given VRBT; the choice is determined for each terminal id.

```
COMPRESSED ::= '2'(2) + DEVID(6) + (STRING = p) + '0'(8)

STRING     ::= ('6'(3) + i:DUPCOUNT(5))  |
```

This form represents a string of i consecutive blanks

```
               ('7'(3) + i:DUPCOUNT(5) + TEXTBYTE(8)) |
```

This form represents string of i consecutive duplicates of TEXTBYTE.

```
               ('2'(2) + j:LENGTH(6) + (TEXTBYTE(8) = j))
```

This form represents a string of j characters.

```
TRUNCATED  ::= '3'(2) + DEVID(6) + n:COUNT(8) + (TEXTBYTE(8)=n)

DEVID(6)   ::= DEVNO(3) + t:DEVTYPE(3)
```

DEVID identifies a particular virtual device, i.e., it identifies a channel. DEVTYPE specifies the type of device, as follows:

```
t = 1:  Output to remote operator console
    2:  Input from remote operator console
    3:  Input from card reader
    4:  Output to printer
    5:  Output to card punch
  6,7:  Unused
```

DEVNO identifies the particular device of type t at this remote site; at present only DEVNO = 0 is possible.

END-OF-DATA ::=X'FE'

Signals end of job (output) or job stack (input).

APPENDIX B

Telnet for VRBT Operator Console

The remote operator console connections use the ASCII Telnet protocol. Specifically:

1. The following one-to-one character mappings are used for the three EBCDIC graphics not in ASCII:

ASCII in Telnet	NETRJS
broken vertical bar	solid vertical bar
tilde	not sign
back slash	cent sign

2. Telnet controls are ignored.

3. An operator console input line which exceeds 133 characters (exclusive of CR LF) is truncated by NETRJS.

4. NETRJS accepts BS (Control-H) to delete a character and CAN (Control-X) to delete the current line. The sequence CR LF terminates each input and output line. HT (Control-I) is translated to a single space. An ETX (Control-C) terminates (aborts) the session. All other ASCII control characters are ignored.

5. NETRJS translates the six ASCII graphics with no equivalent in EBCDIC into the character question mark ("?") on input.

APPENDIX C

Carriage Control

The carriage control characters sent in a printer channel by NETRJS conform to IBM's extended USASI code, defined by the following table:

CODE	ACTION BEFORE WRITING RECORD
Blank	Space one line before printing
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1
2	Skip to channel 2
3	Skip to channel 3
4	Skip to channel 4
5	Skip to channel 5
6	Skip to channel 6
7	Skip to channel 7
8	Skip to channel 8
9	Skip to channel 9
A	Skip to channel 10
B	Skip to channel 11
C	Skip to channel 12

APPENDIX D

Network/RJS Command Summary

This section presents an overview of the RJS Operator Commands, for the complete form and parameter specifications please see references 2 and 3.

Terminal Control and Information Commands

SIGNON	First command of a session; identifies VRBT by giving its terminal id.
SIGNOFF	Last command of a session; RJS waits for any data transfer in progress to complete and then closes all connections.
STATUS	Outputs on the remote operator console a complete list, or a summary, of all jobs in the system for this VRBT, with an indication of their processing status in the batch host.
ALERT	Outputs on the remote operator console an "Alert" message, if any, from the computer operator. The Alert message is also automatically sent when the user does a SIGNON, or whenever the message changes.
MSG	Sends a message to the computer operator or to any other RJS terminal (real or virtual). A message from the computer operator or another RJS terminal will automatically appear on the remote operator console.

Job Control and Routing Commands

Under CCN's job management system, the default destination for output is the input source. Thus, a job submitted under a given VRBT will be returned to that VRBT (i.e., the same terminal id), unless the user's JCL overrides the default destination.

RJS places print and punch output destined for a particular remote terminal into either an Active Queue or a Deferred Queue. When the user opens his print or punch output channel, RJS immediately starts sending job output from the Active Queue, and continues until this queue is empty. Job output in the Deferred Queue, on the other hand, must be called for by job name, (via a RESET command from the remote operator) before RJS will send it. The Active/Deferred choice for output from a job is determined by the

deferral status of the VRBT when the job is entered; the deferral status, which is set to the Active option when the user signs on, may be changed by the SET command.

SET — Allows the remote user to change certain properties of his VRBT for the duration of the current session;

(a) May change the default output destination to be another (real or virtual) RJS terminal or the central facility.

(b) May change the deferral status of the VRBT.

DEFER — Moves the print and punch output for a specified job or set of jobs from the Active Queue to the Deferred Queue. If the job's output is in the process of being transmitted over a channel, RJS aborts the channel and saves the current output location before moving the job to the Deferred Queue. A subsequent RESET command will return it to the Active Queue with an implied Backspace (BSP).

RESET — Moves specified job(s) from Deferred to Active Queue so they may be sent to user. A specific list of job names or all jobs can be moved with one RESET command.

ROUTE — Re-routes output of specified jobs (or all jobs) waiting in the Active and Deferred Queues for the VRBT. The new destination may be any other RJS terminal or the central facility.

ABORT — Cancels a job which was successfully submitted and awaiting execution or is currently executing.

Output Stream Control Commands

BSP (BACKSPACE) — "Backspaces" output stream within current sysout data set. Actual amount backspaced depends upon sysout blocking but is roughly equivalent to a page on the line printer.

CAN (CANCEL) — (a) On an output channel, CAN causes the rest of the output in the sysout data set currently being transmitted to be omitted. Alternatively, may omit the rest of the sysout data sets for the job currently being transmitted; however, the remaining

system and accounting messages will be sent.

(b) On an input channel, CAN causes RJS to ignore the job currently being read. However, the channel is not aborted as a result, and RJS will continue reading in jobs on the channel.

(c) CAN can delete all sysout data sets for specified job(s) waiting in Active or Deferred Queue.

RST (RESTART) (a) Restarts a specified output stream at the beginning of the current sysout data set or, optionally, at the beginning of the job.

(b) Marks as restarted specified job(s) whose transmission was earlier interrupted by system failure or user action (e.g., DEFER command or aborting the channel). When RJS transmits these jobs again it will start at the beginning of the partially transmitted sysout data set or, optionally, at the beginning of the job. This function may be applied to jobs in either the Active or the Deferred Queue; however, if the job was in the Deferred Queue then RST also moves it to the Active Queue. If the job was never transmitted, RST has no effect other than this queue movement.

REPEAT Sends additional copies of the output of specified jobs.

EAM Echoes the card reader stream back in the printer and/or punch stream.

APPENDIX E

NETRJS TERMINAL OPTIONS

When a new NETRJS virtual terminal is defined, certain options are available; these options are listed below.

1. Truncated/Compressed Data Format

   A VRBT may use either the truncated data format (default) or the compressed format for printer and punch output. See Reference 9 for discussion of the virtues of compression.

2. Automatic Coldstart Job Resubmission

   If "R" (Restart) is specified in the accounting field on the JOB card and if this option is chosen, RJS will automatically resubmit the job from the beginning if the server operating system should be "coldstarted" before all output from the job is returned. Otherwise, the job will be lost and must be resubmitted from the remote terminal in case of a coldstart.

3. Automatic Output RESTART

   With this option, transmission of printer output which is interrupted by a broken connection always starts over at the beginning. Without this option, the output is backspaced approximately one page when restarted, unless the user forces the output to start over from the beginning with a RESTART command when the printer channel is re-opened and before printing begins.

4. Password Protection

   This option allows a password to be supplied when a terminal is signed on, preventing unauthorized use of the terminal ID.

5. Suppression of Punch Separator and Large Letters.

   This option suppresses both separator cards which RJS normally puts in front of each punched output deck, and separator pages on printed output containing the job name in large block letters. These separators are an operational aid when the ouptut is directed to a real printer or punch, but generally undesirable for an ARPA user who is saving the output in a file for on-line examination.

APPENDIX F

Character Translation by CCN Server

A VRBT declares its character set for job input and output by the initial connection socket it chooses. A VRBT can have the ASCII-68, the ASCII-63, or the EBCDIC character set. The ASCII-63 character mapping was added to NETRJS at the request of users whose terminals are equipped with keyboards like those found on the model 33 Teletype.

Since CCN operates an EBCDIC machine, its NETRJS server translates ASCII input to EBCDIC and translates printer output back to ASCII. The details of this translation are described in the following.

For ASCII-68, the following rules are used:

1. There is one-to-one mapping between the three ASCII characters broken vertical bar, tilde, and back slash, which are not in EBCDIC, and the three EBCDIC characters vertical bar, not sign, and cent sign (respectively), which are not in ASCII.

2. The other six ASCII graphics not in EBCDIC are translated on input to unused EBCDIC codes, shown in the table below.

3. The ASCII control DC4 is mapped to and from the EBCDIC control TM.

4. The other EBCDIC characters not in ASCII are mapped in the printer stream into the ASCII question mark.

For ASCII-63, the same rules are used except that the ASCII-63 codes X'60' and X'7B' - X'7E' are mapped as in the following table.

EBCDIC		ASCII-68 VRBT		ASCII-63 VRBT	
vertical bar	X'4F'	vertical bar	X'7C'	open bracket	X'5B'
not sign	X'5F'	tilde	X'7E'	close bracket	X'5D'
cent sign	X'4A'	back slash	X'5C'	back slash	X'5C'
underscore	X'6D'	underscore	X'5F'	left arrow	X'5F'
.	X'71'	up arrow	X'5E'	up arrow	X'5E'
open bracket	X'AD'	open bracket	X'5B'	.	X'7C'
close bracket	X'BD'	close bracket	X'5D'	.	X'7E'
.	X'8B'	open brace	X'7B'	.	X'7B'
.	X'9B'	close brace	X'7D'	.	X'7D'
.	X'79'	accent	X'60'	.	X'60'

APPENDIX G

REFERENCES

1. "Interim NETRJS Specifications", R. T. Braden. RFC #189: NIC #7133, July 15, 1971.

This was the basic system programmer's definition document. The proposed changes mentioned on the first page of RFC #189 were never implemented, since the DTP then in vogue became obsolete.

2. "NETRJS Remote Operator Commands", R. T. Braden. NIC #7182, August 9, 1971

This document together with References 3 and 8 define the remote operator (i.e. user) command language for NETRJS, and form the basic user documentation for NETRJS at CCN.

3. "Implementation of a Remote Job Service", V. Martin and T. W. Springer. NIC #7183, July, 1971.

4. "Remote Job Entry to CCN via UCLA Sigma 7; A scenario", UCLA/CCN. NIC #7748, November 15, 1971.

This document described the first NETRJS user implementation available on a server host. This program is no longer of general interest.

5. "Using Network Remote Job Entry", E. F. Harslem. RFC #307: NIC #9258, February 24, 1972.

This document is out of date, but describes generally the Tenex NETRJS user process "RJS".

6. "EBCDIC/ASCII Mapping for Network RJS", R. T. Braden. RFC #338: NIC #9931, May 17, 1972.

The ASCII-63 mapping described here is no longer correct, but CCN's standard ASCII-68/EBCDIC mapping is described correctly. This information is accurately described in Appendix F of the current document.

7. "NETRJT--Remote Job Service Protocol for TIP's", R. T. Braden. RFC #283: NIC 38165, December 20, 1971.

This was an attempt to define an rje protocol to handle TIPs. Although NETRJT was never implemented, many of its features are incorporated in the current Network standard RJE protocol.

8. "CCN NETRJS Server Messages to Remote User", R. T. Braden. NIC #20268, November 26, 1973.

9. "FTP Data Compression", R. T. Braden. RFC #468: NIC #14742, March 8, 1973.

10. "Update on NETRJS", R. T. Braden. RFC #599: NIC #20854, December 13, 1973.

This updated reference 1, the current document combines the two.

11. "Network Remote Job Entry -- NETRJS", G. Hicks. RFC #325: NIC 9632, April 6, 1972.

12. "CCNRJS: Remote Job Entry between Tenex and UCLA-CCN", D. Crocker. NUTS Note 22, [ISI]<DOCUMENTATION>CCNRJS.DOC, March 5, 1975.

13. "Remote Job Service at UCSB", M. Krilanovich. RFC #477: NIC #14992, May 23, 1973.

Network Working Group
Request for Comments: 818

J. Postel
ISI
November 1982

The Remote User Telnet Service

This RFC is the specification of an application protocol. Any host that implements this application level service must follow this protocol.

This RFC was suggested by Mike Mulligan some months ago when he was at BBN.

In the ARPANET Host-to-Host Network Control Protocol (NCP) and in the Internet Transmission Control Protocol (TCP) well known sockets or ports are used to identify services. The general notion is that there are a few types of services that are distinct and useful enough to use the NCP or TCP demultiplexing mechanism directly.

The most common of these is the Server Telnet which generally speaking defines the network terminal access procedure for a system executive. That is, making a connection to the server Telnet port actually puts the caller in contact with the system executive, for example, the TOPS20 EXEC or the Unix Shell.

On some small hosts there may be very limited functionality and no executive. In such cases it may be useful to designate specific well known ports for specific applications.

This memo specifies that the specific service of User Telnet may be accessed (on hosts that choose to provide it) by opening a connection to port 107 (153 octal). The Telnet Protocol is to be used on the connection from the originating user to the server.

EXAMPLE: REMOTE TELNET SERVICE ON THE BBN TC68K

The TC68K is a Terminal Concentrator based on the Motorola MC68000 microprocessor. It is used at Bolt Beranek & Newman to provide access by terminals to the FiberNet, a local area network.

The custom hardware provides one network connection, sixteen RS232 terminal connections, and a programmable timer.

The software is based on the Micro-Operating System (MOS) using the IP, ICMP, TCP, and Telnet protocols. A user TC-Telnet application provides an interface to allow the user to use the network to connect to a host,

providing a network virtual terminal. A server Telnet also exists on the TC68K to serve as a front end for devices that have no awareness of the net. This is used for remote printer/plotters and computers with no network software.

The TC68Ks at BBN are distributed about several buildings. To provide an operational tool to test remote TC68Ks, the TC68K software was configured to put a user Telnet back to back with a server Telnet. An operator can open a connection to a remote TC68K and appear to be a terminal local to that unit. This verifies that the network path between the two units is operational and provides the operator with access to statistics that are kept as part of the standard user TC-Telnet application.

This solution was attractive as the only extra piece of software necessary for this was the "Pseudo Teletype" (PTY) device driver for MOS. This "device" appears as a terminal to its application, but what it is really doing is providing a character stream between two processes.

A Network Graphics Protocol

August 16, 1974

Robert F. Sproull
Xerox Palo Alto Research Center

Elaine L. Thomas
Massachusetts Institute of Technology -- Project MAC

August 16, 1974

Table of Contents

August 16, 1974

SECTION I

Guide to the Document

This report describes a network graphics protocol (NGP) developed by the Network Graphics Group, a working group of ARPA network members. We believe that the design presented here should appeal to two groups:

-- Those interested in device-independent graphics systems and graphics standards. Indeed, the philosophy behind the protocol design originates in principles of graphics system design.

-- Those interested in using graphics via any network or communication system. Although the design was developed for the ARPA network, we believe that it has far wider applicability.

There are three major parts to the report: (1) an introductory section that gives the goals of the protocol, the general philosophy that gave rise to the particular protocol design, and definitions for a number of terms used throughout the report; (2) the details of the protocol; and (3) some suggestions to aid implementation of the protocol.

The authors of this report are by no means the only contributors to the ideas presented here. The Network Graphics Group members, too numerous to mention, have all contributed. Especially useful ideas were provided by Jim Michener, Dan Cohen, Ira Cotton, William Newman, Ken Victor and Ed Taft.

August 16, 1974

SECTION II

Introduction

Protocol proposals tend to contain so much detail that the overall intent and design considerations are lost in a pile of bits. This introduction attempts to describe the high-level considerations of the graphics protocol, in preparation for the mass of detail in section III.

The aim of a graphics protocol is to allow users with various different kinds of display hardware at different sites in a network to make use of common "graphics application programs." For example, a user may want to access the BIOMOD system at Rand, the On-Line System at UCSB or the NLS system at SRI with his or her display hardware.

One approach is a collection of "special-purpose protocols." Each application program would publish a description of the protocol needed to drive the program and to view the output (e.g. the UCSB system was so documented). The prospective user would then write a program at his site to interpret the published protocol and to drive his display (probably making use of existing graphics programming facilities at his site). This might permit a convenient division of labor between the computer executing the application program and the computer driving the display (in pursuit of true resource sharing); it might be able to achieve very smooth performance despite poor network response, etc. The disadvantage of this approach is that the user must write a new program to interface his display to each different application protocol. In addition, there is no guarantee that the protocol required by the application program can actually be implemented within the user's operating system and display hardware.

Another approach is to develop one "general-purpose protocol" that tries to provide facilities that a large number of application programs could use and that a large number of user sites could interpret. Thus one user program can be used to interface to a number of application programs. The disadvantage of this approach is that the generality may preclude adequate response through the network, or that some application programs will find the general-purpose protocol too restrictive to be used at all. In addition, the design of such a protocol is not easy: attempting to provide a common device-independent framework for driving hardware of qualitatively different capabilities may be very difficult.

The protocol proposed in this document is, of course, a general-purpose protocol. However, we anticipate that special-purpose protocols will be absolutely necessary for certain applications. In these cases, the general-purpose protocol can perhaps be used as a starting point for development of special-purpose protocols. The general-purpose protocol should be used whenever possible, however, so that separate user programs need not be written for each user site.

The network protocol considered in this document is not intended to satisfy all graphics needs, for all terminals, now and in the future. It is limited to calligraphic pictures, to moderate interaction demands, and to "best effort" attempts to generate the required graphics. Certainly the impact of video technology will increase demand for variable character sets, shading, and maybe even animation techniques. These uses are clearly too new to consider in the present protocol.

II.1. *A Typical Session*

At the beginning of an interactive session, a (human) user sits down at a graphics display attached to a computer on the network (the user host, UH; see Figure 1). He uses a keyboard associated with the display to communicate with the user host and to initiate a program that is dubbed the "user program," UP. This program initially performs TELNET* functions, allowing the user to establish a connection with another host on the network where a service he desires is offered (the server host, SH). The TELNET functions can be used to log in, query system status, and so forth, and eventually to initiate a graphics application program (AP) in the server.

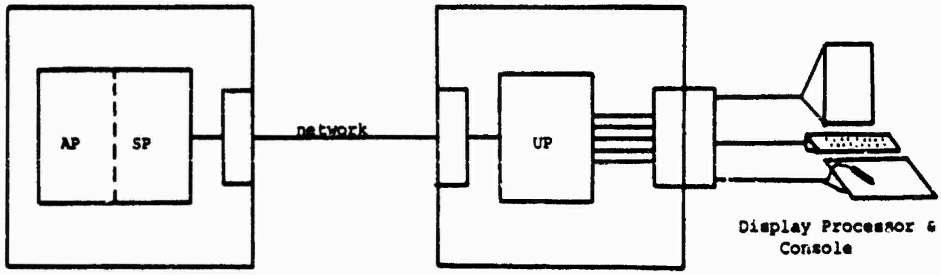


Figure 1: A network graphics session. A graphics application program running in one computer (the Server Host) drives a display console attached to another computer (the User Host).

The application program, when it wishes to produce graphical output or to request graphical input, makes use of a server program (SP) which may simply be a subroutine package. The job of the SP is to interface to the network graphics protocol on one side and to a "graphics language" on the other. (We shall use the term "graphics language" loosely here. It may just be a set of subroutine calls. The reason for making the distinction at all will appear below.)

The protocol transmitted between the SP and the UP is the graphics protocol described in this document. It provides facilities so that:

1. The SP can cause images to appear on the user's display screen.

2. The UP can report to the SP any interactions that the user initiates, such as keys depressed on the keyboard or stylus interactions.

---

*TELNET is the name of a class of programs provided by many sites on the ARPA network that allow users to log in on time-sharing or multi-access systems at other sites in the network. The term TELNET also refers to the message-transmission protocol used by the network to accomplish this function. (See L.G. Roberts and B.D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS SJCC Proceedings*, Vol. 36, May 1970, p. 543-597.)

3. The SP can discover various special properties of the user's display terminal, and can take advantage of them in conjunction with the application program.

We shall dub these types of protocol transmission "output," "input," and "inquiry."

The job of the UP is to interpret the protocol and to generate appropriate device-dependent information so that the image shown on the user's display corresponds to that specified by the SP using the protocol. It also implements the input and inquiry aspects of the protocol. The UP may have many "local options" that are not covered by the protocol. For example, the UP might contain facilities for creating hard copies of the image on the display without engaging in any protocol exchanges.

If the display terminal is attached to a TIP*, the organization changes very little. The TIP is not the "user host," but only provides communications between the UH and the display terminal itself. In this case, the UH is often dubbed the "last intelligent host." Note that nothing prevents the UH and SH from being the very same computer.

II.2. *A Model for the Network Graphics Protocol*

The analog of the task of defining a general-purpose graphics protocol is that of designing a "general-purpose interactive graphics system." This activity has been pursued by graphics system designers for years. Of course, these designs have been single-site designs; issues of device-independence and networking have been rarely addressed.

The philosophy behind the network graphics protocol can be demonstrated by giving a model of a general-purpose graphics system and combining it with the network model of Figure 1. The model of the system is shown in Figure 2. (The model is described in detail in [N&S] and [10GR]; we present only a brief description here.) To avoid misunderstandings, and for the sake of defining terminology, it is worthwhile to describe briefly each of the elements of Figure 2:

1. The input devices -- keyboard, stylus etc. -- are used by the operator of the application program to provide data and to control the program during execution.

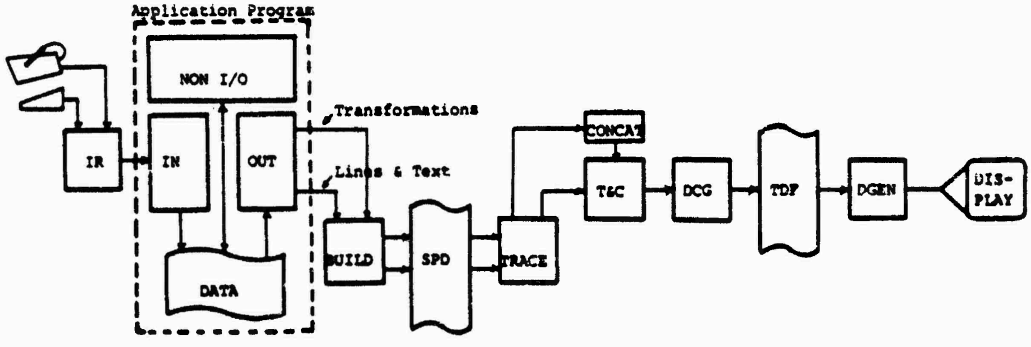
2. The application data structure contains data, basically non-graphical, relating to the application program.

3. The input routines receive data from the input devices, make appropriate changes to the application data structure, and hand control to other routines.

4. The non-I/O routines analyze and modify the application data structure.

---

*TIP is the name of a particular kind of small computer attached to the ARPA network that implements TELNET functions for a number of dial-up communications lines. The TIP performs no significant computations for any users that dial it, but simply interfaces the network TELNET protocol to a number of terminals.



Application Program:

IN	Input Routines
OUT	Output Routines
NON I/O	Application Routines
DATA	Application Data Structure

Graphics Package and Hardware:

IR	Interrupt Routines
BUILD	Routines to build the SPD
SPD	Structured Picture Definition
TRACE	Routines to trace the SPD
CONCAT	Concatenation routine
T&C	Transformation and clipping routines
DCG	Display code generator
TDF	Transformed Display File
DGEN	Display generator

Figure 2: A Conceptual Model of a Graphics Application Program, a Graphics Package and Display Console.

5. The output routines build a structured picture definition from data drawn from the application data structure. Effectively they define how this data may be visualized for display purposes.

6. The structured picture definition defines the entire picture to be displayed, part or all of which may be visible on the screen. The picture definition is generally made up of a number of elements, representing parts of the picture known collectively as subpictures; subpictures may themselves be made up of other subpictures. A familiar type of subpicture is the symbol which is often used many times within a single picture or subpicture. Each reference or call to a subpicture element may denote a transformation -- scale, rotation, etc. -- to be applied to the subpicture.

7. The transformation routines apply the transformations specified in the structured picture definition and clip out information that lies

off-screen. Often an arbitrary rectangular viewport is used as the clipping boundary instead of the screen edge. These routines also handle the concatenation of transformations necessitated by multi-level structured picture definitions.

8. The transformed display file is essentially what remains of the picture after transformation and clipping. It is defined in the screen's coordinate system and is generally stored in a format that allows direct refresh or regeneration of the picture. The display file contains "primitives" that specify lines, dots and text to be displayed.

9. The display generator generally includes a vector generator and a character generator, which transform the contents of the transformed display file into signals that the display's deflection system can understand.

10. The display itself.

We shall now analyze Figure 2 to see how the system can be implemented. The diagram illustrates three information structures (an application data structure, the structured picture definition, and the transformed display file), and three processes (the output routines, the transformations, and the display generator). The design of a general-purpose graphics system requires specifying the roles of all of these elements, with the exception of the application data structure. Our examination amounts to asking: what can the hardware do, and what does the graphics programmer think he can do? (For a more careful treatment of the concepts discussed in the next few paragraphs, see [10Gfl].)

Most display hardware is "processor" hardware, capable of implementing some or all of the three processes in Figure 2. If, for example, a display processor is available that implements transformations and display generation, then the transformed display file is absent, and we can view the hardware as "an interpreter of structured picture definitions." If the available hardware has fewer capabilities (e.g. no transformation ability), the picture is refreshed from the transformed display file, and the hardware is "an interpreter of transformed picture definitions." Or, if the hardware is a storage-tube terminal that does not require a display file for refreshing purposes, part of the display generation process is a software process. (However, a display file is still required, as we shall see below.)

Determination of what the programmer can do is somewhat more subtle, because the graphics system designer has complete control of the facilities he presents to the programmer. For example, the programmer of the system shown in Figure 2 would think he was creating a structured picture definition: the output routines allow him to create, modify, and delete elements of that structure. The remaining processes (transformation and display generation) and structures (transformed display file, if it exists) are inaccessible to the programmer; in some sense, he thinks that a "display processor" is interpreting the structured picture definition in order to generate a display. He cannot determine whether a transformed display file exists, nor whether transformations are done in hardware or software, etc.

If the structured picture definition is deleted, the programmer sees a quite different set of facilities. The output routines create (after transformation) a transformed display file. The programmer now thinks that the hardware is a "transformed display file interpreter." Again, the programmer is unaware of the

details of the display generator process (for example, if the display is a storage tube terminal, the display generator is a combination of a software display-file interpreter and some hardware vector and character generators. If, on the other hand, the transformed display file is used to refresh a display, the display generator is presumably entirely in hardware).

The discussion suggests that relative device independence can be provided if we permit two different kinds of output information: (1) information for building structured picture definitions, or (2) information for building transformed picture definitions directly. The first of these is matched to high-performance displays such as the LDS-2 or LDS-1; the second to standard refresh displays such as the IMLAC or GT-40.

How does this relate to network protocol? We can essentially view the UP as a "display processor," i.e. an "interpreter of structured picture definitions," or an "interpreter of transformed picture definitions." The network protocol is used to build and modify a picture definition contained in the user host. Various UP options are shown in Figure 3 (the dotted lines surround those processes that are implemented with display processor hardware). Figures 3a and 3b show the network used to transmit transformed information; the UP uses this information to build a display file for refreshing a display or for updating a storage-tube. Figures 3c, 3d and 3e show the network being used to transmit information for building a structured picture defintion; the UP is an interpreter of a structured display file.

Figure 4 illustrates some possibilities for the server; the server can generate either structured or transformed display information. In Figures 4a and 4b the programmer "sees" a structured picture definition; in Figure 4c a transformed picture definition.

In the protocol, the UP tells the SP which kinds of format it can implement. It is perfectly possible for the UP to implement *both* -- the display images due to each of the formats are merged onto the screen.

The two different kinds of output format can be summarized as follows:

Transformed

> The protocol is used to build and modify a set of "segments" (sometimes called "records") of a transformed display file, stored in the user host. A segment is a list of graphical primitives that specify lines, dots and text to be displayed at specific positions on the display screen. Individual segments may be deleted or replaced; they may be added to or removed from a list of segments to actually display on the screen (this is called "posting" and "unposting" segments). If a picture is composed of many segments, changes to the picture can often be made by replacing one or two segments; thus segmenting the display file helps to reduce the amount of information that must be transmitted through the network to effect a change. Considerable experience with this type of picture definition has demonstrated that device independence can easily be achieved [Omnigraph and Omnigraph.brief].
>
> One advantage of transformed format is that the UP can be kept very simple; no transformations need be performed in the UH. A UP along the lines of Figure 3a should be able to be implemented in an IMLAC. The burden of transformation is left to the SH, presumably a large computer quite capable of being programmed to do transformations.

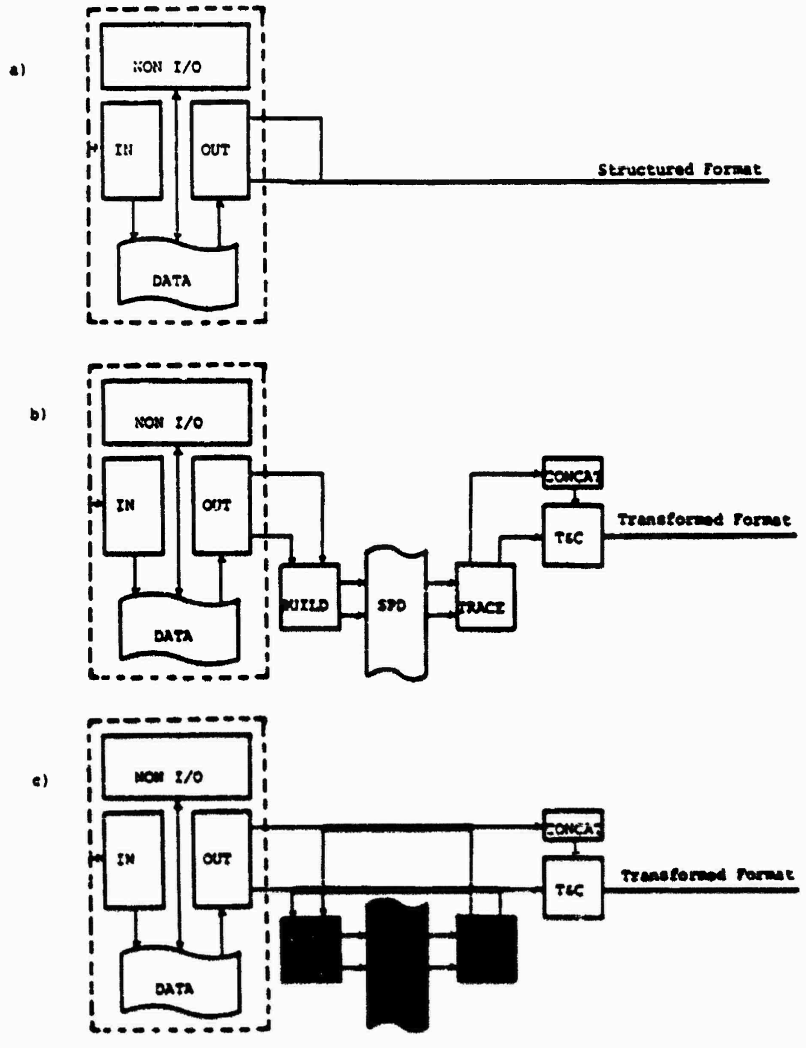


Figure 4: Various Configurations for the SP. These mate with the UP configurations of Figure 3.

single transformation (e.g. to change the viewing transformation for a three-dimensional object).

Although a structured picture definition can be interpreted directly by a few display processors that have transformation ability, most UP's that implement structured format will perform the transformation in software (Figure 3d,e; [10GR]). This implementation is relatively difficult, and certainly requires a fairly powerful UH computer.

The kind of network protocol (structured or transformed) should be clearly distinguished from the graphics language and the "output routines" of Figure 2. For example, the application data structure might be elaborately structured (e.g.

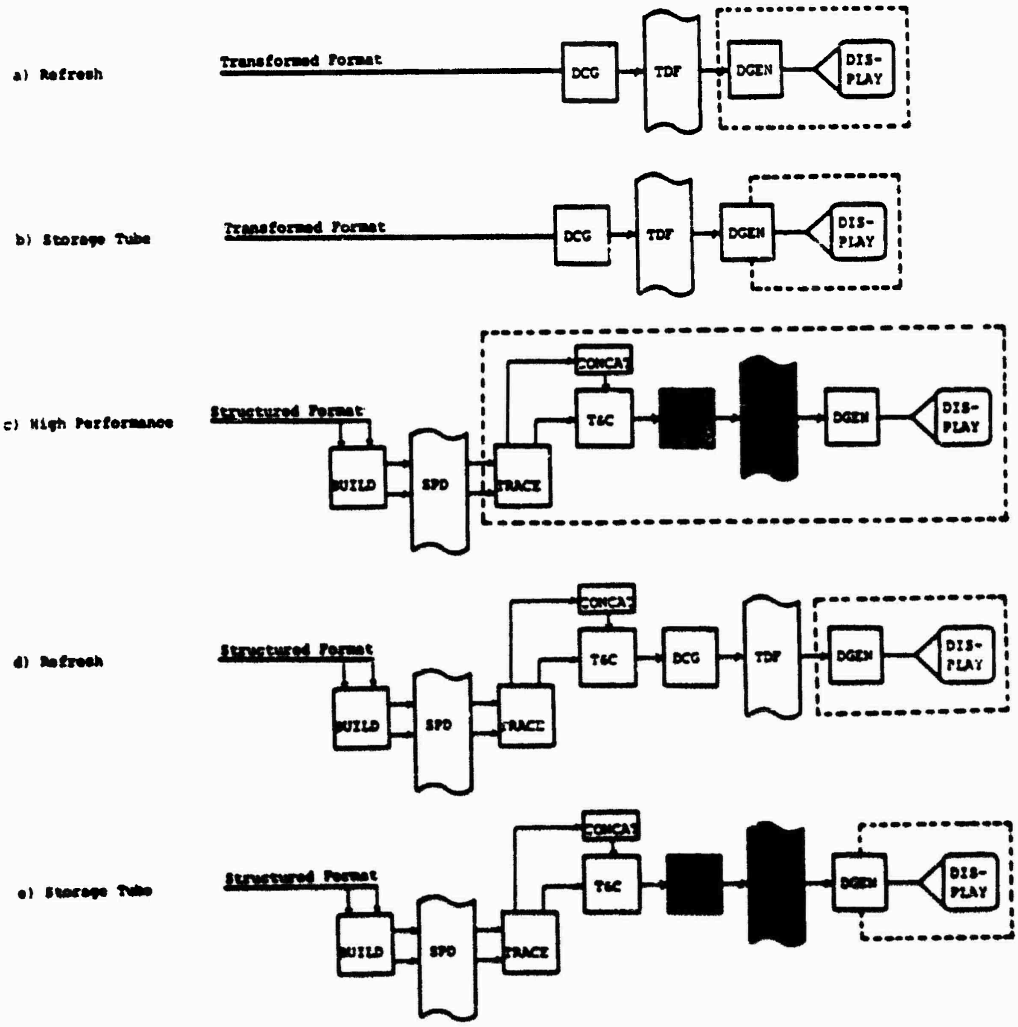


Figure 3: Various Configurations for the UP. Dashed lines surround processes implemented with hardware. Blackened processes are omitted in the particular configuration.

Structured

The protocol is used to build and modify "figures;" each figure is a collection of "units." A unit may be a list of graphical primitives, such as lines, dots and text; or it may specify a "call" on another figure, together with a transformation to apply to the called figure. The protocol can replace individual units; altering a figure that is called in several places may cause widespread changes to the visible display. The structured format requires (in principle) even less network bandwidth for updates than does the transformed format; many updates may involve changing a

a circuit diagram, with instances of flip-flops and gates, etc.), and the programmer may think of the display generation in a structured way even though the network traffic is transformed (i.e. unstructured). This effect can be achieved with the display procedure technique.

Another way of looking at this is as follows: if we view the UP as a "display processor," then we might view the SP as a "graphics package" for driving the display processor. Most graphics packages attempt to insulate the programmer from the vagaries of a particular display processor, to provide structure even though the display processor does not, etc. In other words, the application programmer may have a structured view of the world even though the UP cannot provide such a view.

We have glossed over a significant problem introduced when some of the processes of Figure 2 are implemented in software. If the programmer thinks he is producing a structured picture defintion, but the hardware is interpreting a transformed picture defintion, when is the transformation software run? A similar problem occurs in Figure 3b: when is the display-generation software run in order to update the display? One answer is: whenever a display file is changed, necessary software processes are invoked to update the display. This has several bad effects. A more useful technique is for the SP (and application program) to signal the UP whenever a collection (or batch) of changes is complete and the display should be updated. This technique has the following advantages:

1. Transformation software (e.g. 3d, 3e) is executed only when necessary in order to update the display; we never needlessly repeat transformations.

2. Screen erasures on storage tubes (e.g. 3b) can be minimized -- we need erase the screen a maximum of once per batch of updates, rather once per update. See [Omniqraph.brief].

3. All changes to the display appear "instantly." Network speed means that display-file updates will arrive at the UP over a longish period of time. If the effect of these changes is delayed until a batch is finished, the display will appear to change "all at once," a much more satisfying effect than many slow changes. This is particularly true if an image is being replaced by another image that is a scaled-up version of the original: some things grow before others, and the display passes through a number of nonsensical states.

To take full advantage of this technique, the AP should specify when the screen should be updated to represent precisely what is specified in the display file. These "end batch of updates" commands should probably precede each request for new user input -- thus the user will see an up-to-date image before formulating his response. Specifying this information is quite easily done, and is not at all unnatural for the application programmer.

If this policy of delaying changes is not to a programmer's liking, the SP could be instructed by the AP to issue an "end batch of updates" command following each and every update to a segment. Updates only occur as a result of a small number of protocol commands (see section III); this should not be difficult.

II.3. *Positioned Text*

The graphics facilities already described can be used to put text information on the screen. However, certain application programs display exclusively text (e.g. NLS [NLS]) and require a rather different set of facilities for controlling the display. Strings of text are "positioned" on a display screen and "edited" by commands from the server host.

The positioned text facilities have been separated from the graphics facilities for two reasons: (1) users with simple alphanumeric terminals (e.g. Hazeltine) can in fact implement the positioned text protocol even though they cannot implement the full graphics protocol; (2) it simplifies the design of user programs that only need to provide text interfaces (e.g. allows one to build a UP in an IMLAC that is optimized for NLS use).

This graphical output format is completely independent of the transformed and structured formats. A UP may report to the SP that it implements *only* the positioned text format -- this might be the case of a UP for NLS use.

II.4. *Input Facilities*

The problem of providing input facilities is even harder than that of providing output facilities. The difficulties are chiefly those of device independence and of adequate performance. The device independence issue is easily demonstrated: display hardware can have a large number of very different kinds of input gadgets attached (light pens, tablet and styli, joysticks, knobs, buttons, etc.) that have different properties and different methods of reporting their output (e.g. periodically, on computer demand, or "when something changes"). In addition, operating systems at user sites often enforce restrictions on the use of input equipment in order to avoid undue system degradation.

The problem of performance is nicely demonstrated by Figure 1 -- if each input must be shipped to the server host, processed by the AP and SP, then any display updates shipped to the user host and processed by the UP before the user sees the response, it would be impossible to use many interactive graphical techniques.

The protocol attacks these problems in simple and probably inadequate ways. For this reason, the input facilities are the most controversial and experimental of the protocol.

The device independence issue is solved by inquiry: the UP reports to the SP a list of available devices. The SP and AP can then collaboratively arrive at an acceptable set needed for operating the AP. If the set of input devices is insufficient, the AP can perhaps engage in a dialog with the (human) user to seek remedies. Perhaps a spare device can be plugged in; perhaps another version of the UP can be run which implements the required device. Or, if the AP and SP are sufficiently flexible, perhaps the command language of the application program can be altered dynamically to permit its operation with the available devices. For example, if the UP responds that it has no coordinate input device (and that it is not willing to simulate one with, say, two knobs) then the AP might want to use a keyboard-based interaction sequence and to organize the entire command system differently.

The performance difficulties are addressed by permitting the SP to ask the UP to

use a particular interactive technique in conjunction with an input device, and to report the results of the interaction. We shall term such interaction techniques *events*. The techniques often involve providing "local feedback," so that the user sees the results of his interaction without a long network delay. Examples are: displaying a "tracking dot" at the current location of a coordinate input device, or displaying a trail of "ink" behind the tracking dot, etc. Examples of the special *events* that the protocol provides are:

> Positioning -- Using a coordinate (or other) device to provide a pair of coordinates to specify the position for something. The UP will usually display a tracking dot to aid the user in coordinating his input with the display.
>
> Pointing -- Using a coordinate (or other) device to identify an object currently being displayed on the screen. Again, the UP will probably provide tracking. There are two ways of providing this technique: one is to assume that the display terminal has some hardware feature that aids identifying a graphical feature being pointed at (e.g. light pen or comparator). However, the same information can be deduced from a positioning interaction and some software calculation (either in the SP or UP) to determine what object is being identified. Thus, even if a UP cannot provide the "pointing" interaction, the SP may be able to (see [N&S] for a description of the process).
>
> Stroke -- Using a coordinate (or other) device to trace out a free-form curve and reporting a stream of coordinate points on the curve. The UP will provide tracking and leave a trail of dots ("ink") along the curve.
>
> Dragging -- Using a coordinate (or other) device to cause some portion of the display image to move in synchronism with the coordinate device. This technique could be classed as "highly interactive," and some display terminals cannot provide it.

The protocol provides the SP with two basic methods for dealing with input devices: (1) to request and obtain the *state* of an input device, e.g. the current position of a coordinate input device, and (2) to enable various *events*, and to obtain a "report" describing the events resulting from user actions.

The user site has a wide latitude in implementing these input facilities. Since inquiry is used to find out what devices and events the user site implements, the site may implement as many or as few as it likes. The latitude permits individual users to use devices differently or to establish special feedback mechanisms (e.g. a number displayed on the screen that represents the current reading of a knob). It also permits user sites with weird hardware or operating systems to emulate input devices in any way they choose. (For example, no requirements are put on sampling rates for inked strokes; something "reasonable and proper" is adequate.)

II.5. *Inquiry*

The protocol has no set of "standard" features; there is thus no standard graphics terminal as viewed by the protocol. The inquiry function is used to transmit to the SP a certain amount of detailed information about the terminal in use and the UP that drives it. This information is a "constant" that will probably be requested by the SP when it initiates a graphics session.

The information transmitted by the inquiry response is in part for information only. However, some of the information returned is essential in order for the SP to transmit legal protocol to the UP. In outline, the information returned is:

List of implemented protocol commands. This list tells the SP whether the UP implements transformed format, or structured format, or positioned text, or any combination of them, and so forth. In addition, this report tells which optional parts of the protocol are implemented by the UP.

Coordinate information. This information is necessary for the SP to carry out transformations that generate coordinates in the coordinate system used by the terminal.

Parameters that describe available character sizes, available intensity resolution, available line textures, etc.

A list of available input devices and events. A "device number" is specified for each device; this is used when reading the state of a device. Similarly, an "event number" is specified for each event, and is cited when enabling or disabling it.

An ASCII text string that describes the terminal, e.g. "IMLAC PDS-1 in room 22."

The information in the inquiry response (that transmitted from UP to SP) that is not essential to further protocol operation may still be useful to the SP in order to drive the terminal intelligently. For example, inquiry can determine the kind of display being used, not so as to send device-specific code to it, but so that the AP does not try to use a graphic technique on a terminal that cannot handle it (e.g. some sort of dynamics on a storage tube).

II.6. *UP Implementations*

Although the description of the protocol is quite lengthy, the protocol itself is quite simple. The aim of the design is that the UP could be implemented in an IMLAC or GT-40 or similar "smart" terminal. Of course, it could also be implemented on any host computer, with a less smart terminal attached to the host.

There are three main mechanisms for simplifying the implementation: (1) the inquiry function specifies many of the terminal details to the SP, thus freeing the UP from coping with complicated logic to implement complicated operations; (2) much of the protocol is optional, a subset being quite adequate for most applications; (3) in thorny areas (e.g., input protocol), the protocol is deliberately vague, allowing the UP implementation considerable latitude to obey the protocol as best it can.

The philosophy of "making the SP *drive* the terminal," rather than making the UP *achieve* an ideal performance is key. This approach puts ingenious graphics programming and command languages where they belong, in the server.

II.7. *Summary*

Here is a brief summary of the main philosophical points of the protocol:

0. The protocol is based on the premise that much, but not all, graphics can be done within a "general-purpose" framework. Special-purpose protocols are inevitable.

1. The protocol facilities for graphical output can be likened to those of a graphics system driving a display processor. The protocol creates and modifies a display file at the user site.

2. The protocol provides options: the SP and UP must agree on what kind of "display processor" the UP can implement (Structured, Transformed, Positioned Text, or some combination) and on selection of input devices. The protocol thus implements no fixed "virtual display" but rather a large variety of display processors.

3. Portions of the protocol are left deliberately vague. The user program is expected to implement features to the best of its ability; if the implementation is inadequate, the shortcomings will probably be quickly discovered by a user.

4. Although the protocol appears largely "device independent," inquiry functions permit the application program to discover many hardware details necessary to drive the display intelligently.

August 16, 1974

SECTION III

The Protocol

This section presents details of the graphics protocol. The topics covered are the connection protocol, the graphical output protocols, the graphical input protocol and the inquiry protocol.

The section describes network traffic as a series of commands and operands, all expressed in a common notation. The smallest unit of traffic is an 8-bit byte. The construct <...> refers to a specific byte, i.e. a command that is assigned a particular op-code (listed in the appendix). Constructs of the form <"..."> refer to sequences of bytes that are defined elsewhere in this document.

Following are some standard definitions:

All numbers in this document are decimal unless preceded by an apostrophe, in which case they are octal (8='10).

A <"small.integer"> is one 8-bit byte that contains the integer (range 0 to 255).

A <"large.integer"> is two 8-bit bytes that together comprise a 16-bit integer. The first byte transmitted is the high-order 8 bits; the second the low-order 8 bits (range 0 to '177777).

A <"small.fraction"> is a two's complement fraction in the range [-1:1-1/128]. It is defined as (<"small.integer">-128)/128.

A <"large.fraction"> is a two's complement fraction in the range [-1:1-1/32768]. It is defined as (<"large.integer">-32768)/32768.

A <"count"> is either one or two 8-bit bytes, depending on the size of the count. If the count is less than or equal to 127, then <"count"> is simply one byte that contains the count; otherwise it is two bytes, and the count is computed as (byte1-128)*256+byte2.

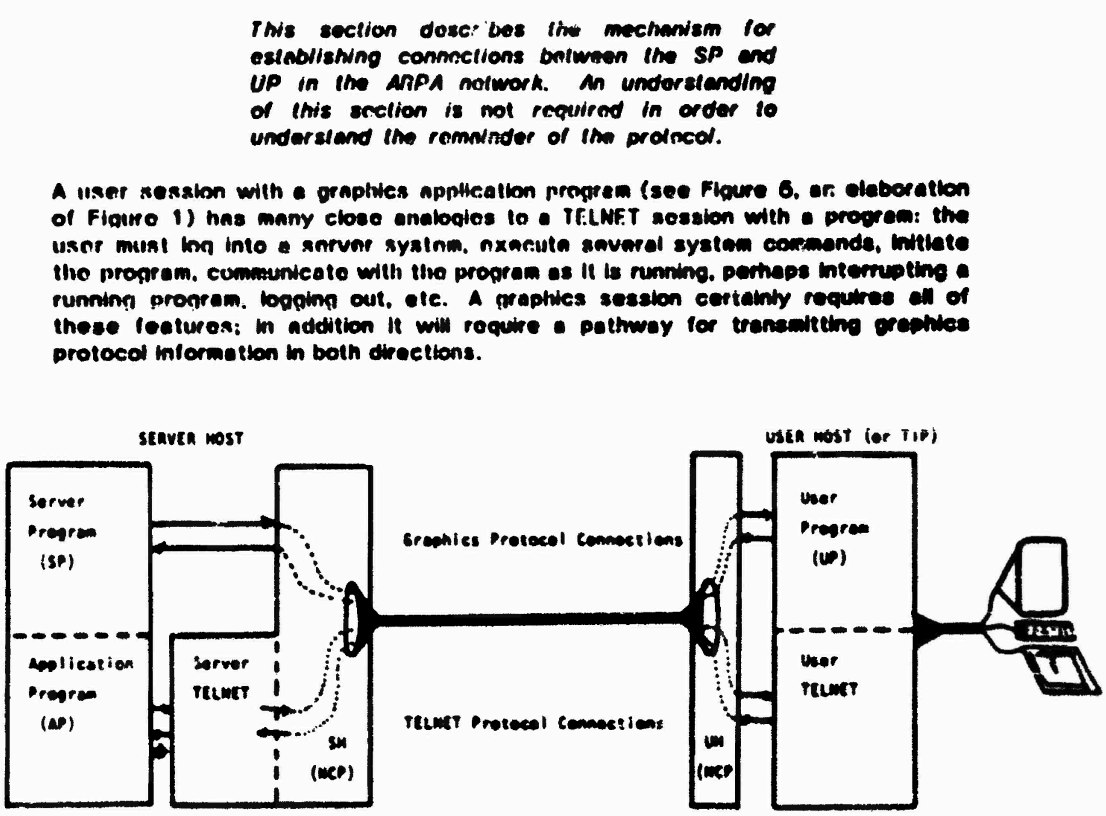
A text string <"text"> is defined as a character count, <"count">, followed by that number of 8-bit bytes. If the text string is intended to be interpreted as ASCII text, then the following conventions are observed: (1) every printing character in the ASCII set is in the set (high-order bit is zero), (2) the ASCII control characters carriage return ('15), line feed ('12), tab ('11) and formfeed ('14) may appear; (3) codes in the range '200 to '377 may be used for whatever purposes user and server desire, but the protocol establishes no conventional meanings.

The various forms of picture definitions in the user host are given names (e.g. <"seg.name">, <"picture.name">). These are all defined as 16-bit quantities, in <"large.integer"> format. The name spaces are all separate.

III.1. *Initial Connection Protocol*

This section describes the mechanism for establishing connections between the SP and UP in the ARPA network. An understanding of this section is not required in order to understand the remainder of the protocol.

A user session with a graphics application program (see Figure 6, an elaboration of Figure 1) has many close analogies to a TELNET session with a program: the user must log into a server system, execute several system commands, initiate the program, communicate with the program as it is running, perhaps interrupting a running program, logging out, etc. A graphics session certainly requires all of these features; in addition it will require a pathway for transmitting graphics protocol information in both directions.



In the case of an intelligent terminal attached to a TIP, the boxes labeled UH(NCP) and User TELNET are implemented by the TIP; the box labeled UP would be implemented in the intelligent terminal.

* This communication path is accomplished with operating-system calls that type characters on the controlling terminal and that accept characters from that terminal. (That is, this is not a direct network connection.)

Figure 5: A Network Graphics Connection and Associated Processes

The graphics protocol traffic is inherently separate from the TELNET traffic, and should use a separate network connection. It is inconvenient to multiplex graphics protocol on the TELNET connections because many operating systems give AP's access to the TELNET connection only via the mechanisms that they use to access a controlling terminal: the application program "types" to the

terminal. In addition, the TELNET connection is used by some operating systems when signaling errors; this summary use might interfere with a multiplexing scheme. Thus, the protocol requires a pair of network connections dedicated to graphics traffic.

This approach can also accomodate intelligent terminals (e.g. IMLAC's) attached to TIP's. If the IMLAC program interprets (1) ASCII text and (2) graphics protocol, and uses a simple multiplexing scheme to transmit these two kinds of traffic to and from the TIP, then the TIP can support the traffic with minor modification. The TIP need only be able to establish the extra pair of connections and to multiplex the two kinds of traffic for consumption by the intelligent terminal. (This is *not* the same as multiplexing the TELNET connection.)

There are two connection schemes, one to be used for now, and one that is intended for use when the new style TELNET protocol is fully operational and when a sufficient number of operating systems have been modified to give the UP and SP access to the negotiation mechanism.

For now,

> When a graphics SP is started (usually by the user's issuing an appropriate command to the user host through the TELNET connection), and wishes to initiate a graphics dialog, it gets a pair of complementary socket numbers from its operating system. These are called SP-send and SP-receive. The SP then "types" over its TELNET connection an ASCII string that consists of 17 characters: the first six characters are "GICP"; the remaining 11 characters are the 11 octal digits (in ASCII, of course) of the socket number for SP-receive. Note that SP-receive is an even number, and that SP-send = (SP-receive)+1. (This is a network convention that seems quite nice for now.) After typing this information, the SP does "listens" on those socket numbers.
>
> The UP recognizes the special character string and following digits, and issues RFC's (requests for connection) from two of its sockets (UP-send and UP-receive) to the complementary sockets at the SP. The SP will return [illegible] RFC's, thus completing the connections. The dust has settled, and the connections UP-send=>SP-receive and SP-send=>UP-receive are ready for use.

Ultimately,

> When reason descends on the world, the TELNET negotiation mechanism (see NIC 15372) will be used to establish the willingness to transmit graphics information (DO, DONT, WILL, WONT GRAPHICS), and a subnegotiation transmission will transmit the socket number. In the case of an intelligent graphics terminal attached to a TIP, the TIP TELNET responds to the negotiation and establishes the connections.

Even after graphics protocol has been initiated, not all communications between SP and UP will be graphics protocol; there still may be TELNET traffic. From UP to SP will go "breaks;" from SP to UP will go text "typed" by the application program (rather than enclosed in some graphical command for displaying text) as well as system error or informational messages. Such SP-to-UP text is called "unescorted text." The user will probably wish to read this text, since it is often important for understanding or operating the AP. The text can be handled by the UP in either or both of:

1. Show it on the display screen. This option is controlled by the graphics protocol (see positioned text).

2. Local (UP) option. This treatment is up to the UP: the text can be displayed regardless of provisions of method 1; it can be typed on an adjacent hard-copy terminal, or whatever.

III.2. *Output Protocol Formats*

The network graphics protocol makes provision for three kinds of formats for graphical output:

1: Transformed format.
2: Structured format.
3: Positioned text format.

It is possible to design a UP that can correctly interpret any combination of formats coming from the SP. The UP reports to the SP, via the inquiry response, which formats are implemented (this information is contained in the list of implemented commands).

The design of the structured format is still in preliminary stages. This is chiefly because of difficulties designing a suitably device-independent view of transformations (e.g., clipping and rotation conflicts, providing for the constraints of transformations performed with analog hardware). A brief description of the preliminary design appears in the appendix.

III.3. *Transformed Format*

The protocol for "transformed format" is used to build and modify a set of segments of the picture definition stored in the UH. All coordinate transformations are performed in the SP prior to sending data to the UP; the segmented picture definition thus contains descriptions of lines, dots and text that will have a fixed location on the screen.

A segment is created in the following fashion. The "open segment" command is sent to the UP, together with a "name" for the segment. Any subsequent graphical primitives (e.g. line, dot, text) sent to the UP are added, in order received, to the currently open segment. The creation process is terminated by a "close segment" command. The segment now specifies how to draw some (or all) of the desired display image.

The creation of a segment simply specifies a list of graphical primitives and not the use to which they are put. If the segment is to be displayed, a "post" command specifies that a segment is to be added to a list of segments to be displayed. The "unpost" command removes a segment from the display list. The "kill" command is used to destroy the segment altogether.

No changes to the visible display are made until the "end batch of updates" command is received at the UP. Thus, the effects of the "post," "unpost," and "kill" commands must be delayed until this command is received.

Although the graphical primitives that are contained in a segment cannot be

modified, a certain number of "attributes" associated with each segment may be individually modified. These facilities are described more fully below.

III.4. *Segment Control*

A detailed description of the commands follows:

<seg.open> <*seg.name*>

This command opens a new segment and specifies its name. All subsequent graphical primitives will be added, in order received, to the open segment. Graphical primitives need not follow contiguously -- other graphics protocol commands may intervene between specification of primitives to be added to the segment. If a segment with the same name already exists, it is not destroyed at this point (this technique is called "superceding" a segment; the protocol insists that the segment being created is double-buffered).

Immediately after the <seg.open> <*seg.name*>, any attributes that are to be associated with the new segment must be set, before the first graphical primitive is specified. This operation indicates to the UP which attributes of this segment might be changed later on. Such attribute settings are accomplished with the <*attribute*> sequence, described in section III.10. The reason for this convention is that the UP may wish to build the display file in a slightly different way if certain attributes are specified, so that they may later be changed.

<seg.close>

This commmand signals the end of generation of (or appending to) the currently open segment. The segment can now be posted, unposted, or killed.

<seg.post> <*seg.name*>

The specified segment is added to the list of segments to be displayed. If the named segment is the currently open segment, it is "closed" first. No change is made to the currently visible display.

<seg.unpost> <*seg.name*>

The specified segment is removed from the display list. Again, no change is made to the currently visible display.

<seg.kill> <*seg.name*>

The specified segment is deleted entirely. If the segment is currently in the display list, it is "unposted" first. Note that this means deletion may be delayed so as not to alter the currently visible display until the "end batch of updates" command arrives.

<seg.append> <*seg.name*>

This command specifies that all subsequent graphical primitives are to be added to the end of the segment named, which must already exist. Note,

however, that even if the segment named is currently being displayed, the appended information cannot be displayed until the next "end batch of updates" command. A segment that is appended to leaves attribute settings unchanged. In particular, the set of available attributes cannot be augmented beyond those specified originally following the <seg.open> command.

The "append" feature is entirely optional; if the UP implementation does not permit appending, the inquiry response will so specify. Failing to implement this command has no effect on the interpretation of the rest of the commands.

<end.batch.of.updates>

This command specifies that a collection of updates is complete, and the visible display should be updated to reflect the changes. In particular:

-- Any killed segments are entirely deleted and returned to free storage.

-- The old versions of any superceded segments are deleted and replaced by the new versions.

-- Any "posts" or "unposts" transmitted since the last "end batch of updates" can be performed.

-- Any "appends" specified are actually added to the appropriate segment.

(If the display file is not used to refresh a display, as is the case with a storage tube terminal, the modifications to the display file structure itself need not be delayed until the <end.batch.of.updates> command arrives, but all screen changes must be delayed. This minimizes the number of full-screen erasures required on storage tubes.)

Some application programs may wish to cause changes to appear as soon as the change has been successfully transmitted to the UH (e.g. when a <seg.post> is sent). In this case, the AP or SP can simply arrange to follow each such modification command with a <end.batch.of.updates> command.

One drawback of delaying changes is that up to twice the amount of display-file storage can be consumed, compared to that required to store one picture. This will happen if a batch of changes involves superseding every segment. This might cause the UP to exhaust the free storage available to it for display files. If this happens, the UP may simulate the effect of an <end.batch.of.updates> command prematurely, and thus reclaim storage used for superseded segments. This should really be considered an error.

A number of anomalous or "illegal" sequences of the segment-controlling commands might occur. Specific remedies are described below. (A UP implementation is not required to follow these conventions, and SP implementations should not count on them.)

1. If graphical primitives are received by the UP when no segment has been opened (either by <seg.open> or <seg.append>) they are discarded. The UP might issue an error indication.

2. If a <seg.open> or <seg.append> is received when a segment is already open, the newly-received command is ignored. Again, the UP might issue an error.

3. If the segment named by a <seg.append> does not exist, the command should be treated as a <seg.open>.

4. If the segment named in a <seg.kill>, <seg.post> or <seg.unpost> does not exist, the command is ignored.

5. <end.batch.of.updates> may occur anywhere, even while a segment is open. Primitives added to the currently-open segment should not, however, be displayed (because the segment is being created, and is not yet finished!).

6. If the <seg.kill> or <seg.unpost> commands give a name that matches that of the currently open segment, the command refers to the old version of the segment (if any), not to the one currently open.

III.5. *Graphical Primitives*

The following commands cause primitives to be added to the currently open segment:

<seg.dot> <"x.s.coord"> <"y.s.coord">
<seg.move> <"x.s.coord"> <"y.s.coord">
<seg.draw> <"x.s.coord"> <"y.s.coord">
<seg.text> <"text">

Those primitives are the familiar commands for adding points, lines and text to the open segment. No relative mode is provided: the SP can easily let the application program specify relative information, and convert to absolute for transmission.

III.6. *Coordinate Systems*

The coordinate system used for arguments to dot, move and draw in the transformed format is called the "screen coordinate system." *The format of a <" .s.coord"> construct is determined from the inquiry response, and is in the coordinate system actually used by the graphics terminal.* The inquiry response defines how many 8-bit bytes are used to specify such a coordinate, and what values correspond to the left, right, bottom and top addressable points on the screen. (For a discussion of other possibilities for the screen coordinate system, see [NIC 10933].) See section III.16, item 2 for an example of the screen coordinate calculations.

III.7. *Intensity*

The SP can select an intensity that is to be used for all subsequent graphical primitives added to segments (except as noted below). The command

<set.intensity> <"count">

specifies the intensity in <"count"> format. Permissible values of the <"count"> are returned in the inquiry response. Exception: if intensity is specified as an attribute of a segment, that value overrides any that is specified within the segment. (See the section on attributes, below. One of the reasons for declaring the intention to change attributes when opening a segment is so that the UP can generate the segment in such a way that the attribute may be implemented efficiently on the display hardware.) A default intensity (anything visible) is established by the UP initially. *

III.8. *Line Type*

The SP can govern the type of line added to segments by the draw primitive. The sequence

<set.type> <"count">

sets the line type for all subsequent lines. The inquiry function can be used to find out how many distinct types (e.g., dotted, dashed) are supported by the UP. Type 0 is always a normal solid vector, and is established as the default type by the UP initially. The protocol makes no precise definition of line types.

III.9. *Character Display*

The text primitive adds alphanumeric information to the open segment; most terminals will have hardware character generators for actually drawing the characters. However, the SP can control certain aspects of character display: size and orientation. There are two methods of specifying the size or orientation: (1) the SP may select one of several discrete character sizes and orientations available (details about these sizes are contained in the inquiry response), or (2) may select an exact size. All subsequent text primitives (up to the next time the SP asks to change) use that style.

Initially, the UP sets a default size (any legible size) and orientation (horizontal). The UP implementation should try to be resilient when the SP generates text that lies off the screen.

The orientation is selected by the sequence:

<set.character.orientation.discrete> <"count">

where <"count"> specifies one of several discrete orientations available, as specified in the inquiry response. Alternatively, the following (optional) command may be used:

---

* All that is needed to implement this facility in the UP is one variable that maintains the "current intensity." Whenever a line, dot or text string is added to an open segment, the value of the variable specifies the intensity. A similar method is applicable for line type, character size, and character orientation.

<set.character.orientation.continuous> <"large.fraction">

where the fraction is a measure of the angle between the text drawing direction and the horizontal as fractions of 2*pi. The fraction for horizontal direction is 0, for vertical text running up is 1/4, etc. If this command is implemented, the UP agrees to draw characters at any orientation.

The size is selected by one of two methods. A specific discrete size, the details of which are returned in the inquiry response, is set with

<set.character.size.discrete> <"count">

where the count is the index of the character size returned by the inquiry response.

The following command can be used to specify a "continuous" character size. This command is optional; if it is implemented, the UP agrees to display characters of any size:

<set.character.size.continuous> <"char.size.description">

where <"char.size.description"> is

<"BW.s.coord"><"BH.s.coord">
<"CW.s.coord"><"CH.s.coord"><"CT.s.coord">

The coordinates specify five dimensions, in screen coordinates, of the character, as shown in Figure 6.

III.10. *Segment Attributes*

Although the graphical primitives that make up a segment may not be modified, a small number of "attributes" may be. This section describes the attributes and the mechanisms for changing them.

Just after the <seg.open> command is transmitted, any number of <"attribute">s may be specified: these are attributes that are to be associated with the new segment. At some later time, the <change.attribute> command can be used to change any attribute that was specified in this original list. The reason for the initial list is that the UP may wish to generate the display file for a segment differently if it knows that certain attributes might later be changed.

The attributes are individually optional. If the the UP does not implement the <set.xxxx.attribute> command, then it has no facilities for dealing with the corresponding attribute

Attributes may be changed by the sequence:

<change.attribute> <"seg.name"> <"attribute">

Such changes do not take effect immediately; an <end.batch.of.updates> command causes changes.

The possible <"attribute"> sequences are:

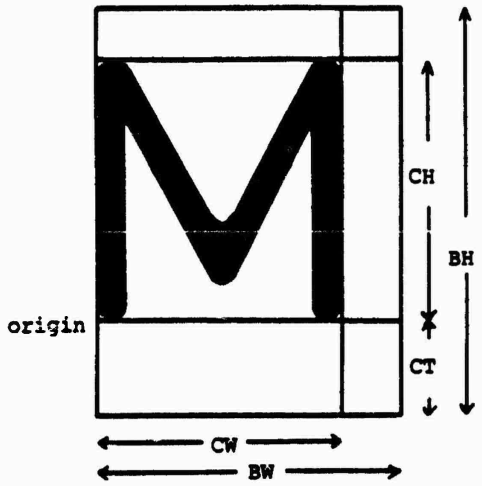


Figure 6: A Character Size Description. All measurements are made in the screen coordinate system. The point labeled "origin" is the reference point for the character, i.e. it coincides with the (x,y) point set by the <seg.move>, <seg.dot> or <seg.draw> previous to the <seg.text>.

Highlighting. This is an on/off condition for highlighting (e.g. blinking or intensifying unnaturally) an entire segment. The relevant <"attribute"> sequence is either

<set.highlight.attribute> <on> or
<set.highlight.attribute> <off>

The default is <off>.

Hit sensitivity. This on/off attribute is used in conjunction with input facilities (see III.13). If a segment is hit sensitive, then its name may be reported to the SP if a coordinate input device is pointed at any line, dot or text that is part of the segment. The <"attribute"> sequence is either

<set.hit.sensitivity.attribute> <on> or
<set.hit.sensitivity.attribute> <off>

Default is <off>.

Intensity. This attribute controls the intensity of all graphical primitives in the entire segment. It overrides any <set.intensity> settings within the segment.

<set.intensity.attribute> <"count">

The permissible values of <"count"> are returned in the inquiry response. Default is something visible.

The intensity attribute and <set.intensity> are mutually exclusive within a segment: a segment created with an intensity attribute will ignore any <set.intensity> commands; one created without an intensity attribute will honor all <set.intensity> commands.

Screen select. If a user site has several display screens, the SP can control which screens a particular segment is to be viewed on (see inquiry section for a description of how the SP can discover the number of

screens in use at the user site). Screens are enabled by a mask byte, with a bit for each of up to eight display screens (the high-order bit of the byte is the first display, the next the second, etc.). If the bit is on, 'posting' the segment will cause the segment to appear on the corresponding screen. The default is '200 (screen number 1 on).

<set.screen.select.attribute> <"small.integer">

Initial position. This command sets the position on the screen of the first item in the segment. This indicates the intention of the SP to either cause dragging to be performed, or to change the position of the segment at some future time.

<set.position.attribute> <"x.s.coord"> <"y.s.coord">

As an example: <seg.open> <seg.name=2> <set.position.attribute> <coordinate=500> <coordinate=500> <seg.move> <coordinate=400> <coordinate=500> <seg.draw> <coordinate=600> <coordinate=500> <seg.post> <seg.name=2> <end.batch.of.updates> causes a segment with one visible line to be created. The line extends 100 units left of the initial position, and 100 units to the right. If we later wish to change the position, the command <change.attribute> <seg.name=2> <set.position.attribute> <coordinate=200> <coordinate=500> will move the line 300 units to the left. Note that some UP's will wish to store segments that are to be repositioned in an internal format that is different from that for other segments (e.g., as a sequence of relative moves and vectors).

III.11. *Segment Readback*

This section describes facilities for transmitting information about a segment stored in the UP back to the SP. This allows the SP to save "copies" of segments to use later or to drive hard-copy equipment. The facilities can also be used for debugging or for certain kinds of "group sessions" in which users at several points in the network may be working concurrently and wish to "transmit" pictures to each other.

This collection of facilities is optional. The implementation should, however, be very simple, and we expect that most UP's will offer this service.

The command sent from the SP to the UP

<seg.readback.seglist>

causes the UP to transmit to the SP the sequence:

<seg.readback.seglist> <"count"> <"seg.name"> ...
<"count"> <"seg.name"> ...

The UP returns the count of posted segments, followed by their segment names, followed by the count of unposted segments, followed by their segment names.

The command from SP to UP

<seg.readback.seg> <"seg.name">

causes the UP to send back to the SP a sequence in network format that completely describes the named segment. The sequence can be any legal sequence of segment commands described in this section (including character size and type control commands) sufficient to unambiguously describe the segment (i.e., if the sequence were transmitted to the same UP, it would generate an identical display). The sequence is <seg.open> <"seg.name"> followed by any attributes of the segment, such as <set.hit.sensitivity.attribute> <on> or <set.intensity.attribute> <.5> followed by a sequence of graphical primitives that are contained in the segment.

Intensity, line type and character size commands (just like those sent from SP to UP) are imbedded in the sequence of primitivies whenever necessary in order to specify the segment exactly.

Finally, at the end of the sequence of primitives, a <seg.close> is sent if the segment was unposted, or a <seg.post> <"seg.name"> is sent if the segment was posted.

III.12. *Positioned Text*

This section describes protocol for displaying positioned text on the screen. The positioned text facilities are intended to cater for the needs of display-oriented text editors, like NLS. The SP can instruct the UP to create "text windows" on the screen: a text window is a rectangular area in which a series of "lines" of text can be displayed. Protocol commands from the SP permit characters within a line to be changed, permit lines to be moved, and the like. All changes to the text windows specified by positioned text protocol take effect immediately.

The same mechanisms for dealing with positioned text can be used to provide "teletype simulation" for TELNET-like dialog with the server host, although this is entirely up to the UP implementation.

The commands for creating and destroying text windows are

<ptext.open> <"ptext.name"> <"count"> <"x.s.coord"> <"y.s.coord">
<"x.s.coord"> <"y.s.coord"> <"count">

<ptext.kill> <"ptext.name">

The open command creates a text window, specifies a unique name for it, the number of lines it is to contain (the first <"count">), the coordinates of the lower left and upper right corners of the window in screen coordinates, and the (discrete) character size to use for all characters in the window (the second <"count">). If the name specified in the open command already exists, the old window is destroyed.

The kill command destroys the named text window. If no such window exists, the command is ignored.

A number of attributes of a text window can be set with the optional command

<ptext.set> <"ptext.name"> <"ptext.flags">

The <"ptext.flags"> byte is simply a <"small.integer"> of flag bits:

['001] Accept unescorted characters ("teletype simulation"). If this bit is on, any characters transmitted by the server host as part of TELNET protocol should be displayed in this text window. They are added "at the end," with normal ASCII conventions about format characters. Optional.

['002] Automatically scroll when text window fills up. This is probably only useful in conjunction with teletype simulation. Optional.

['004] Wrap long lines around to a new line. This too is most useful in conjunction with teletype simulation. A wrapped string counts as a new string. Optional.

['010] Make this text window visible. If the bit is off, the text window remains (and may be edited) but is not displayed. Optional.

['020] Make this text window "hit sensitive." That is, the input facilities for pointing can be applied to the window. Optional.

A default setting of <*ptext.flags*> to '010 is performed when a text window is first created.

There are several commands for changing the contents of strings in a text window. Strings within a text window are identified by a number (0 to n-1 where n is the number of strings in the window and the topmost string in the window is numbered 0). A <*string.number*> is simply the number of the string, in <*count*> format. Each string has an implicit length associated with it (internally in the UP); characters in a string are referred to by position (numbered starting at 1). Strings will normally contain standard ASCII printing characters.

The commands that modify strings are:

<ptext.scroll.up> <*ptext.name*> <*string.number*> <*string.number*>

Within a text window, scroll lines from the first string number to the second string number up one line, i.e. if the two string numbers specified are x and y, replace line x with x+1, x+1 with x+2, ... y-1 with y, and y with a null string.

<ptext.scroll.down> <*ptext.name*> <*string.number*> <*string.number*>

Similar, but scroll down.

<ptext.move> <*ptext.name*> <*string.number*> <*ptext.name*> <*string.number*>

This command causes the first string (specified by the text window name and string number) to be moved to the second string location. The first string is replaced with a null string.

<ptext.edit> <*ptext.name*> <*string.number*> <*pos1*> <*pos2*> <*text*>

This is the main command for editing strings. The arguments <*pos1*> and <*pos2*> are character positions within the specified string (in <*count*> format). The effect of the edit command is to replace the substring beginning with character pos1 and ending with character pos2-1 with the specified text string. If pos1 = pos2, this simply means that the string is inserted before the pos1 th character. There are two classes of illegal substrings that can be specified with pos1 and pos2:

pos1 > length of string. In this case, the text string is appended to the end of the present string.

pos2 > (length of string)+1. In this case, the command has the same effect as if pos2 were exactly (length of string)+1.

Note that the text specified may be a null string -- hence if pos1=1, pos2=large number (e.g. 127), and text=null, the entire specified string is replaced by the null string.

<ptext.modify> <"ptext.name"> <"string.number"> <"pos1"> <"pos2"> <"ptext.feature">

This optional command is used to select a substring that is to receive special treatment (e.g. highlighting, or blanking). It can also be used to make individual lines visible and later invisible. The values of <"ptext.feature"> are:

<ptext.highlight.on>
<ptext.highlight.off>
<ptext.visible.on>
<ptext.visible.off>

Whenever a substring is inserted with the <ptext.edit> command, the default (<ptext.highlight.off>, <ptext.visible.on>) is established.

<ptext.remote.edit> <"ptext.name"> <"string.number">

This optional command is an *ad hoc* attempt to allow UP's to implement various local line-editing regimens, without requiring interaction with the SH or SP. This command specifies a string in a positioned text window that is to be "edited" by the user. When the editing is finished, the line is returned to the SP via the TELNET connection. (This solution is unsatisfactory for several reasons, but it seems necessary to offer such a capability.)

(There have been some suggestions that we try to devise a subset of these facilities that could be implemented on some kinds of text terminal with no external character memory. If the terminal has the ability to insert and delete characters at will, and to do the scrolling functions listed above, then the only troublesome commands are the <ptext.move> command, which requires reading characters from an arbitrary line, and the <ptext.modify> command.)

III.13. *Input Facilities*

This section describes a set of input facilities for the graphics protocol. They have been kept very simple to allow simple interaction with graphics programs without tremendous complexity. Specific interactive requirements may necessitate special-purpose protocols. The input facilities are optional: many graphics application programs need only keyboard facilities provided by TELNET.

Many details concerning provision of input facilities are left to the designers of the UP. The main reason for this approach stems from vast differences among operating systems and input device hardware -- no detailed set of specifications could be met by any one site. For example, an operating system may use some

"thinning" algorithm when passing tablet coordinates to a program in the system; different operating systems will surely have different conventions. In addition, individual (human) users may prefer slightly different styles of interactions; some of the stylistic flexibility that the protocol leaves open to the UP can be exploited by the user. However, the protocol does specify the purpose of each kind of input device and event. The UP should abide by the spirit of these descriptions, although the details may vary.

The input provisions provide mechanisms for the server to:

1. Read the *state* of a device on demand.
2. Use an interactive technique, called an *event*.

III.14. *Reading the State of Input Devices*

This section describes facilities for reading the state of any input device available at the UH.

The repertoire of available input devices is reported by the UP to the SP in the inquiry response. This list includes, for each device that is available, a one-byte "index" that uniquely identifies the device. This index, referred to as an <"input.device.number">, is used by the SP to request measurement of the state of the device.

The protocol provides for the following five types of devices:

Coordinate Device

> The state of a coordinate device is a pair of coordinates in the screen coordinate system. These coordinates could be derived from a tablet and stylus, from a light pen, from two knobs, from a keyboard (by typing in values, or using a keyboard-propelled cursor) or whatever.

Linear Device

> The purpose of linear devices, such as knobs, is to provide a fraction in the range 0 to 1 to the SP. The UP may, if it wishes, provide some "echoing" of the current knob value, such as a changing number on the screen.

Key Device

> The purpose of keys (or buttons) is to provide "function button" information to the SP. A key may be a single button (on/off) or a collection, arranged to form a number of possible code combinations (e.g., NLS keyset). The state of the key is a code describing the state of the key (bit=0 for normal position, bit=1 for depressed position).

Keyboard Device

> A keyboard device provides a way of typing characters in the standard ASCII set. The state of the keyboard is the ASCII character code for the key most recently struck, or zero if the character has already been reported. (Note that the "state" of a keyboard is somewhat nonsensical -- we are more interested in the "events" caused by a keyboard. The keyboard is listed here for completeness.)

Time

If the UP provides a "time" input device, then it is willing to report to the SP a measurement of the current time. The protocol makes no tight specifications about how time is measured, except that it should be counted up once every 10 to 100 milliseconds. The UP may choose to count time somewhat inaccurately (e.g., by counting in an idle loop).

The SP can request that the state of a number of input devices be read and reported back with the optional command:

<input.report> <*count*> <*input.device.number*> ...

which specifies a list of devices whose states are to be measured as nearly simultaneously as possible and returned to the SP in the format:

<input.report> <*device.report.sequence*>

where <*device.report.sequence*> is a list:

<*count*> <*input.device.report*> ...

Each class of device has a canonical reporting format, <*input.device.report*>. The following paragraphs describe the format of <*input.device.report*>.

Keyboard Device: <*input.device.number*> <*count*>

The character code of the last key struck is returned in <*count*> format. Zero is returned if no key has been struck since the last report.

Key Device: <*input.device.number*> <*count*>

This report simply specifies the current value of the code of the key device (0 to n-1 where n is the number of states of the device).

Linear Device: <*input.device.number*> <*large.fraction*>

This simply reports the reading of the device as a fraction of its maximum excursion.

Coordinate Device: <*input.device.number*> <*x.s.coord*> <*y.s.coord*> <*sw*>

This report gives the current coordinates of the input device, and an indication of whether the "pen switch" (if it exists) is depressed (<*sw*> = <on>) or not (<*sw*> = <off>).

Time Device: <*input.device.number*> <*time*>

The time report specifies the current time. The format of <*time*> is <*large.integer*>. Note that time is recorded modulo 2^{16}

III.15. *Input Events*

The protocol provides a set of facilities for reporting to the SP the results of several kinds of input events initiated by the user. This is in contrast to the

"report-on-demand" facilities of the previous section. In particular, the protocol associates with several of the events a particular kind of interactive technique, often involving local feedback.

Keyboard Event

A keyboard event occurs when a key of a "keyboard device" is struck. The "report" for this event is the ASCII code of the key struck.

Key Event

Key events cause reports to be sent to the SP when the user manipulates a key device. A key event can be reported when the key is depressed or when it is released. (The NLS keyset is almost impossible to use unless key values are reported when a key is released.) The "report" is identical to reporting the state of a key, as described above.

Linear Event

The definition of this event, which can only be provided by "linear devices," is left to the UP. It might occur if the user changes the reading of a knob more than a certain threshold amount, or whatever.

Positioning Event

A coordinate device is used to specify a particular position or coordinate pair. For example, the user might briefly depress a tablet stylus and thus specify a position. The details of how the positioning event is caused are left up to the UP.

Pointing Event

A coordinate device is used to identify some segment, figure (structured picture definitions) or portion of positioned text that is currently displayed and that has been made "hit sensitive." The user can thus point at something on the screen. The SP expects to be told the name of the thing pointed at and the coordinates where the "hit" occurred. The UP can use a number of devices and techniques to provide these features (e.g. light pen, tablet and stylus with a hardware or software comparator). The details of when the hit is caused are left to the UP. The size of the "window" used to search for the hit is also left up to the UP (the human user may want to control this). As a local option, the UP may want to highlight in some way the segment or character that is pointed at. This identification can be removed when pointing is disabled or when the user is no longer pointing at the object (after a possible time lag). These details are up to the UP.

Stroke Event

A coordinate device is used to "draw" a stroke. The UP shows on the screen a track of ink left behind the coordinate positions, and reports to the SP the coordinates of points along the stroke. The details about when a stroke is terminated (and hence when the inking event is caused) are left to the UP. One common convention is to begin a stroke when the stylus pen switch is depressed, to continue recording points at some rate as long as the switch stays closed, and to terminate the stroke when the switch opens.

Multiple Stroke Event

This event is similar to the stroke event, but is used to record a sequence of several strokes. This is useful for on-line character recognition. The usual technique is to assume the user is finished with a collection of strokes when a period of, say, 0.5 second has elapsed since the completion of the last stroke.

Dragging Event

A coordinate device is used to move figures around on the display screen without requiring intervention from the SP. The idea is to attach a segment to the coordinates delivered from a coordinate input device, such as a tablet stylus. This interaction is not possible on many kinds of displays (e.g.storage tubes); UP's driving these terminals will not be able to provide dragging. On many refresh displays, an implementation of dragging is very simple: each segment can be defined as an absolute position, followed by relative motions (i.e. relative vectors, and relative invisible motions). In addition as the segment is recorded in the UP, we record the maximum and minimum values of x and y that the beam visits. In order to drag the segment about, we receive coordinates from the input device, check against the x and y maxima and minima to be sure that the new coordinate position will not cause any portion of the segment to go off the edge of the screen, and if not, the coordinates of the initial position instruction are replaced by the tablet coordinates (this check is only necessary for displays that cannot tolerate vectors or text that go off the screen). In order to be dragged, a segment must have been created with the <set.position.attribute> attribute specified (thus the UP can generate the segment of the display file as described above).

Pendown, Penup Events

These are special cases of positioning events which may be meaningful in certain applications. The pendown event is caused when a stylus is pressed onto a tablet surface; the penup event when it is raised.

If any of the positioning, pointing, stroke, multiple stroke, dragging, penup or pendown events are enabled, the UP should cause a tracking dot (or some form of positional feedback) to appear on the screen.

III.16. *Enabling Events*

The repertoire of input events is reported by the UP to the SP in the inquiry response. This list includes a one-byte index, the <"input.event.number">, that uniquely identifies each event implemented by the UP. If, for example, a pointing event can be caused by a light pen or by a stylus device, then two separate <"input.event.number">s are assigned, one for the corresponding event on each device.

The SP may send commands that enable and disable input events. If an event is not enabled, the UP can ignore the corresponding device (i.e. need not buffer events that occur on an un-enabled device). (As a local option, the UP may send characters typed on an unenabled keyboard through the TELNET connection as "unescorted characters.")

When the SP enables for an event, it specifies the event being enabled, the conditions under which the event will be disabled, and what is to be reported when the event occurs. The command is:

<input.enable> <"input.event.number">
<"input.disable.condition">
<"input.report.sequence">

The <"input.disable.condition"> is one of:

<never>	Event remains enabled (until further notice from SP).
<with.this.event>	Disable this event when this event occurs.
<with.any.event>	Disable this event when the next event occurs.

When an enabled event occurs, the UP sends to the SP an <"input.event.report"> that describes the results of the event that occured. In addition, the application program may desire that the *state* of various input devices be measured when the event occurs, and that these readings also be reported. When an event is enabled, the <"input.report.sequence"> specifies an ordered list of devices whose state should be measured:

<"count"> <"input.device.number"> ...

The UP should save the report list and associate it with the event that is enabled with this command. When the event occurs, the report list is used to compose a message for the SP that describes the event. (This is the mechanism used to report the time at which an event occurs.)

The dragging event requires an additional argument, the <"seg.name"> of the segment that is to be dragged. This is treated as a special case, and is set (before enabling) with the command:

<input.drag.set> <"input.event.number"> <"seg.name">

An event may be explicitly disabled by

<input.disable> <"input.event.number">

(If stroke collection is disabled, the ink is cleared. If pointing is disabled, any highlighting for the object pointed at can be cleared.)

The entire input system is cleared and all events disabled with the command

<input.reset.system>

III.17. *Event Reports*

When an input event occurs, an event report is sent from the UP to the SP. The form of an event report is:

<input.report> <"input.event.report"> <"input.report.sequence">

The <"input.report.sequence"> is defined above, and is the collection of state measurements made on various input devices when the event occurred. The <"input.event.report"> has a canonical form for each event class as follows (unless otherwise noted, the format is the same as the corresponding <"input.device.report">):

Keyboard Event: <"input.event.number"> <"count">

Key Event: <"input.event.number"> <"count">

Linear Event: <"input.event.number"> <"large.fraction">

Positioning Event: <"input.event.number"> <"x.s.coord"> <"y.s.coord">

Pendown Event: <"input.event.number"> <"x.s.coord"> <"y.s.coord">

Penup Event: <"input.event.number"> <"x.s.coord"> <"y.s.coord">

These last three reports simply specify a coordinate pair, in the screen coordinate system.

Pointing Event: <"input.event.number"> <"hit">

This report varies with different kinds of things that are hit. A <"hit"> is one of two things:

<segment.hit> <"seg.name"> <"x.s.coord"> <"y.s.coord">
<ptext.hit> <"ptext.name"> <"string.number"> <"pos1">

The first specifies the name of the entity that was pointed to. The last specifies the name of the text window, the string number and character position within the string that was identified.

Stroke Event: <"input.event.number"> <"timed"> <"stroke">

There are two ways of reporting stroke information: with and without times associated with each point. The SP requests that time be reported by including the "time" device in the <"input.report.sequence"> when enabling the stroke event. The UP will then record times if it can.

The stroke report contains an indication of whether timing information is associated with each coordinate pair. If <"timed"> is <off>, a <"stroke"> is:

<"count"> <"x.s.coord"> <"y.s.coord"> ...

where <"count"> is the number of coordinate pairs that follow. If <"timed"> is <on>, a stroke is:

<"count"> <"x.s.coord"> <"y.s.coord"> <"time"> ...

In other words, each coordinate pair has a time associated with it as well.

Multiple Stroke Event: <"input.event.number"> <"timed"> <"count"> <"stroke"> ...

This report is very similar to the stroke report, but has a provision for listing several strokes. The <"count"> is the number of strokes reported.

Dragging Event: <"input.event.number"> <"x.s.coord"> <"y.s.coord">

This report simply specifies the coordinate pair that replaces the original home (i.e. first 'move' instruction) of the dragged segment.

III.18. *Inquiry*

The UP can transmit to the SP a number of bytes that describe the terminal serviced by the UP. This information is constant (so that a UP implementation does not have to compute it each time), and is usually requested by the SP at the beginning of a graphics session.

The SP can ask for the inquiry response by transmitting to the UP the command:

<inquire>

The UP then transmits to the SP the sequence

<inquire.response> <"count"> <"response.phrase"> ...

This response includes a count of the number of response "phrases" that follow, and the response phrases, in any order. A <"response.phrase"> is

<"response.tag"> <"count"> <"response.value">

The count is the number of bytes in this particular <"response.value">. The reason for this organization is to make the inquiry responses open-ended: the SP can ignore <"response.value">s it cannot interpret.

In the description below, the tags and values for each kind of information are specified. If a default is specified, it may be assumed if the inquiry response does not include a phrase that overrides the default.

UP Features

1. What protocol commands are implemented in the UP?
   Tag: <i.implemented.commands>
   Value: up to 32 bytes of bit mask
   The i th bit of the mask is a 1 if command i is implemented (i ranges from 0 to 255). Since the <"response.phrase"> for this information contains a count of the number of bytes that comprise the response, it is not necessary to provide the entire 32 bytes in the response. In the present protocol, there are only 102 command codes assigned (0-101), so only 13 bytes are required to completely describe which commands are implemented. This information is mandatory in the response.

Terminal Features

2. What is the screen coordinate system?
   Tag: <i.screen.coordinates>
   Value: <"x.left.lv"> <"y.bottom.lv"> <"x.right.lv"> <"y.top.lv"> <"count">
   This specifies precisely the admissible values for the <"x.s.coord"> and <"y.s.coord"> sequences in the subsequent protocol. The <"count"> specifies how many bytes are used to make up a coordinate value (e.g., this would be 2 for displays that are addressed as 0 to 1023). *Thus, the number of bytes in a <" .s.coord"> is fixed by the UP.*

The four <" . .lv"> constructs that follow are examples of the screen coordinate system. Each <" . .lv"> is a 4-byte sequence that specifies a 32-bit signed two's complement number. The four constructs thus specify the left, bottom, right, and top limits of the addressing space of the terminal.

Example: An IMLAC requires two bytes per coordinate; the lower-left corner of the screen is (0,0) and the upper right is (1023,1023). The 17 bytes of response are thus:

```
'000 '000 '000 '000 ;;x left
'000 '000 '000 '000 ;;y bottom
'000 '000 '003 '377 ;;x right
'000 '000 '003 '377 ;;y top
          '002 ;;number of bytes in <" .s.coord">
```

As an example of the computation of a screen coordinate, suppose that we wish to compute the x screen coordinate of a spot that is the fraction *f* of the width of the screen (*f*=0 is at the left edge, *f*=1 at the right). Compute s = (<"x.right.lv">-<"x.left.lv">)"*f* + <"x.left.lv">. Then transmit to the UP the low-order *n* bytes of the result, where *n* is the value of <"count"> in the response.

This response is mandatory.

3. What is the screen size?
   Tag: <l.screen.size>
   Value: <"text"> <"text">
   The two text strings specify, in decimal format (e.g. 125.43), the x and y dimensions of the screen in centimeters. Note that this format is chosen so that roll plotters can work effectively: they might choose a huge screen coordinate system, and specify a long dimension as well, e.g. 1000 centimeters.

4. How many screens are there?
   Tag: <l.screen.number>
   Value: <"count">
   Default is 1.

5. What is the device name?
   Tag: <l.terminal.name>
   Value: <"text"> <"text">
   Two strings are returned: the first is some form of manufacturer's name for the terminal (e.g. "IBM 2250"). The second is a string that can uniquely identify the terminal at the user site (e.g. "Terminal in room 34."). The protocol makes no specific format requirements on these strings -- they are for information only.

6. What is the terminal type?
   Tag: <l.terminal.type>
   Value: either <storage.terminal>, <storage.with.selective.erase>, <refresh.calligraphic> or <refresh.video>

7. How many resolvable intensity values are there?
   Tag: <l.intensities>
   Value: <"count">
   If the <"count"> value is n, then the permissible values as arguments to

the intensity-setting functions are 0 (no intensity) through n-1 (maximum intensity). Default n=2.

8. How many different line types are there?
   Tag: <I.line.type>
   Value: <*count*>
   If the <*count*> value is n, then the permissible values as arguments to the type-setting functions are 0 (solid) through n-1. Default n=1.

9. What characters can be displayed on the terminal?
   Tag: <I.characters>
   Value: 32 bytes
   Each of the 128 ASCII characters has a 2-bit code for it. The codes are 00 (cannot display), 01 (can display exactly), 10 (can transliterate, e.g. lower case to upper case, or tab to spaces), 11 (this is some visible character, but not ASCII). Default: 64 character ASCII.

10. What character orientations are there?
    Tag: <I.character.orientations>
    Value: <*count*> <*large.fraction*> ...
    This response returns a list of available discrete orientations. Each fraction represents an angle (in range 0 to 2*pi) that is available (e.g. 0=normal horizontal; 1/4 is vertical running upward, etc.). If the <*count*> is n, then indices passed to the <set.character.orientation.discrete> command are in the range 0 to n-1. Default n=1, and the corresponding direction is 0.

11. What are the character sizes?
    Tag: <I.character.size>
    Value: <*count*> <*char.size.description*> ...
    If <*count*> is n, then there are n discrete character sizes available, and the arguments to <set.character.size.discrete> should range from 0 to n-1. The character size descriptions are (as nearly as possible) in order of increasing size.

12. What input devices are available?
    Tag: <I.available.input.device>
    Value: <*input.device.number*> <*input.device.description*> <*events.provided*> <*text*>
    This response phrase specifies the identity of one input device (they are broken out so that the SP can skip over individual ones whose format it cannot interpret). The <*input.device.number*> is one byte that is to be used by the SP to reference the device. The <*text*> is a text string for human consumption that describes the device. (This string might be used by the SP to engage in a dialog with the user, asking him which devices to use for what function.) The <*input.device.description*> is one of:

    <device.keyboard>
    <device.key> <*count*>
    <device.linear>
    <device.coordinate>
    <device.time> <*count*>

    The key device gives, in <*count*>, the number of states the key can assume (e.g., an NLS keyset can assume 32 states). The time device gives, in <*count*>, the approximate number of milliseconds that elapse between increments to the time counter.

Each device also lists the kinds of events it can provide, and the <"input.event.number">s used to reference the events. The reason for providing unique <"input.event.number">s for each (device,event) pair is so that more than one device can provide similar interactive techniques. The <"events.provided"> is:

<"count"> <"input.event.number"> <"event.description"> ...

The count is the number of different events this device can provide. Each event is specified by its unique index and the <"event.description">, which is one of:

<event.keyboard>
<event.key>
<event.linear>
<event.positioning>
<event.pointing>
<event.stroke>
<event.multiple.stroke>
<event.dragging>
<event.pendown>
<event.penup>

III.19. *Miscellaneous*

This section describes a collection of miscellaneous commands provided in the protocol.

The sequence

<escape.protocol> <"text">

is a way for the SP to transmit device-dependent information to the UP or from the UP to the SP. The protocol makes no provision for the format or encoding of the text string.

The command from SP to UP

<synchronize> <"count">

causes the UP to respond

<synchronize> <"count">

This provides a method of synchronizing things if absolutely necessary (e.g. a way of knowing whether a certain input event was caused before or after the display was changed with an "end batch of updates" command).

The command

<reset>

causes the UP to reset itself to a point right after the initial connection protocol has been completed and the connections opened. During early stages of debugging server and user software, this will doubtless be very useful.

Error conditions detected in the UP may be handled in several ways (the protocol makes no precise requirements):

1. Ignore them, or have the UP try to continue with as little damage as possible. Even severe errors should not crash the UP, since its operation is essential to permit the user to communicate with the SH and the application program.

2. Inform the SP of the error detected.

3. Inform the user (locally) of the error detected.

As a general strategy, the UP should probably try these in order. Certainly the UP should attempt to deal adequately with all errors (particularly such things as running out of display buffer space) so as to minimize the chances of a user losing a session's work.

Experience with common errors is probably needed before a useful error-management scheme can be devised. For this reason, we provide a mechanism for the UP to report to the SP any error conditions it detects (in free text format) and vice-versa. Thus system programmers at each end can, by saving and later examining error messages, keep track of major sources of error. The error report is

<error.string> <"text">

In some networks, it may be necessary to establish synchronization of sender and receiver of protocol. For example, if a message is lost in the network, the UP may start interpreting operand bytes as if they were command codes. Since this is not a serious problem in the ARPA network, the present protocol does not enforce a synchronization mechanism. The following scheme is believed to be adequate, and will be instituted if network reliability is a problem:

We shall define a synchronization command, called GS, that has a code different from that of any protocol command. Whenever a data byte that is equal to GS is transmitted, it must be doubled (i.e., it is transmitted as GS, GS). A single GS may precede any protocol command code. Thus, whenever a receiver encounters a single GS, it knows that the next byte is a protocol command, and not an operand. The sender may transmit a GS preceding any command byte. It may choose to transmit as many or as few of these as seem appropriate.

August 16, 1974

SECTION IV

Implementation Suggestions

The details of the protocol may seem prohibitively forbidding. This section attempts to dispel that notion and to make it all appear so simple that standard mortals can implement it.

The op-code defintions (see following section) group the commands into four groups:

Mandatory
Group 1 -- Transformed Format
Group 2 -- Positioned Text
Group 3 -- Input

Once the mandatory functions are implemented, *any* combination of the remaining three groups and the various optional commands may be implemented. For example, a UP that implements the Mandatory functions and Group 1 will be very useful indeed: many curve-plotting and mathematics packages that wish to do graphic output but are content with teletype-like input (provided by TELNET) can now be used.

An implementation of Mandatory and Group 2 would be adequate for simple page-oriented text editors; addition of Group 3 permits NLS and other more interactive systems to work. (The only combination that is probably not meaningful is just Mandatory and Group 3.)

Since many of the features of the protocol are optional, it is likely that many implementations will not include many of the options. *There is no stigma associated with omitting optional portions.*

In order to distribute information about sites that have implemented server or user programs that conform to the protocol, we would like to establish an informal "clearing-house" for such information. Those with information to give or request should address:

Robert F. Sproull
Xerox Palo Alto Research Center
3180 Porter Drive
Palo Alto, Calif. 94304

or

SPROULL@PARC-MAXC (ARPA network mail)

Especially welcome is information about implementations of the protocol that can be offered to others (e.g., if someone writes an SP facility for INTERLISP, or a UP for an IMLAC).

August 16, 1974

SECTION V

Op-Code Assignments and Options

This section assigns a number for each of the protocol command bytes described in section III, and tries to indicate what is optional and what is not.

The options column describes the conditions under which the UP should implement the command. M stands for "mandatory," and O for "optional." M-1 means mandatory if any commands in group 1 are implemented, i.e. if any M-1 command is implemented, then all M-1 commands must be. O-1 means optional if M-1 commands are implemented; otherwise not implemented.

The op-codes are grouped so that they may be decoded without requiring a full dispatch table if entire groups are unimplemented. The high-order three bits of the op code are the group number, the remaining five bits are the command within the group.

Inquiry Commands		
⟨inquire⟩	1	M
⟨inquire.response⟩	2	M
General Commands		
⟨escape.protocol⟩	3	O
⟨synchronize⟩	4	O
⟨reset⟩	5	O
⟨error.string⟩	6	O
Transformed Format Commands (Group 1)		
⟨seg.open⟩	32	M-1
⟨seg.close⟩	33	M-1
⟨seg.post⟩	34	M-1
⟨seg.unpost⟩	35	M-1
⟨seg.kill⟩	36	M-1
⟨end.batch.of.updates⟩	37	M-1
⟨seg.append⟩	38	O-1
⟨seg.dot⟩	39	M-1
⟨seg.move⟩	40	M-1
⟨seg.draw⟩	41	M-1
⟨seg.text⟩	42	M-1
⟨set.intensity⟩	43	O-1
⟨set.type⟩	44	O-1
⟨set.character.orientation.discrete⟩	45	O-1
⟨set.character.orientation.continuous⟩	46	O-1
⟨set.character.size.discrete⟩	47	O-1
⟨set.character.size.continuous⟩	48	O-1
⟨change.attribute⟩	49	O-1
⟨set.highlight.attribute⟩	50	O-1
⟨set.hit.sensitivity.attribute⟩	51	O-1
⟨set.intensity.attribute⟩	52	O-1
⟨set.screen.select.attribute⟩	53	O-1
⟨set.position.attribute⟩	54	O-1

<seg.readback.seglist>	55	O-1
<seg.readback.seg>	56	O-1
Positioned Text Commands (Group 2)		
<ptext.open>	64	M-2
<ptext.kill>	65	M-2
<ptext.set>	66	O-2
<ptext.scroll.up>	67	O-2
<ptext.scroll.down>	68	O-2
<ptext.move>	69	O-2
<ptext.edit>	70	M-2
<ptext.modify>	71	O-2
<ptext.remote.edit>	72	O-2
Input Commands (Group 3)		
<input.enable>	96	M-3
<input.disable>	97	M-3
<input.report>	98	O-3
<input.event>	99	M-3
<input.reset.system>	100	M-3
<input.drag.set>	101	O-3

Other code assignments (these are *not* commands)

<off>	0
<on>	1
Positioned Text	
<ptext.visible.off>	0
<ptext.visible.on>	1
<ptext.highlight.off>	2
<ptext.highlight.on>	3
Input facilities	
<device.keyboard>	0
<device.key>	1
<device.linear>	2
<device.coordinate>	3
<device.time>	4
<event.keyboard>	0
<event.key>	1
<event.linear>	2
<event.positioning>	5
<event.pointing>	6
<event.stroke>	7
<event.multiple.stroke>	8
<event.dragging>	9
<event.pendown>	10
<event.penup>	11
<never>	0
<with.this.event>	1
<with.any.event>	2
<segment.hit>	0
<ptext.hit>	1

August 16, 1974 Op-Code Assignments and Options

Inquiry tag definitions		
<i.implemented.commands>	1	M
<i.screen.coordinates>	2	M
<i.screen.size>	3	C
<i.screen.number>	4	O
<i.terminal.name>	5	O
<i.terminal.type>	6	O
<i.intensities>	7	O
<i.line.type>	8	O
<i.characters>	9	O
<i.character.orientations>	10	O
<i.character.size>	11	O
<i.available.input.device>	12	O
<refresh.calligraphic>	0	
<storage.terminal>	1	
<storage.with.selective.erase>	2	
<refresh.video>	3	

August 16, 1974

Appendix

Outline of Structured Format Protocol

This appendix presents a preliminary design for a structured output format protocol. It is similar to the "groups and items" technique [N&S]. It caters primarily for high-performance displays that are capable of implementing transformations in hardware and of interpreting a structured display file. However, software processes can be used by a UP to simulate these facilities if the display does not have the capability.

Display Structure

A *display structure* consists of *figures*, each containing any number of *units*. There are two types of units, *primitive* units and *call* units. Primitive units contain drawing instructions and associated coordinates that may generate visible information on the display screen. Drawing instructions and coordinates can occur only in primitive units.

Call units give the display structure a subroutine capability. A call unit invokes the display of another figure. In other words, a call unit allows one figure to contain *instances* of other figures. As well as providing for subroutine-style control transfer, call units can be used to establish the parameters to be used in the display of the subfigure. For example, a call unit can be used to call a figure with a specified intensity setting and translation. On return from the called figure, these parameters are restored to their original values.

A figure is an ordered list of units which can be any mixture of primitive and call units. Each figure begins with a *header* and terminates with the *figure end unit*. The ordering of units within a figure does not affect the display produced, but, particularly in languages such as LISP, it may be convenient to have this ordering correspond to some application data structure ordering.

In order to understand how control passes through a structure, one can think of the display elements as follows: figures are subroutines and units are linked blocks of in-line code. When all of the units contained in a figure have been executed, the figure end unit returns control to wherever the figure was called from. A primitive unit contains line and character descriptions and a transfer to the next unit. A call unit contains a subroutine call to a subfigure and a transfer to the next unit in line. Figure 7 shows a typical display structure.

Accessing Mechanisms

Figures are referenced by user-assigned names. Units may be given names at the option of the user. No two figures can have the same name, and no two units within a figure can have the same name. However, units in separate figures can have identical names, and a unit can be "named after" (i.e., carry the same name as) a figure. Figure names are called *global* and unit names *local* to reflect the fact that a unit name only distinguishes between the units within a figure.

To reference a figure, one merely refers to it by name. To reference a unit within a figure, one supplies both the figure and unit names. It is this naming mechanism which makes it possible to make incremental changes in the display.

The display structure exists at the UH, rather than at the application program.

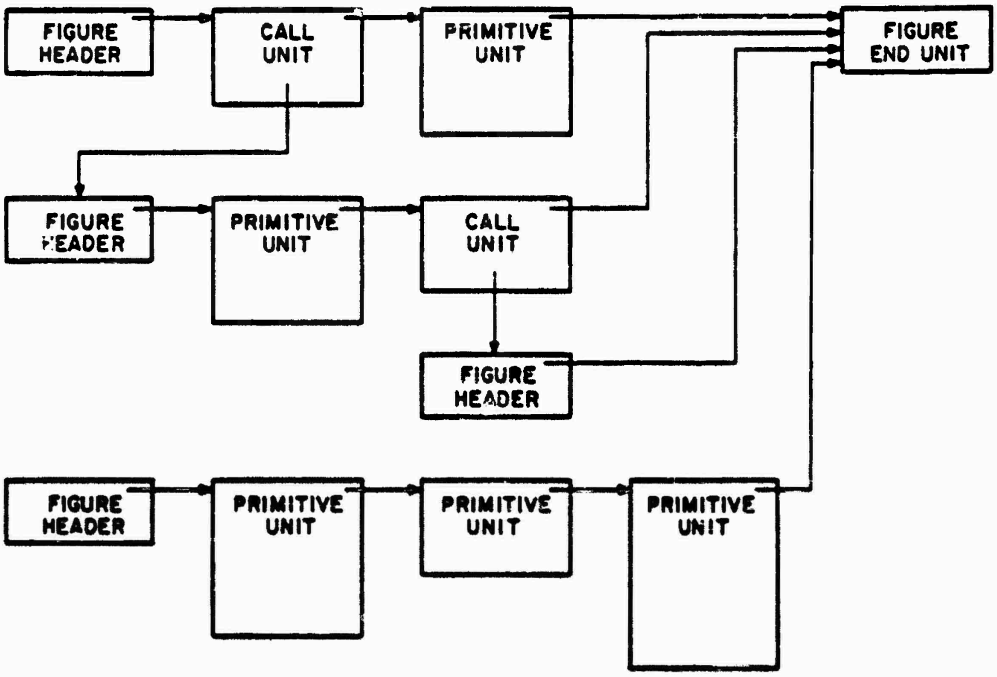


Figure 7: A Typical Display Structure

For figure and unit names to be useful, the application program should generate display names carefully to insure a one-to-one correspondence between display names and application program data structure parts.

Primitive Unit

A primitive unit can be used to draw any combination of lines and characters. More specifically, a primitive unit can consist of any number or combination of graphical primitives for drawing dots, lines and text (similar to the primitives for the transformed format, section III.6). Display coordinates are two's complement fractions of appropriate resolution centered about the point (0,0).

An option should be provided in the protocol to specify graphical primitives in a three-dimensional coordinate system.

Call Unit

Call units enable several similar picture parts to be described by the same figure. For example, the display of a circuit diagram can be constructed from a figure which is composed of the lines for a resistor and call units of this figure for each resistor in the display. All coordinate transformations and changes in intensity are specified as parameters of call units.

Master and instance rectangles are used to specify the clipping and scaling of

coordinates. A *master rectangle* is an area in the called figure which is to be mapped into an area specified by the *instance rectangle* in the calling figure (see Figure 8). Lines in the called figure which have coordinates outside the master rectangle are not displayed in the calling figure; lines in the called figure which cross the master rectangle are clipped when they are displayed in the calling figure.

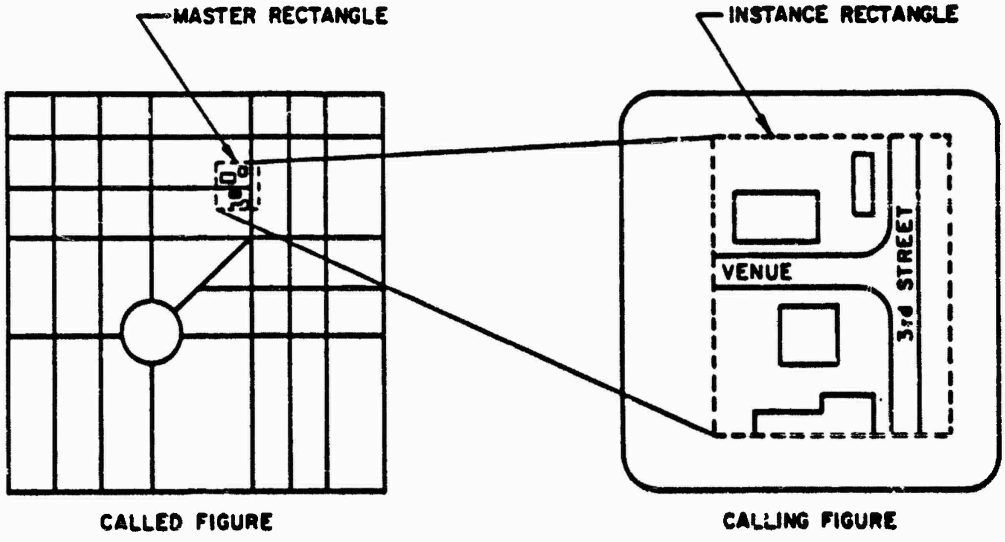


Figure 8: Clipping: Master and Instance Rectangles

Several ways of describing the master and instance rectangles of a call unit should be provided in the protocol:

1. If no master and instance rectangles are specified when creating a call unit, the coordinates of the called figure are not clipped or transformed but are displayed just as they appear in the called figure.

2. A *move call unit* has master and instance rectangles which are maximum in size and offset with respect to each other, thus "moving" all the coordinates of the called figure by a specified amount in the calling figure.

3. A *scale call unit* moves and scales the called figure with respect to the calling figure. Either the master rectangle or the instance rectangle is maximum in size, and the other rectangle is suitably smaller to achieve the desired scale.

4. Both the master and instance rectangles can be specified directly by the user. A rectangle is designated by its lower left-hand and upper right-hand corners.

A call unit also can specify an angle of rotation to be applied to the called figure. Display coordinates are rotated before they are scaled and clipped.

An option should be provided so that a call unit can specify a three-dimensional transformation. This allows displays of graphical primitives that have been specified in three dimensions. Ideally, the transformation should include the ability to specify a perspective view.

The intensity value specified in a call unit is the absolute intensity level at which the called figure is to be displayed. If no intensity is specified, the intensity remains unchanged from that of the calling figure. The default intensity of the display is the scope's brightest intensity.

Many displays have blinking line and dashed line capabilities. A call unit can be used to change the line type to dashed and/or blinking for all the lines of the called figure. If a particular scope does not have the requested capability then the line type remains unchanged (analogous to transformed format, section III.8).

Display Structure Construction and Modification

A display structure is constructed by creating its units. When a unit is created, the figure containing the unit must be specified. If the figure does not exist, it is also created. In addition, if the figure called by a call unit does not exist, it is created. Thus, new figures are created implicitly by placing units in them or by calling them from some other figure.

A unit can be inserted in the structure in one of three ways:

1. It can be inserted at the end of a figure.

2. It can be inserted after a particular unit in a figure (where the figure header is an acceptable unit after which to insert).

3. It can replace a particular unit which already exists.

If a unit with the same local name as the new unit already exists in the figure, the old unit will be deleted as a result of the creation of the new unit.

The function "display figure" causes the specified figure to be displayed. The arguments to this function are similar to those of a call unit: a master rectangle is applied to the called figure, and mapped onto a "viewport," specified in the screen coordinate system. This call unit is distinguished from all others because it alone specifies a mapping from the large two's complement coordinate system of figures and units to the screen coordinate system (which may not have a square aspect ratio). At any one time, only a single figure can be displayed. However, this is not restrictive since the figure on display can call any or all other figures. As units are added to the displayed figure, they are displayed. An empty figure is created as a result of the "display figure" function if the figure does not already exist. The function "clear scope" removes all such displayed figures from view.

The protocol should provide several mechanisms to change a display structure incrementally. Individual units and figures can be deleted from the display structure, the names of figures and units can be changed, and units can be blanked and unblanked.

Three types of deletion operations may be performed:

1. A single unit can be deleted.

2. A figure can be cleared, an operation which deletes each of the units of a figure but retains the figure header.

3. A figure can be deleted, thereby deleting all the units of the figure, the figure header, and all call units which reference the figure.

In all of the deletion operations, there is no error generated if the object to be deleted does not exist.

A unit or figure can be given a new name. If a unit is given the same name as some other unit in the same figure, the unit originally carrying the name is eliminated. If a figure name is changed to that of some other figure, the figure that already had the name is eliminated, along with any calls to it.

A blanked unit is a unit which exists in the display structure but which is marked not to be displayed. Thus, the lines and characters of a blanked primitive unit are not drawn and the figure referenced by a blanked call unit is not processed. Blanking and unblanking a unit is more efficient than deleting and recreating it.

If the transformations are being interpreted in software, it is often desirable to make several changes to the display structure before repainting the scope with the updated picture. The "end batch of updates" command is used to cause a complete update to the visible display.

Input Facilities

Two of the input facilities of the protocol take on a different meaning in conjunction with a structured display file: dragging and pointing.

For pointing, the SP can make individual figures "hit sensitive." Then, if the pointing event is enabled and the user identifies an object visible on the screen, the event report cites the global name and local name (if any) of the unit "hit."

Dragging is accomplished by identifying a move call unit whose parameters are to be changed by coordinates delivered from a coordinate input device. (For simplicity, the input device coordinates are used directly as the offset of the called figure to the calling figure. Thus, if the coordinates of the calling figure are transformed in any way, the movement of the called figure will be related to, but not directly tied to the movement of the input device.)

August 16, 1974

REFERENCES

[N&S]

W.M. Newman and R.F. Sproull, *Principles of Interactive Computer Graphics*, McGraw Hill, 1973.

[10GR]

W.M. Newman and R.F. Sproull, "An Approach to Graphics System Design," *Proceedings of the IEEE*, April 1974.

[Omnigraph.brief]

R.F. Sproull, "Omnigraph -- Simple Terminal-Independent Graphics Software," Xerox PARC Report CSL-73-4.

[Omnigraph]

"PDP-10 Display Systems," available from Computer Center Branch, Division of Computer Research and Technology, National Institutes of Health, Bethesda, Maryland 20014.

[NIC 19933]

"Proposed Network Graphics Protocol," Network Information Center 19933. (Unfortunately, this is no longer available from the NIC.)

[NLS]

D.C. Engelbart and W.K. English, "A Research Center for Augmenting Human Intellect," FJCC 1968, p. 395.

August 16, 1974

INDEX

August 16, 1974 INDEX

Network Working Group
Request for Comments: 931
Supersedes: RFC 912

Mike StJohns
TPSC
January 1985

Authentication Server

STATUS OF THIS MEMO

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. This is the second draft of this proposal (superseding RFC 912) and incorporates a more formal description of the syntax for the request and response dialog, as well as a change to specify the type of user identification returned. Distribution of this memo is unlimited.

INTRODUCTION

The Authentication Server Protocol provides a means to determine the identity of a user of a particular TCP connection. Given a TCP port number pair, it returns a character string which identifies the owner of that connection on the server's system. Suggested uses include automatic identification and verification of a user during an FTP session, additional verification of a TAC dial up user, and access verification for a generalized network file server.

OVERVIEW

This is a connection based application on TCP. A server listens for TCP connections on TCP port 113 (decimal). Once a connection is established, the server reads one line of data which specifies the connection of interest. If it exists, the system dependent user identifier of the connection of interest is sent out the connection. The service closes the connection after sending the user identifier.

RESTRICTIONS

Queries are permitted only for fully specified connections. The local/foreign host pair used to fully specify the connection are taken from the query connection. This means a user on Host A may only query the server on Host B about connections between A and B.

QUERY/RESPONSE FORMAT

The server accepts simple text query requests of the form

<local-port>, <foreign-port>

where <local-port> is the TCP port (decimal) on the target (server) system, and <foreign-port> is the TCP port (decimal) on the source (user) system.

For example:

23, 6191

The response is of the form

<local-port>, <foreign-port> : <response-type> : <additional-info>

where <local-port>,<foreign-port> are the same pair as the query, <response-type> is a keyword identifying the type of response, and <additional info> is context dependent.

For example:

23, 6191 : USERID : MULTICS : StJohns.DODCSC.a
23, 6193 : USERID : TAC : MCSJ-MITMUL
23, 6195 : ERROR : NO-USER

RESPONSE TYPES

A response can be one of two types:

USERID

In this case, <additional-info> is a string consisting of an operating system name, followed by a ":", followed by user identification string in a format peculiar to the operating system indicated. Permitted operating system names are specified in RFC-923, "Assigned Numbers" or its successors. The only other names permitted are "TAC" to specify a BBN Terminal Access Controller, and "OTHER" to specify any other operating system not yet registered with the NIC.

ERROR

For some reason the owner of <TCP-port> could not be determined, <additional-info> tells why. The following are suggested values of <additional-info> and their meanings.

INVALID-PORT

Either the local or foreign port was improperly specified.

NO-USER

The connection specified by the port pair is not currently in use.

UNKNOWN-ERROR

Can't determine connection owner; reason unknown. Other values may be specified as necessary.

CAVEATS

Unfortunately, the trustworthiness of the various host systems that might implement an authentication server will vary quite a bit. It is up to the various applications that will use the server to determine the amount of trust they will place in the returned information. It may be appropriate in some cases restrict the use of the server to within a locally controlled subnet.

APPLICATIONS

1) Automatic user authentication for FTP

A user-FTP may send a USER command with no argument to the server-FTP to request automatic authentication. The server-FTP will reply with a 230 (user logged in) if it can use the authentication. It will reply with a 530 (not logged in) if it cannot authenticate the user. It will reply with a 500 or 501 (syntax or parameter problem) if it does not implement automatic authentication. Please note that no change is needed to currently implemented servers to handle the request for authentication; they will reject it normally as a parameter problem. This is a suggested implementation for experimental use only.

2) Verification for privileged network operations. For example, having the server start or stop special purpose servers.

3) Elimination of "double login" for TAC and other TELNET users.

This will be implemented as a TELNET option.

FORMAL SYNTAX

```
<request>      ::= <port-pair> <CR> <LF>

<port-pair>    ::= <integer-number> "," <integer-number>

<reply>        ::= <reply-text> <CR> <LF>

<reply-text>   ::= <error-reply> | <auth-reply>

<error-reply>  ::= <port-pair> ":" ERROR ":" <error-type>

<auth-reply>   ::= <port-pair> ":" USERID ":" <opsys> ":" <user-id>

<error-type>   ::= INVALID-PORT | NO-USER | UNKNOWN-ERROR

<opsys>        ::= TAC | OTHER | MULTICS | UNIX ...etc.
                   (See "Assigned Numbers")
```

Notes on Syntax:

1) White space (blanks and tab characters) between tokens is not important and may be ignored.

2) White space, the token separator character (":"), and the port pair separator character (",") must be quoted if used within a token. The quote character is a back-slash, ASCII 92 (decimal) ("\"). For example, a quoted colon is "\:". The back-slash must also be quoted if its needed to represent itself ("\\").

Notes on User Identification Format:

The user identifier returned by the server should be the standard one for the system. For example, the standard Multics identifier consists of a PERSONID followed by a ".", followed by a PROJECTID, followed by a ".", followed by an INSTANCE TAG of one character. An instance tag of "a" identifies an interactive user, and instance tag of "m" identifies an absentee job (batch job) user, and an instance tag of "z" identifies a daemon (background) user.

Each set of operating system users must come to a consensus as to

what the OFFICIAL user identification for their systems will be. Until they register this information, they must use the "OTHER" tag to specify their user identification.

Notes on User Identification Translation:

Once you have a user identifier from a remote system, you must then have a way of translating it into an identifier that meaningful on the local system. The following is a sketchy outline of table driven scheme for doing this.

The table consists of four columns, the first three are used to match against, the fourth is the result.

USERID	Opsys	Address	Result
MCSJ-MITMUL	TAC	26.*.*.*	StJohns
*	MULTICS	192.5.42.*	=
*	OTHER	10.0.0.42	anonymous
MSJ	ITS	10.3.0.44	StJohns

The above table is a sample one for a Multics system on MILNET at the Pentagon. When an authentication is returned, the particular application using the userid simply looks for the first match in the table. Notice the second line. It says that any authentication coming from a Multics system on Net 192.5.42 is accepted in the same format.

Obviously, various users will have to be registered to use this facility, but the registration can be done at the same time the use receives his login identity from the system.

RFC 778

DCNET Internet Clock Service
D.L. Mills, COMSAT Laboratories
18 April 1981

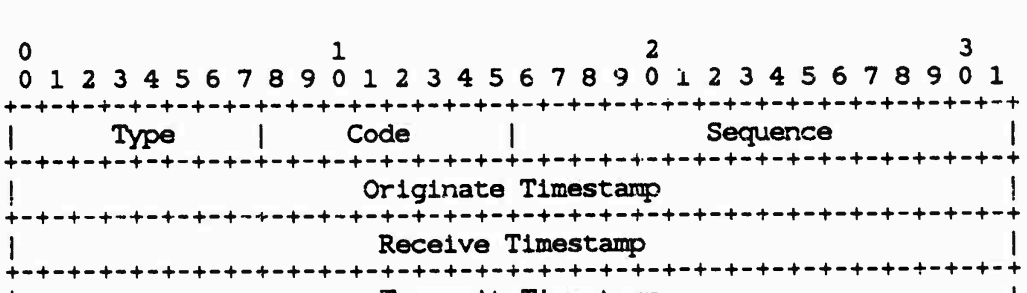
Introduction

Following is a description of the Internet Clock Service (ICS) provided by all DCNET hosts. The service, intended primarily for clock synchronization and one-way delay measurements with cooperating internet hosts, is provided using the Timestamp and Timestamp Reply messages of the proposed Internet Control Message Protocol (ICMP). In addition, in order to maintain compatability with present systems, this service will be provided for a limited time using the Echo and Echo Reply messages of the Gateway-Gateway Protocol (GGP).

It should be understood that ICMP and GGP datagrams are normally considered tightly bound to the Internet Protocol (IP) itself and not directly accessable to the user on a TOPS-20 system, for example. These datagrams are treated somewhat differently from user datagrams in gateways and DCNET hosts in that certain internal queueing mechanisms are bypassed. Thus, they can be a useful tool in providing the most accurate and stable time reference. The prime motivation for this note is to promote the development of this service in other internet hosts and gateways so that the feasibility for its use thoughout the community can be assessed.

ICS Datagrams and Timestamps

At present, the ICS is provided using either ICMP or GGP datagrams. The only difference between these is that ICMP uses protocol number 1 and GGP uses protocol number 3. In the following these will be referred to interchangably as ICS datagrams. ICS datagrams include an internet header followed by an ICS header in the following format:



ICS Datagram Format

The originator fills in all three timestamp fields just before the datagram is forwarded to the net. Each of these fields contain the local time at origination. Although the last two are redundant, they allow roundtrip delay measurements to be made using remote hosts without timestamping facilities. The "Type" field can be either 8 (GGP Echo) or 13 (ICMP Timestamp). The "Code" field should be zero. The "Sequence" field can contain either zero or an optional sequence number provided by the user. The length of the datagram is thus 36 octets inclusive of the 20-octet internet header and exclusive of the local-network leader.

The host or gateway receiving an ICS datagram fills in the "Receive Timestamp" field just as the datagram is received from the net and the "Transmit Timestamp" just as it is forwarded back to the sender. It also sets the "Type" field to 0 (GGP Echo Reply), if the original value was 8, or 14 (ICMP Timestamp Reply), if it was 13. The remaining fields are unchanged.

The timestamp values are in milliseconds from midnight UT and are stored right-justified in the 32-bit fields shown above. Ordinarily, all time calculations are performed modulo-24 hours in milliseconds. This provides a convenient match to those operating systems which maintain a system clock in ticks past midnight. The specified timestamp unit of milliseconds is consistent with the accuracy of existing radio clocks and the errors expected in the timestamping process itself.

Delay Measurements

Delay measurements can be made with any DCNET host by simply sending an ICS datagram in the above format to it and processing the reply. Let t1, t2 and t3 represent the three timestamp fields of the reply in order and t4 the time of arrival at the original sender. Then the delays, exclusive of internal processing within the DCNET host, are simply (t2 - t1) to the DCNET host, (t4 - t3) for the return and

(t2 - t1) + (t4 - t3) for the roundtrip. Note that, in the case of the roundtrip, the clock offsets between the sending host and DCNET host cancel.

Although ICS datagrams are returned by all DCNET hosts regardless of other connections that may be in use by that host at any given time, the most useful host will probably be the COMSAT-WWV virtual host at internet address [29,0,9,2], which is also the internet echo virtual host formerly called COMSAT-ECH. This virtual host is resident in the COMSAT-GAT physical host at internet address [29,0,1,2], which is connected to the ARPANET via the COMSAT Gateway, Clarksburg SIMP and a 4800-bps line to IMP 71 at BBN. The roundtrip delay via this path between the COMSAT-GAT host and the BBN Gateway is typically 550 milliseconds as the ICS datagram flies.

As in the case of all DCNET hosts, if the COMSAT-WWV virtual host is down (in this case possible only if the Spectracom radio clock is down or misbehaving) a "host not reachable" GGP datagram is returned. In unusual circumstances a "net not reachable" or "source quench" GGP datagram could be returned. Note that the references to "GGP" here will be read "ICMP" at some appropriate future time.

Local Offset Corrections

All DCNET timestamps are referenced to a designated virtual host called COMSAT-WWV (what else?) with internet address [29,0,9,2]. This host is equipped with a Spectracom radio clock which normally provides WWVB time and date to within a millisecond. The clock synchronization mechanism provides offset and drift corrections for other hosts relative to this host; however, offsets up to an appreciable fraction of a second routinely occur due to the difficulty of tracking with power-line clocks in some machines. A table of the current offsets can be obtained using the following procedure.

1. Connect to COMSAT-GAT host at internet address [29,0,1,2] using TELNET and local echo.

2. Send the command SET HOST HOST. A table with one line per DCNET host should be returned. Note the entry under the "Offset" column for the WWV host. This contains the offset in milliseconds that should be added to all timestamps generated by either the COMSAT-GAT or COMSAT-WWV hosts to yield the correct time as broadcast by WWVB.

3. Send the command SET WWV SHOW. A summary of datagram traffic is returned along with an entry labelled "NBS

time." The string following this is the last reply received from the Spectracom unit in the format:

<code> DDD HH:MM:SS TZ=00

where <code> is normally <SP> in case the WWVB signal is being received correctly or ? in case it is not. The DDD represents the day of the year and HH:MM:SS the time past UT midnight. The two digits following TZ= represent the time zone, here 00 for UT.

4. Close the connection (please!).

REFERENCES

[1] ICMP

Postel, J., "Internet Control Message Protocol", RFC 777, USC/Information Sciences Institute, April 1981.

[2] GGP

Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.

Following is a specification of the ICS header in PDP11 code:

```
;
; GGP/ICMP Header
;
.       =       0
GH.TYP:  .BLKB   1              ;Message type
GC.RPY   =       0              ;Echo reply
GC.UPD   =       1              ;Routing update
GC.ACK   =       2              ;Positive acknowledgment
GC.DNR   =       3              ;Destination unreachable
GC.SQN   =       4              ;Source quench
GC.RDR   =       5              ;Redirect
GC.ECH   =       10             ;Echo
GC.STA   =       11             ;Net interface status
GC.NAK   =       12             ;Negative acknowledgment
GC.TIM   =       15             ;Timestamp
GC.TRP   =       16             ;Timestamp Reply
GH.COD:  .BLKB   1              ;Message code
GH.SEQ:  .BLKW   1              ;Sequence number
GH.HDR   =       .              ;Beginning of original
                                ;internet header
GH.ORG:  .BLKW   2              ;Originating timestamp
GH.REC:  .BLKW   2              ;Received timestamp
GH.XMT:  .BLKW   2              ;Transmitted timestamp
GH.LEN   =       .              ;End of timestamp header
```

Note that all PDP11 word fields (.BLKW above) are "byte-swapped," that is, the order of byte transmission is the high-order byte followed by the low-order byte of the PDP11 word.

NWG/RFC# 734
SUPDUP Display Protocol

MRC 07-OCT-77 08:46 41953
Page 1

Network Working Group
Request for Comments 734
NIC 41953

Mark Crispin
SU-AI
7 October 1977

SUPDUP Protocol

INTRODUCTION

This document describes the SUPDUP protocol, a highly efficient display telnet protocol. It originally started as a private protocol between the ITS systems at MIT to allow a user at any one of these systems to use one of the others as a display. At the current writing, SUPDUP user programs also exist for Data Disc and Datamedia displays at SU-AI and for Datamedias at SRI-KL. The author is not aware of any SUPDUP servers other than at the four MIT ITS sites.

The advantage of the SUPDUP protocol over an individual terminal's protocol is that SUPDUP defines a "virtual" or "software" display terminal that implements relevant cursor motion operations. The protocol is not built on any particular display terminal but rather on the set of functions common to all display terminals; hence it is completely device-independent. In addition, the protocol also provides for terminals which cannot handle certain operations, such as line or character insert/delete. In fact, it is more than this. It provides for terminals which are missing any set of features, all the way down to model 33 Teletypes.

The advantage over the TELNET protocol is that SUPDUP takes advantage of the full capabilities of display terminals, although it also has the ability to run printing terminals.

It is to be noted that SUPDUP operates independently from TELNET; it is not an option to the TELNET protocol. In addition, certain assumptions are made about the server and the user programs and their capabilities. Specifically, it is assumed that the operating system on a server host provides all the display-oriented features of ITS. However, a server may elect not to do certain display operations available in SUPDUP; the SUPDUP protocol is far-reaching enough so that the protocol allows terminals to be handled as well as that host can handle terminals in general. Of course, if a host does not support display terminals in any special way, there is no point in bothering to implement a SUPDUP server since TELNET will work just as well.

A more complete description of the display facilities of SUPDUP and ITS can be found by FTP'ing the online file .INFO.;ITS TTY from ARPAnet host MIT-AI (host 206 octal, 134. decimal). For more information, the mailing address for SUPDUP is "(BUG SUPDUP) at MIT-AI". If your mail system won't allow you to use parentheses, use Bug-SUPDUP@MIT-AI.

BACKGROUND

The SUPDUP protocol originated as the internal protocol used between parts of ITS, and between ITS and "intelligent" terminals. Over the network, a user host acts like an intelligent terminal programmed for ITS.

The way terminal output works in ITS is as follows: The user program tells the system to do various operations, such as printing characters, clearing the screen, moving the cursor, etc. These operations are formed into 8-bit characters (using the %TD codes described below) and stored into a buffer. At interrupt level, as the terminal demands output, characters are removed from the buffer and translated into terminal dependent codes. At this time padding and cursor motion optimization are also done.

In some cases, the interrupt side does not run on the same machine as the user program. SUPDUP terminals have their "interrupt side" running in the user host. When SUPDUP is run between two ITS's, the SUPDUP user and server programs and the network simply move characters from the buffer in the server machine to the buffer in the user machine. The interrupt side then runs on the user machine just as if the characters had been generated locally.

Due to the highly interactive characteristics of both the SUPDUP protocol and the ITS system, all transactions are strictly character at a time and all echoing is remote. In addition, all padding and cursor control optimization must be done by the user.

Because this is also the internals of ITS, the right to change it any time if necessary to provide new features is reserved by MIT. In particular, the initial negotiation is probably going to be changed to transmit additional variables, and additional %TD codes may be added at any time. User programs should ignore those they don't know about.

The following conventions are used in this document: function keys (ie, keys which represent a "function" rather than a "graphic character") are in upper case in square brackets. Prefix keys (ie, keys which generate no character but rather are held down while typing another character to modify that character) are in upper case in angle brackets. Hence "<CONTROL><META>[LINE FEED]" refers to the character generated when both the CONTROL and META keys are held down while a LINE FEED is typed. Case should be noted; <CONTROL>A refers to a different character from <CONTROL>a. Finally, all numbers which do not explicitly specify a base (ie, octal or decimal) should be read as octal unless the number is immediately followed by a period, in which case it is decimal.

INITIALIZATION

The SUPDUP server listens on socket 137 octal. ICP proceeds in the normal way for establishing 8-bit connections. After the ICP is completed, the user side sends several parameters to the server side in the form of 36.-bit words. Each word is sent through the 8-bit connection as six 6-bit bytes, most-significant first. Each byte is in the low-order 6 bits of a character. The first word is the negative of the number of variables to follow in the high order 18. bits (the low-order 18. bits are ignored), followed by the values of the TCTYP, TTYOPT, TCMXV, TCMXH, and TTYROL terminal descriptor variables (these are the names they are known by at ITS sites). These variables are 36.-bit binary numbers and define the terminal characteristics for the virtual terminal at the REMOTE host.

The count is for future compatability. If more variables need to be sent in the future, the server should assume "reasonable" default values if the user does not specify them. PDP-10 fans will recognize the format of the count (ie, -count,,0) as being an AOBJN pointer. At the present writing there are five variables hence this word should be -5,,0.

The TCTYP variable defines the terminal type. It MUST be 7 (%TNSFW). Any other value is a violation of protocol.

The TTYOPT variable specifies what capabilities or options the user's terminal has. A bit being true implies that the terminal has this option. This variable also includes user options which the user may wish to alter at his or her own descretion; these options are included since they may be specified along with the terminal capabilities in the initial negotiation. See below for the relevant TTYOPT bits.

The TCMXV variable specifies the screen height in number of lines.

The TCMXH variable specifies the line width in number of characters. This value is one less than the screen width (ITS indicates line overflow by outputting an exclamation point at the end of the display line before moving to the next line). Note: the terminal must not do an automatic CRLF when a character is printed in the rightmost column. If this is unavoidable, the user SUPDUP must decrement the width it sends by one.

Note: Setting either the TCMXV or TCMXH dimension greater than 128. will work, but will have some problems as coordinates are sometimes represented in only 7 bits. The main problems occur in the SUPDUP protocol when sending the cursor position after an output reset and in ITS user programs using the display position codes ^PH and ^PV.

The TTYROL variable specifies the "glitch count" when scrolling. This is the number of lines to scroll up when scrolling is required. If zero, the terminal is not capable of scrolling. 1 is the usual value, but some terminals glitch up by more than one line when they scroll.

Following the transmission of the terminal options by the user, the server should respond with an ASCII greeting message, terminated with a %TDNOP code (%TD codes are described below). All transmissions from the server after the %TDNOP are either printing characters or virtual terminal display codes.

The user and the server now both communicate using the intelligent terminal protocol (described below) from the user and %TD codes from the server. The user has two commands in addition to these; they are escaped by sending 300 (octal). If following the escape is a 301 (octal), the server should attempt to log off the remote job (generally this is sent immediately before the user disconnects, so this logout procedure should be done regardless of the continuing integrity of the connection). If the character following the escape is a 302 (octal), all ASCII characters following up to a null (000 octal) are interpreted as "console location" which the server can handle as it pleases. No carriage return or line feed should be in the console location text. Normally this is saved away to be displayed by the "who" command when other users ask where this user is located.

TTYOPT FUNCTION BITS

The relevant TTYOPT bits for SUPDUP usage follow. The values are given in octal, with the left and right 18-bit halves separated by ",," as in the usual PDP-10 convention.

Bit name	Value	Meaning
%TOALT	200000,,0	characters 175 and 176 are converted to altmode (033) on input.
%TOERS	40000,,0	this terminal is capable of selectively erasing its screen. That is, it supports the %TDEOL, the %TDDLF, and (optionally) the %TDEOF operations. For terminals which can only do single-character erasing, see %TOOVR.
%TOMVB	10000,,0	this terminal is capable of backspacing (ie, moving the cursor backwards).
%TOSAI	4000,,0	this terminal has the Stanford/ITS extended ASCII graphics character set.
%TOOVR	1000,,0	this terminal is capable of overprinting; if two characters are displayed in the same position, they will both be visible, rather than one replacing the other. Lack of this capability but the capability to backspace (see %TOMVB) implies that the terminal can do single character erasing by overstriking with a space. This allows terminals without the %TOERS capability to have display-style "rubout processing", as this capability depends upon either %TOERS or [%TOMVB and not %TOOVR].
%TOMVU	400,,0	this terminal is capable of moving the cursor upwards.
%TOLWR	20,,0	this terminal's keyboard is capable of generating lowercase characters; this bit is mostly provided for programs which want to know this information.
%TOFCI	10,,0	this terminal's keyboard is capable of generating CONTROL and META characters as described below.
%TOLID	2,,0	this terminal is capable of doing line insert/delete operations, ie, it supports %TDILP and %TDDLP.
%TOCID	1,,0	this terminal is capable of doing character insert/delete operations, ie, it supports %TDICP and %TDDCP.

TTYOPT FUNCTION BITS (continued)

Bit name	Value	Meaning
%TPCBS	0,,40	this terminal is using the "intelligent terminal protocol". THIS BIT MUST BE ON.
%TPORS	0,,10	the server should process output resets instead of ignoring them. IT IS HIGHLY RECOMMENDED THAT THIS BIT BE ON; OTHERWISE THERE MAY BE LARGE DELAYS IN ABORTING OUTPUT.

The following bits are user option bits. They may be set or not set at the user's discretion. The bits that are labelled "normally on" are those that are normally set on when a terminal is initialized (ie, by typing [CALL] on a local terminal).

Bit name	Value	Meaning
%TOCLC	100000,,0	convert lower-case input to upper case. Many terminals have a "shift lock" key which makes this option useless. NORMALLY OFF.
%TOSAI	2000,,0	characters 001-037 should be displayed using the Stanford/ITS extended ASCII graphics character set instead of uparrow followed by 100+character. NORMALLY OFF.
%TOMOR	200,,0	the system should provide "**MORE**" processing when the cursor reaches the bottom line of the screen. **MORE** processing is described in ITS TTY. NORMALLY ON.
%TOROL	100,,0	the terminal should scroll when attempting output below the bottom line of the screen instead of wrapping around to the top. NORMALLY OFF.

INPUT -- THE INTELLIGENT TERMINAL PROTOCOL

Note: only the parts of the intelligent terminal protocol relevant to SUPDUP are discussed here. For more information, read ITS TTY.

CHARACTER SETS

There are two character sets available for use with SUPDUP; the 7-bit character set of standard ASCII, and the 12-bit character set of extended ASCII. Extended ASCII has 5 high order or "bucky" bits on input and has graphics for octal 000-037 and 177 (see the section entitled "Stanford/ITS character set" for more details). The two character sets are identical on output since the protocol specifies that the host should never send the standard ASCII formatting characters (ie, TAB, LF, VT, FF, CR) as formatting characters; the characters whose octal values are the same as these formatting characters are never output unless the user job has these characters enabled (setting %TOSAI and %TOSA1 generally does this).

Input differs dramatically between the 7-bit and 12-bit character sets. In the 7-bit character set, all characters input whose value is 037 octal or less are assumed to be (ASCII) control characters. In the 12-bit character set, there are 5 "bucky" bits which may be attached to the character. The two most important of these are CONTROL and META, which form a 9-bit character set. TOP is used to distinguish between printing graphics in the extended character set and ASCII controls. The other two are reserved and should be ignored. Since both 7-bit and 12-bit terminals are commonly in use, 0001, 0301, and 0341 are considered to be <CONTROL>A on input by most programs, while 4001 is considered to be downwards arrow.

MAPPING BETWEEN CHARACTER SETS

Many programs and hosts do not process 12-bit input. In this case, 12-bit input is folded down to 7-bit as follows: TOP and META are discarded. If CONTROL is on, then if the 7-bit part of the character specifies a lower case alphabetic it is converted to upper case; then if the 7-bit part is between 077 and 137 the 100 bit is complemented or if the 7-bit part is 040 the 040 bit is subtracted (that's right, <CONTROL>? is converted to [RUBOUT] and <CONTROL>[SPACE] is converted to [NULL]). In any case the CONTROL bit is discarded, and the remainder is treated as a 7-bit ASCII character. It should be noted that in this case downwards arrow is read by the program as standard ASCII <CONTROL>A.

Servers which expect 12-bit input and are told to use the 7-bit character set should do appropriate unfolding from the 7-bit character set to 12-bit. It is up to the individual server to decide upon the unfolding scheme. On ITS, user programs that use the 12-bit character set generally have an alternative method for 7-bit; this often takes the form of prefix characters indicating that the next character should be "controllified" or "metized", etc.

INPUT -- THE INTELLIGENT TERMINAL PROTOCOL (continued)

BUCKY BITS

Under normal circumstances, characters input from the keyboard are sent to the foreign host as is. There are two exceptions; the first occurs when an octal 034 character is to be sent; it must be quoted by being sent twice, because 034 is used as an escape character for protocol commands. The second exception occurs when %TOFCI is set and a character with non-zero bucky bits is to be sent. In this case, the character, which is in the 12-bit form:

Name	Value	Description
%TXTOP	4000	This character has the [TOP] key depressed.
%TXSEL	2000	Reserved, must be zero.
%TXSFT	1000	Reserved, must be zero.
%TXMTA	400	This character has the [META] key depressed.
%TXCTL	200	This character has the [CONTROL] key depressed.
%TXASC	177	The ASCII portion of the character

is sent as three bytes. The first byte is always 034 octal (that is why 034 must be quoted). The next byte contains the "bucky bits", ie, the %TXTOP through %TXCTL bits, shifted over 7 bits (ie, %TXTOP becomes 20) with the 100 bit on. The third byte contains the %TXASC part of the character. Hence the character <CONTROL><META>[LINE FEED] is sent as 034 103 012.

OUTPUT RESETS

The intelligent terminal protocol also is involved when a network interrupt (INR/INS) is received by the user program. The user program should increment a count of received network interrupts when this happens. It should not do any output, and if possible abort any output in progress, if this count is greater than zero (NOTE: the program MUST allow for the count to go less than zero).

Since the server no longer knows where the cursor is, it suspends all output until the user informs it of the cursor position. This also gives the server an idea of how much was thrown out in case it has to have some of the aborted output displayed at a later time. The user program does this when it receives a %TDORS from the server. When this happens it should decrement the "number of received network interrupts" count described in the previous paragraph and then send 034 followed by 020, the vertical position, and the horizontal position of where the cursor currently is located on the user's screen.

OUTPUT -- DISPLAY PROTOCOL (%TD CODES)

Display output is somewhat simpler. Codes less than 200 octal are printing characters and are displayed on the terminal (see the section describing the "Stanford/ITS character set"). Codes greater than or equal to 200 (octal) are known as "%TD codes", so called since their names begin with %TD. The %TD codes that are relevant to SUPDUP operation are listed here. Any other code received should be ignored, although a bug report might be sent to the server's maintainers. Note that the normal ASCII formatting characters (011 - 015) do NOT have their formatting sense under SUPDUP and should not occur at all unless the Stanford/ITS extended ASCII character set is in use (ie, %TOSAI is set in the TTYOPT word).

For cursor positioning operations, the top left corner is (0,0), ie, vertical position 0, horizontal position 0.

%TD code	Value	Meaning
%TDMOV	200	General cursor position code. Followed by four bytes; the first two are the "old" vertical and horizontal positions and may be ignored. The next two are the new vertical and horizontal positions. The cursor should be moved to this position. On printing consoles (non %TOMVU), the old vertical position may differ from the true vertical position; this can occur when scrolling. In this case, the user program should set its idea of the old vertical position to what the %TDMOV says and then proceed. Hence a %TDMOV with an old vpos of 20. and a new vpos of 22. should always move the "cursor" down two lines. This is used to prevent the vertical position from becoming infinite.
%TDMV1	201	An internal cursor motion code which should not be seen; but if it is, it has two argument bytes after it and should be treated the same as %TDMV0.
%TDEOF	202	Erase to end of screen. This is an optional function since many terminals do not support this. If the terminal does not support this function, it should be treated the same as %TDEOL. %TDEOF does an erase to end of line, then erases all lines lower on the screen than the cursor. The cursor does not move.
%TDEOL	203	Erase to end of line. This erases the character position the cursor is at and all positions to the right on the same line. The cursor does not move.

OUTPUT -- DISPLAY PROTOCOL (%TD CODES) (continued)

%TD code	Value	Meaning
%TDDLF	204	Clear the character position the cursor is on. The cursor does not move.
%TDCRL	207	If the cursor is not on the bottom line of the screen, move cursor to the beginning of the next line and clear that line. If the cursor is at the bottom line, scroll up.
%TDNOP	210	No-op; should be ignored.
%TDORS	214	Output reset. This code serves as a data mark for aborting output much as IAC DM does in the ordinary TELNET protocol.
%TDQOT	215	Quotes the following character. This is used when sending 8-bit codes which are not %TD codes, for instance when loading programs into an intelligent terminal. The following character should be passed through intact to the terminal.
%TDFS	216	Non-destructive forward space. The cursor moves right one position; this code will not be sent at the end of a line.
%TDMV0	217	General cursor position code. Followed by two bytes; the new vertical and horizontal positions.
%TDCLR	220	Erase the screen. Home the cursor to the top left hand corner of the screen.
%TDBEL	221	Generate an audio tone, bell, whatever.
%TDILP	223	Insert blank lines at the cursor; followed by a byte containing a count of the number of blank lines to insert. The cursor is unmoved. The line the cursor is on and all lines below it move down; lines moved off the bottom of the screen are lost.
%TDDLP	224	Delete lines at the cursor; followed by a count. The cursor is unmoved. The first line deleted is the one the cursor is on. Lines below those deleted move up. Newly-created lines at the bottom of the screen are blank.

OUTPUT -- DISPLAY PROTOCOL (%TD CODES) (continued)

%TD code	Value	Meaning
%TDICP	225	Insert blank character positions at the cursor; followed by a count. The cursor is unmoved. The character the cursor is on and all characters to the right on the current line move to the right; characters moved off the end of the line are lost.
%TDDCP	226	Delete characters at the cursor; followed by a count. The cursor is unmoved. The first character deleted is the one the cursor is on. Newly-created characters at the end of the line are blank.
%TDBOW	227	Display black characters on white screen. HIGHLY OPTIONAL.
%TDRST	230	Reset %TDBOW and such any future options.

STANFORD/ITS CHARACTER SET

This section describes the extended ASCII character set. It originated with the character set developed at SAIL but was modified for 1968 ASCII.

This character set only applies to terminals with the %TOSAI and %TOFCI bits set in its TTYOPT word. For non-%TOSAI terminals, the standard ASCII printing characters are the only available output characters. For non-%TOFCI terminals, the standard ASCII characters are the only available input characters.

PRINTING CHARACTERS

The first table describes the printing characters. For output, the 7-bit code is sent (terminal operations are performed by %TD codes). For input, the characters with values 000-037 and 177 must have the %TXTOP bit on to indicate the graphic is intended rather than a function or ASCII control.

Value	Character
4000	centered dot
4001	downward arrow
4002	alpha
4003	beta
4004	logical AND
4005	logical NOT
4006	epsilon
4007	pi
4010	lambda
4011	gamma
4012	delta
4013	uparrow
4014	plus-minus
4015	circle-plus
4016	infinity
4017	partial delta
4020	proper subset (left horseshoe)
4021	proper superset (right horseshoe)
4022	intersection (up horseshoe)
4023	union (downward horseshoe)
4024	universal quantifer
4025	existential quantifier
4026	circle-X
4027	double arrow
4030	left arrow
4031	right arrow
4032	not-equal
4033	lozenge (diamond)
4034	less-than-or-equal
4035	greater-than-or-equal
4036	equivalence
4037	logical OR
0040	first standard ASCII character (space)
..	. . .
0176	last standard ASCII character (tilde)
4177	integral

STANFORD/ITS CHARACTER SET (continued)

FUNCTION KEYS AND SPECIAL CHARACTERS

In addition, the following special characters exist for input only. These characters are function keys rather than printing characters; however, some of these characters have some format effect or graphic which they echo as; the host, not the SUPDUP program, handles any such mappings.

Value	Character	Usual echo	Usual Function
0000	[NULL]		
0010	[BACK SPACE]		text formatting
0011	[TAB]		text formatting
0012	[LINE FEED]		text formatting
0013	[VT]		text formatting
0014	[FORM]		text formatting
0015	[RETURN]		text formatting
0032	[CALL]	uparrow-Z	escape to system
0033	[ALTMODE]	lozenge or $	special activation
0037	[BACK NEXT]	uparrow-underscore	monitor command prefix
0177	[RUBOUT]		character delete
4101	[ESCAPE]		local terminal command
4102	[BREAK]		local subsystem escape
4103	[CLEAR]		
4110	[HELP]		requests a help message

BUCKY BITS

For all input characters, the following "bucky bits" may be added to the character. Their interpretation depends entirely upon the host. <TOP> is not listed here, as it has been considered part of the character in the previous tables.

<CONTROL> is different from ASCII CTRL, however, many programs may request the operating system to map such characters to the ASCII forms (with the <TOP> bit off). In this case <META> is ignored.

Value	Key
2000	Reserved
1000	Reserved
0400	<META>
0200	<CONTROL>

ACKNOWLEDGEMENTS

Richard M. Stallman (RMS@MIT-AI) and David A. Moon (Moon@MIT-MC) of the MIT-AI and MIT-MC systems staff wrote the source documentation and the wonderful ITS terminal support that made this protocol possible. It must be emphasized that this is a functional protocol which has been in operation for some years now.

In addition, Moon, Stallman, and Michael McMahon (MMcM@SRI-KL) provided many helpful comments and corrections to this document.

For further reference, the sources for the known currently existing SUPDUP user programs are available online as:

[MIT-AI]	SYSENG;SUPDUP >	for the ITS monitor,
[SU-AI]	SUPDUP.MID[NET,MRC]	for the SAIL monitor,
[SRI-KL]	<MMcM>SD.FAI	for the TOPS-20 monitor.

The source for the known currently existing SUPDUP server program is:

[MIT-AI]	SYSENG;TELSER >	for the ITS monitor.

These programs are written in the MIDAS and FAIL dialects of PDP-10 assembly language.