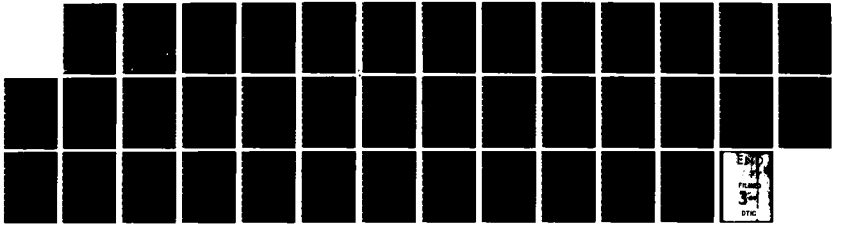
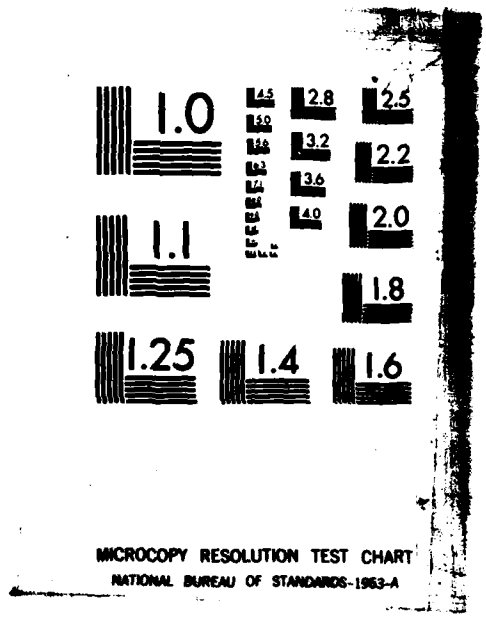
AD-A137 661 REPORT ON THE DARPA INTERNET PROJECT M/A-COM LINKABIT EASTERN OPERATIONS(U) M/A-COM LINKABIT INC MCLEAN VA D L MILLS ET AL. 30 JUN 83 MDA903-83-C-0024 1/1

UNCLASSIFIED F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

SECOND QUARTERLY PROGRESS REPORT ON THE DARPA INTERNET PROJECT

SECOND QUARTERLY PROGRESS REPORT ON THE DARPA INTERNET PROJECT

M/A-Com Linkabit
Eastern Operation

30, June 1983

MDA 903 83 C-0024

David L. Mills
Zorica Avramovic
&
Phillip G. Gross

Prepared for:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

SECOND QUARTERLY PROGRESS REPORT ON THE DARPA INTERNET PROJECT

David L. Mills, Zorica Avramovic and Phillip G. Gross
M/A-COM LINKABIT Eastern Operations

June 30, 1983

Contents

List of Figures

1. INTRODUCTION

This second Quarterly Project Report describes Linkabit's contribution to the DARPA Internet Program during the period of 1 April through 30 June 1983. Work was performed at the Eastern Operations facility in McLean, Virginia. Contributing to the effort were David L. Mills, Project Engineer and Manager, Zorica Avramovic, Senior Engineer, and Phillip Gross, Senior Engineer. Robert Enger and Walter Roehr provided technical support and assistance, while Doreen Klein provided secretarial support.

LINKABIT's efforts in support of the Internet Program are concentrated in the areas of protocol design, implementation, testing and evaluation. In addition, the group is providing integration and support services for certain computer systems to be installed at DARPA sites in Washington, D.C., and Stuttgart, West Germany.

During the period covered by this report, the level of effort was increased to three full-time equivalents with the addition of Phillip Gross, a software specialist with background in operating systems and communications. Staff responsibilities were realigned as follows: Ms. Avramovic was assigned to the mail system, including multimedia mail; and Mr. Gross to operating-system enhancements, including facsimile support. Dr. Mills remained responsible for EGP gateway design and implementation and general systems evaluation.

The level of staff involvement in computer systems and network support remained fairly high which indicates that the local-network configuration has not yet stablized completely. EGP implementation has progressed well, with test implementations converging and protocol details being finalized. Performance tests and evaluations continued with mixed results, especially with respect to FTP. In addition, several operating system enhancements were made, and bugs were fixed. Finally, LINKABIT personnel attended the ICCB, SATNET, and RG meetings at NTARE (Oslo) in July 1983.

The remainder of this report consists of four major functional parts. Section 2 summarizes specific items of progress, including the status of equipment integration, software enhancements, and testing activities; while Section 3 summarizes plans for the third quarter. Section 4 discusses of certain issues involved with TCP congestion controls. Section 5 contains the implementation plan conceived for the EGP gateway as part of the DCN Fuzzball system, while Section 6 discusses certain unresolved issues related to the EGP model itself. Finally, the appendix describes DCN local-net architecture and implementation details important for the EGP implementation, including packet

formats and peer interactions.

2. ACCOMPLISHMENTS

Major activities during this quarter included the refinement of the EGP model and the design of a test EGP implementation. The implementation was begun and initial testing was performed between LINKABIT hosts on the DCN. Additional tests were accomplished with other implementations at BBN and MIT.

2.1. EGP Effort

The EGP effort was concentrated in three areas: (1) the refinement of the EGP model itself, (2) the development of a plan for the implementation of an EGP gateway as part of the DCN and (3) the implementation of a test gateway according to this plan. The refinement of the EGP model is discussed in detail in Section 6, while the DCN implementation plan is discussed in Section 5 and the appendix. A summary of the implementation progress follows.

Although the EGP model is not considered completely stable and is expected to be developed further, many details of the protocol are not expected to change. Because of the urgency attached to this effort, a test implementation was begun to test and evaluate those aspects expected not to change. A test implementation was constructed and tested on two LINKABIT fuzzballs, one (DCN-GATEWAY) used as the ARPANET gateway, and one (DCN6) used for network testing and program development. The primary reason for choosing DCN-GATEWAY as one of the peers was that ordinary GGP routing information was available and could be "leaked" into the EGP interactions.

Initial testing between DCN-GATEWAY and DCN6 resulted in a version that executed substantially the neighbor-acquisition and neighbor-reachability protocols according to the RFC-827 specification, but with certain exceptions found useful during testing. These exceptions included:

1. Use of a two-way instead of a three-way handshake for the neighbor-acquisition protocol. The more complicated three-way handshake was judged unnecessary in view of the state information required.

2. Use of a single sequence number, rather than three as proposed.

3. Restructuring of certain packet formats, in which the polling intervals were moved to the neighbor-acquisition packets and expressed in seconds instead of minutes. In addition, the

network-reachability packet format was changed to conform more closely to the GGP update packet format in the interest of compactness.

4. Modification of the neighbor-acquisition protocol to support the concept of active and passive neighbors, so that the polling overheads could be reduced and resources conserved, especially in the core gateways.

Following initial testing on DCN hosts, tests were conducted with MIT and BBN hosts, but with no routing information being exchanged. This resulted in a general convergence in protocols and packet formats. Tests are continuing, and further work on the support for interior neighbors and polling controls awaits expected gateway memory upgrades.

2.2. Mail System Support

Work continued on the "answer" and "forward" features for the mail system, which were completed late in the previous quarter, but were tested only with other DCN hosts. During testing with various other Internet hosts, a number of compatibility problems and bugs were found and corrected. Most problems were due to lack of RFC-822 compatibility on the part of various other implementations. These problems were mostly solved by incorporating various degrees of tolerance into the parsing mechanisms.

In order to create the most demanding environment possible, a mailbox was created on the DCN5 fuzzball and incorporated into the :Header-People" distribution list on the MIT-MC host. The messages relayed by this host are known from experience to originate at virtually every implementation reachable via Internet paths and to work with various degrees of compliance with RFC-822 hosts. As expected, the fuzzball mail system quickly broke due to non-conforming messages. After intensive effort, most of the incompatibilities were detected, and permissive workarounds were installed. As implementations mature in the RFC-822 community, the need for these workarounds is expected to diminish.

Our activities in the multi-media mail area continued at a relatively low level of effort, due to priority demands of other activities. A medium-resolution color monitor was purchased and integrated with the Peritek bit-map display. The Peritek software support was enhanced to provide a bit-map capability conforming to the file formats produced by the prototype image editor. Finally, the Dacom/Rapifax digital facsimile machine was re-integrated to the DCN3 fuzzball using the DEC DUPV11 interface.

2.3. Operating System Support

A number of system enhancements were incorporated in the DCN Fuzzball system during the quarter. Some of the enhancements resulted from extended development processes, while others were developed in response to problems observed in testing and evaluation. Following is a list of these enhancements.

In early June, the 9600-bps access line to the Mitre IMP was upgraded to 56 Kbps and installed. Tests to assess the performance of the new line relative to the old line were begun. The primary reason for installing this line was to improve the precision of measurements made with other hosts on the ARPANET and other nets. A secondary aim was to reduce the frequency of blocking due to "pinging" by other hosts and gateways. "Pinging" refers to a host or gateway's method of to determining connectivity to a host by sending packets to that host that are designed to be returned to the sender. As the number of hosts and gateways on the ARPANET has grown, so has the pinging traffic, and as a result the target hosts and IMPs have been impacted seriously.

Name and time servers were installed on several DCN hosts and clones elsewhere. These were tested in a mini-bakeoff with other implementations at MIT, Purdue, SRI, and BBN. Once during testing an observer at MIT discovered a discrepancy of exactly ten years in one of the time servers. The discrepancy was caused by the year being incorrectly entered on the DCN host equipped with the NBS radio clock.

An unexplained discrepancy was found in the network time distributed by the DCN master clock on one hand and the Ford DCN clone master clock on the other. The DCN master clock was derived from a Spectracom WWVB radio receiver, while the Ford master clock was derived from a TrueTime GOES satellite receiver. While both receivers are reported to be accurate to approximately one millisecond, the two clocks were found to be off almost 50 milliseconds. Qualitative checks with other sources of time information, including WWV and Naval Observatory, indicate that the problem is with the Ford clock, but definitive conclusions have not yet been made.

Another fuzzball (DCN7) was configured for use in program development. It was used for a short time with a US Design 10-megabyte Winchester disk. Unfortunately, this disk has had a poor maintenance history, and it failed once again shortly after integration. The failure was traced to a defective disk motor, which represents a relatively expensive repair. Considering the maintenance history (about $3000 during the life of the disk), we decided to retire it and replace it with one from another manufacturer.

As laboratory space available to the project steadily decreased, there was mounting pressure to economize on the space taken for the fuzzball zoo, including backplanes, disks, and terminals. A five-foot equipment rack was found, and two of the fuzzballs, along with the NBS clock, were assembled in it. In addition, the operator terminals for two of the machines were moved to staff offices. Additional equipment repackaging is expected in the future, at least until relocation of the staff and laboratory space to another facility in December.

Code to support IP source routing was incorporated in the fuzzball system. It was tested first with other fuzzballs and then with the Internet gateways and other hosts known to support source routing, including ISI (TOPS-20) and CCN (IBM 3033) hosts. At this time, the new feature works with all of these implementations, but does not support record-route and return-route functions. In addition, the packet-allocation algorithm was modified to slice large buffers into smaller ones automatically upon allocation and recombine them when freed. This variant of the "buddy algorithm" markedly improved the utilization of buffers in the DCN-GATEWAY host, which experiences traffic flows well in excess of ten thousand packets per hour.

A particularly useful enhancement to the operating system was the introduction of virtual volumes. A virtual volume is a file that can be made to appear to the system as a separate and distinct physical volume. This feature is intended to support multiple users sharing the same machine. Each user can be assigned a separate virtual volume for private files, while using a common volume for system programs, etc. Features were then incorporated into the TELNET and FTP user and server programs to make use of this feature.

A driver for the DEC DZV11 asynchronous multiplexer was completed and integrated into the software distribution in anticipation of an expansion in the number of lines on the Stuttgart fuzzball.

2.4. Other Activities

Planning continued for the installation of a fuzzball at DARPA Headquarters in Arlington, Virginia. The hardware arrived for this system, which is to be integrated with a Dacom/Rapifax digital facsimile machine already on the premises. The integration job consists of assembling cables, mounting connectors, and other similar tasks. Installation is planned for early July.

At various times during the quarter, LINKABIT personnel assisted other organizations in integrating fuzzballs into other DCN clones and nets. This activity consisted of copying various parts of the software system on distribution media and sending it, with DARPA permission, to other organizations, such as NTARE, DFVLR, Purdue, University of Michigan. We also assisted University of Maryland personnel in developing an Interlan Ethernet driver, Ford Aerospace in developing a 3-COM Ethernet driver, and Systems Development Corporation in developing an NSC HYPERchannel driver. A condition for this assistance was that the drivers be available for our use in support of DARPA activities. Linkabit personnel also assisted University of Maryland personnel in testing their IP/TCP, TELNET, and SMTP implementations for the Univac 1100/80.

The ACC XQ/CP HDH communication interfaces for the Stuttgart fuzzball arrived in June. They arrived with the wrong connectors (RS-449), which will have to be changed. A software driver for the DEC RSX11 system arrived with the interfaces. This driver was used as a model, for the design of a fuzzball driver, which is now in implementation. It is scheduled for completion next quarter.

2.5. Experiments

As an integral part of testing in the development of new protocol modules and the measurement of their performance, several experiments were conducted over the Internet system. The experiments fell into three areas: experiments designed to test the robustness and throughputs of bulk transfers between mutually-syspect FTP peers; experiments involving congestion on the 56-Kbps ARPANET access line; and stability experiments using the DCN local-net routing mechanisms.

Last quarter's FTP experiments were continued through this quarter. The effort to sustain this activity has been a small but important part of network testing and evaluation for two reasons: (1) The file-transfer type of service emphasizes high throughput at the possible expense of delay, as compared to the virtual-terminal type of service, which emphasizes low delay at the possible expense of throughput. In addition, the virtual-terminal service, which is mostly interactive, is readily experienced by many users; therefore, degradation in service is readily apparent. (2) As explained below, the file-transfer service, which in principle includes mail, is often performed by daemons and hidden from view. For these reasons, the performance of file-transfer services often is neglected, and as shown below, can lead to ominous, unsuspected and wasteful traffic in the system.

The FTP testing proceeded this quarter in two areas. The first involved verifing teroperability between the DCN implementation and several others, including the BBN TOPS-20 and BBN VAX implementations, which represent the bulk of the hosts now on the Internet. Serious difficulties that have been experienced with TOPS-20 FTP servers in the connection-close sequence, as reported last quarter, have not been fixed as yet. Tests performed this quarter revealed that the TOPS-20 multiple-get/send facility did not work in stream mode with either the DCN or VAX implementations. This problem also has yet to be fixed. Furthermore, the TOPS-20 initial retransmission timeouts were so short that operation was not possible via the 9600-bps line early in the quarter. This was later fixed, with the result that operation improved dramatically and was possible over paths of speeds as low as 1200 bps.

At the January meeting of the Research Group, we reported what appeared to be a bug in the BBN VAX FTP implementation that subsequently was localized in the VAX TCP implementation. The effort to isolate, identify, and fix this bug represents an important lesson for the community.

The bug appeared during a transfer of files from a BBN VAX host to a fuzzball and a relatively slow disk. The transfer would proceed normally and then hang ostensibly forever. An inspection of the packet trace at the fuzzball revealed that the VAX was in a continuous retransmission loop, sending a one-octet segment positioned in sequence space one octet to the right of the right window edge (in other words, an unacceptable segment). The fuzzball properly returned an ACK for its current left window edge, which should have caused the sender to advance its left window edge and eventually, perhaps through retransmissions, to advance to the point where the retransmissions took place. The sender actually never should have transmitted the one-octet segment in the first place.

The fix for this problem was trivial and eventually was made during this quarter. However, the fix apparently did not propagate to all hosts using the BBN VAX software, including some at BBN and, in particular, a VAX at the University of Maryland. An inspection of packet tallies revealed that this host had, in fact, been hung up in this retransmission loop for three days trying to send mail and generating many thousands of packets. There is reason to believe the same thing may be happening elsewhere in the Internet system even now.

The lesson drawn from this experience is that more care must be taken in the distribution of system software and patches; however, the experience also points to the necessity for some kind of host monitoring and reporting. In the case cited,

fail-safe timeouts at either the sender or receiver could have aborted the transfer after a reasonable period.

A set of experiments was conducted using the new 56-Kbps access line to the ARPANET. Delays were measured between the DCN1 host (DCN-GATEWAY) via the line to several selected hosts measured in a similar set of experiments performed about a year ago. The delay distributions have not yet been assembled and plotted; however, it is clear from the means that the overall delays have increased approximately a hundred milliseconds even considering that the earlier experiments were conducted with a 9600-bps access line, while the later ones were conducted with a 56-Kbps line. In addition a large increase in IMP blocking events lasting a second or more were observed. Reports from BBN and other observers confirm that these observations agree with theirs. Subsequently, it was discovered that a large proportion of the traffic visiting the DCN1 host was due to host and gateway "pinging" as described previously.

Bugs found and fixed during the experiments included one that resulted in internal routing loops in the DCN only in those cases where delays via some link increased rapidly (several hundred milliseconds during one HELLO interval - usually in the 10 - 30 second range). The cause was found to be a defect in the algorithm that permitted paths to form from a sender via a neighbor and loop back to the sender when the delay on a path to another destination increased rapidly. The fix was trivial; it involved forcing delays to "infinity" in a HELLO message for all virtual hosts allegedly reachable via the intended receiver of that message.

3. PLANS FOR THE NEXT QUARTER

Th~ plans for the next quarter include continued development of the EGP model and implementation. This will involve development of a topological model from which the routing constraints dictated by the specification will be clearly apparent. In addition, the model should reveal how the connectivity information exchanged between EGP peers should be assembled, what constraints, if any, should apply to the configuration of the EGP peers themselves; and whether a common metric should be employed. These topics are further discussed in Sections 5 and 6 of this document.

Also to be resolved in the next quarter are questions on the performance of the Internet system. Some of the problems revealed in testing FTP, for example, clearly indicate inadequacies in certain implementations, including TOPS-20 and VAX. Others of a more basic nature concern congestion in the network and the method for improving the source-quench mechanism.

Finally, work will continue in the development of an appropriate process model for TCP, in which the role of parameter estimators is defined clearly and their effectiveness assessed experimentally.

4. TCP FLOW AND CONGESTION CONTROLS

The DCN implementation of TCP has been used many times as a test vehicle to investigate various techniques for the control of network congestion (for example, where packets can be lost due to insufficient buffering in a gateway). Some of these control techniques have been reported before [ ], along with analytical models and experimental performance evaluations. Experience in these activities has resulted in a new approach which is based on modelling a system of TCP peers as a random process and controlling its behavior using information derived from a set of estimators maintained in real time as data transfer proceeds.

The following sections describe this approach, which is still in the conceptual phase. In succeeding quarters, this approach will be refined, implemented, and tested. The approach involves the conceptualization of a model believed generally applicable in the space of TCP/IP host implementations, but was developed with the DCN "Fuzzball" implementation in mind. The model is described along with a set of estimators for delays and flows used to manage resource commitments in the host and the network. Also presented are a procedure for calculating these estimators and algorithms using their values to effect per-connection flows.

4.1. TCP System Model

We are concerned with a single virtual circuit consisting of a pair of TCP peers connected by an Internet path involving some number of local nets and gateways. The most interesting cases occur when 1) either host can generate flows well in excess of that which is acceptable to its local net, and 2) when at some point in the path, a flow mismatch of one or two orders of magnitude exists across a gateway connecting two neighboring nets.

Of primary interest is the control of flows for bulk data transfer, such as would characterize an FTP data connection. In such cases, the primary concern is for throughput, rather than delay. Throughput is degraded by retransmissions due to lost packets, which in turn is due to insufficient resources, primarily packet buffers. While end-to-end flows are controlled in TCP by a window mechanism responsive at the level of individual octets, often insufficient resources at gateways along the path cause packets to be lost anyway.

The problem addressed here is to identify a set of estimators that can be used to predict the resource demands implied by a particular flow of data (octets or packets) from moment to moment as transmission proceeds. The estimators can be used to predict delays and flows at several points along the path. The retransmission timeout (zero and non-zero window cases) and flow strategies designed to minimize resource demands throughout the system while sustaining high throughputs can be derived from the estimators.

The host can support a number of simultaneous TCP connections, each with its own set of state variables and estimators (see below). An interval timer is assumed with resolution in the order of a millisecond, and the host operating system is assumed to have some sort of internal flow controls as part of its internal resource management system. The controls act to deny a request for a packet buffer if the quota assigned that connection has been exhausted. Once a request has been denied, another is not made until after a system-dependent interval.

As each packet is filled, it is sent to the net using some sort of IPC message. The packet may be multiplexed along with others on a queue for transmission into the net. When the packet has been transmitted, an IPC message is returned to the TCP process and used to update the estimators for the particular connection (see below).

The local nets may or may not have intrinsic mechanisms to control flows between the hosts and gateways. In the case of the present ARPANET, flows are controlled by RFNMs and blocking. For present purposes, we can assume that control by RFNMs avoids blocking, and that the delays between a transmitted packet and its RFNM can be used to update the estimators for each connection accordingly. This information is conveyed to the TCP process via the IPC message mentioned above.

The gateways typically allocate packet buffers on the basis of input interface and output queue threshold. If a particular output queue threshold is exceeded for a packet arriving at a particular input interface, it is discarded. If some number of packets are discarded in this manner during a system-dependent interval, the gateway returns a source-quench packet to the originator. A host receiving such a packet sends an IPC message to the TCP process, which then updates the estimators for that particular connection.

4.2. Discussion of the Model

The source-quench mechanism generally is agreed to be

inadequate on the basis of several shortcomings. First, the information arrives too late to effect a useful modification in behavior on the connection. Second, the mechanism is effectively bang-bang in nature and can lead to undamped flow-rate transients. Third, information is sent only after considerable numbers of packets have been lost, and presumably, after long delays for TCP retransmissions have occurred.

Several suggestions to improve the effectiveness of source-quench have been made. One is for the gateway to keep an LRU stack for each output queue to help isolate those hosts claiming excessive resources. Another is to send source-quench messages before packets have to be dropped and to include additional information, such as could be derived fom the LRU stack. A third is to send source-quench messages to the destination host as well as the source host, possibly for use in throttling ACKs. It is assumed that the host receiving a source-quench message will use whatever information is available to update the estimators for that connection.

4.3. Estimators

Each estimator contains a value that is computed from past behavior and can be used to predict future behavior. Typically, it is an average of past samples of a random variable, such as roundtrip delay, with newer samples weighted more heavily than older ones. In recursive-filter averaging, a new sample value weighted by w is added to the running average weighted by (1-w). In matched-filter averaging, a new sample is entered in a shift register containing n of the most recent samples and the average computed as the sum of the n sample values divided by n. There is reason to believe that the matched-filter method may work better than the recursive-filter method for some estimators; however, this issue will not be explored in this discussion.

The accuracy of an estimator depends on the number of samples included in the average and the sample variance. An estimator typically is updated for every received ACK packet or IPC message. Since bulk-data transfers typically use large packet sizes, it is important to gain as much information as possible in each sample. A typical problem occurs when the sample value correlates strongly with the length of the packet, which can occur if the degree of aggregation on the path is small. Thus, differences in the lengths of the packets can show up as a larger sample variance, which can lead to increased retransmissions with some choices of TCP parameters. This is the principle cause of performance degradation observed on so-called "tiny-pipe" nets using relatively low-speed non-multiplexed links.

All estimators suggested below are constructed in the same way. When some event happens, such as sending a packet, the time of the event and the current sequence number are recorded in a FIFO stack. When an ACK packet or IPC message arrives, the FIFO stack is searched for the entry with sequence number less than or equal to the sequence number ACKed. The elapsed time is then computed and adjusted by linear interpolation between the entry found and the following entry.

The elapsed time computed in this way can be used to update a delay estimator directly by using one of the averaging methods above. In addition, the elapsed sequence numbers can be used to update a related estimator. By simply dividing the second by the first, a flow-rate estimator can be derived and used directly to control flows on the connection.

There are three estimators that are suggested naturally by the model: the net-input, TCP-ACK, and TCP-window estimators. Each is described below:

Net-input. Two events update this estimator. The first is an IPC message when a packet buffer has been sent to the net. The second is an IPC message that results from a source quench. Not much can be done with the current source-quench mechanism, other than adjusting the estimator value in an ad-hoc way (such as arbitrarily reducing it by half). The flow-rate estimate indicates the rate the sender should use for most efficient use of the Internet path to the receiver.

TCP-ACK. This is the classic "RSRE Algorithm" refined by the FIFO stack and interpolation technique described above. This technique reduces sample variance and probably should have some correction for packet length. The delay estimate is used in calculating the initial TCP retransmission delay. The flow-rate estimate indicates the rate the sender should use to fill the window.

TCP-window. This is computed in the same way as the TCP-ACK estimator, but the sequence number used is the right-window edge, as determined from the window field in the TCP ACK packet. The flow-rate estimate indicates the rate data are being delivered to the end user and thus the net rate of the end-to-end circuit. The sender should try to send at a somewhat higher rate and rely on the TCP window for fine tuning.

4.4. Flow Management

Strategies that use the values of the delay and flow-rate estimators described above are examined in this section. First,

in the case where the sender flow is small compared to the net and end-user capabilities, the TCP-ACK and TCP-window rates should be about the same. In the above mentioned case involving fire hoses and tiny pipes, the net-input rate is likely to be much higher than either of the others. In the case where the end user is much slower than the sender or net, the TCP-window rate will decrease, possibly falling to zero.

The current DCN fuzzball implementation incorporates the first two of these estimators along with a simple recursive-filter averaging method. Flow control is based on the net-input rate, which follows the local-net backpressure and responds to source-quench messages. This is done by throttling the actual flow rate into the net so as not to exceed the net-input rate. The estimated net-input rate is decreased by half for each source-quench message received and is allowed to return to the measured rate in about eight samples. The effect of this is to avoid tying up packet buffers unnecessarily and to provide a control point for flow modulation.

In the fuzzball implementation, the TCP initial retransmission delay is calculated in the classic way from the TCP-ACK estimated delay, with subsequent delays adjusted for backoff. Our experience shows that packet length should be factored in this estimate; however, this would involve estimating two quantities simultaneously--the absolute delay and the end--to-end flow rate. This actually is not especially difficult and may be considered for future implementation. Nevertheless, the performance is good with the present net configuration of up to five links (not themselves flow-controlled) and from speeds of 100 Kbps or so down to 1200 bps.

Our experience indicates that considerable improvement can be made by incorporating more highly developed estimation methods, such as those suggested above, and by coupling the flow-management algorithms more closely. These actions are planned as the implementation is refined. Some of the possible mechanisms are described below.

1. Use the TCP-ACK and TCP-window estimators to implement a strategy designed to avoid extreme efforts to keep the window closed, which lead to silly-window syndrome.

2. Couple the TCP-ACK and TCP-window estimators into the packet-generation strategy mentioned in connection with the net-input estimator. This would avoid pumping many packets into the net well before the end user is ready for the data.

3. Develop a complementary set of estimators for use at the

receiver. They could be used to control the ACK strategy and avoid buffer fragmentation, while minimizing traffic on the reverse direction.

4. Investigate the feasibility of sending additional flow-control information (for example, in the URGENT field of packets when the URGENT condition is not in effect) to help the sender and/or receiver improve its strategy.

5. EGP IMPLEMENTATION PLAN

The Distributed Computer Network (DCNET) is an experimental distributed network architecture based on a set of local-network control algorithms described in [1]. The architecture includes both point-to-point and common-bus configurations using PDP11-compatible hosts (called Fuzzballs) connected by a variety of interface devices. The control algorithms provide adaptive routing, time synchronization, and gateway functions to subnets and foreign nets.

The Gateway-Gateway Protocol (GGP) [2] has been in use for about four years in the Internet system to provide network-level routing functions for gateways between ARPANET, SATNET, and several local-area nets. For several reasons, the GGP has been found inadequate when large numbers of gateways and nets are involved and where multiple implementations coexist.

The Exterior Gateway Protocol (EGP) [3] is an experimental protocol designed to provide network-level routing between systems of gateways organized as loosely-coupled, autonomous systems. The protocol operates between designated gateways in adjacent systems and provides for the identification of neighbors, verification of reachability, and routing of intersystem traffic.

This design note suggests a strategy for implementing EGP in the DCNET architecture. The design is based on a set of distributed algorithms that interoperate with the existing DCNET distributed control algorithms; it is believed to be a good test of the EGP functionality in such environments.

5.1. Overview of DCNET Control Algorithms

The architectural model of a DCNET clone is a self-organizing system including a set of hosts connected by an essentially ad-hoc set of links. Typical DCNET clones include a number of hosts permanantly connected by a high-speed local net, together with a set of dial-up hosts that share a pool of ports. The ports themselves are distributed among the fixed hosts, and the dial-up hosts connect to them in an undisciplined way. The

local net is connected to the Internet system via one or more gateways using either permanant or dial-up links.

A distributed adaptive routing algorithm, called the HELLO algorithm, is used to bind the hosts of each DCNET clone together. Although the agents participating in this algorithm are the physical hosts, the entities representing the nodes of the topology are designated processes called virtual hosts and each is assigned a unique internet address. A physical host can contain one or more of these virtual hosts, which can migrate about the network in arbitrary ways.

DCNET clones can be connected to subnets and foreign nets. A subnet is a DCNET clone identified by the same net number as the parent, but is assigned a distinct subnet number, depending on the address format. Both subnets and foreign nets are connected by gateways; however, the gateway functions are implemented in a distributed fashion with each function possibly associated with a different virtual host. In the present implementation, all functions for each distinct gateway are provided within a single physical host, and each host can support a single gateway. The GGP protocol is used between these gateways and the neighbor gateways in the Internet system.

Routing within a DCNET clone is entirely a function of the IP header, since no local-net leader is used. Each physical host contains a host table with an entry for each virtual host in the net. Each entry, indexed by the host ID field of the Internet address of that virtual host, contains the port ID for the network-driver process on the minimum-delay path to that virtual host, along with the roundtrip delay and logical-clock offset. In addition, each physical host contains a net table with entries defining each net number and the corresponding host ID. Thus, routing to a foreign net consists of two steps: first searching the net table for the host ID, and then using the host table to obtain the port ID of the appropriate net-driver process.

The host tables are maintained by the HELLO algorithm, which uses periodic HELLO messages exchanged between neigboring physical hosts. The net tables are maintained by the UPDATE algorithm using information piggybacked on the HELLO messages. Provision is made for congestion-control information to be piggybacked as well. Routing is effective at each physical host in a path between virtual hosts, including hosts that act like gateways. Thus, neither gateway-acquisition procedures nor redirects are necessary within the net.

The DCNET virtual-host architecture and control algorithms support a model where the gateway routing and leader-mapping functions can be located at traffic forwarding points, while the

information necessary to construct these tables is accumulated elsewhere. In the intended model, the routing information is accumulated in the following ways:

1. At ports where a host belonging to the DCNET clone is attached, routing information is exchanged by the HELLO protocol.

2. At ports where a host belonging to a DCNET subnet or DCNET foreign net is attached, the host provides its net (and subnet) number as part of the HELLO protocol. This provides routing information for adjacent nets (but not other nets that may be connected to the adjacent net). The routing information is then distributed throughout both adjacent nets by means of the UPDATE protocol running separately in each net.

3. At points where a host (in this case, more properly a gateway) belonging to a non-DCNET net is attached, the connecting interface or line is configured with the address of the port on the adjacent network. A HELLO message containing this address is sent periodically into the adjacent net, with the response providing reachability information for the adjacent net, which is then distributed throughout the DCNET clone by means of the UPDATE protocol.

4. Reachability and routing information for non-adjacent networks is provided either by GGP (presently) or by EGP (proposed) agents implemented as virtual hosts. These agents maintain the data bases required and interact with neighbor gateways elsewhere in the internet system. Routing information is then distributed throughout the DCNET clone by means of the UPDATE algorithm. Note that the agents do not have to be in physical hosts adjacent to the foreign nets.

The HELLO and UPDATE protocols thus provide adaptive routing for virtual hosts in a DCNET clone and between adjacent DCNET clones. Local hosts can connect freely at any port. DCNET subnets and foreign nets can also connect freely, with the provision that that all traffic between hosts on the same net must travel via paths in that net. In the case of foreign nets not responding to these protocols, the configuration is necessarily fixed; however, the HELLO and UPDATE protocols still provide routing information throughout the DCNET clone.

The EGP protocol is to replace the GGP protocol between a DCNET gateway and a foreign autonomous system; however, the GGP protocol (modified somewhat) may still be used within a cluster of DCNET clones acting as an autonomous system. Either protocol provides information to update the routing data base for

non-adjacent networks. This information is necessarily bandwidth-limited and less reliable than the that provided by the local protocols. However, as should be evident from the architecture described here, the layered structure of the protocols (EGP and GGP depending on UPDATE; UPDATE depending on HELLO) is intended to provide the highest reliability and speed of response for paths between local hosts, the next highest for paths between hosts on adjacent nets, and the lowest for hosts on non-adjacent nets.

An important restriction with GGP in the DCNET environment has been removed with EGP. With GGP, the "address" of the DCNET agent that participates in the protocol must be an address on the adjacent network, in this case the ARPANET. As long as the agent virtual host is physically resident in the host terminating the link to the adjacent network, this is handled by simply multi-homing the virtual host; however, this does not work if the agent is in a different physical host. The switch to EGP removes this limitation and allows interesting experiments where the EGP functions themselves are distributed within the DCNET clone.

5.2. Implementation Plan

The implementation plan is designed to reduce the impact on the existing protocols as much as possible while providing the full functionality of EGP. In particular, the HELLO and UPDATE algorithms remain substantially unchanged. However, certain changes will be necessary, simply because the abundance of networks threatens to drown the system regardless of the protocols.

The most significant change is to reduce the amount of network routing information distributed by the UPDATE algorithm. Presently, the entire net table containing the local (gateway) host ID, status, and number of hops is distributed throughout the local net. In typical configurations, usually one path to non-adjacent nets is available for a DCNET clone and only one EGP agent will be necessary. The number of non-adjacent nets usually will be much larger than the number of adjacent ones. The EGP agent can then group a number of nets reachable via the same local (gateway) host ID and assign them a special group ID. For the moment, only a single such group ID will be allowed. This will be distributed as part of the UPDATE algorithm to all hosts in the net.

When a host attempts to find a host (gateway) ID for a given net number, it searches its copy of the net table. If no explicit match is found, it selects the group ID assigned by the EGP agent. This will work for all those nets known to the EGP

agent, but will work improperly for unknown nets. The problem can be handled two ways: 1) The group ID distributed by the EGP agent points to itself. Thus, all traffic to these nets will transit the EGP agent, which can then forward the traffic or return an ICMP net-unreachable message accordingly. 2) If traffic for all non-adjacent nets can be routed via one gateway, the group ID points to that gateway. When a datagram arrives at the gateway, the leader-mapping function (see below) will detect instances of unknown nets, so that ICMP net-unreachable messages can be generated accordingly.

The EGP agent itself will be implemented as a virtual host. It will contain the data base describing the full complement of nets provided by its neighbors and by the UPDATE algorithm in the local net. It will provide the UPDATE algorithm with information about these nets as appropriate in view of the considerations above. The operation of the EGP agent is complicated by the requirement to distribute local-leader information to those gateway processes connected to adjacent nets that require local leaders, such as the ARPANET. This will require a new protocol, possibly a modification to the UPDATE protocol, in a form that will be decided later.

The initial prototype implementation will include an EGP protocol module operating in parallel with GGP (i.e., on the same routing data base). The effect will be to assign one set of neighbors to GGP and another (possibly overlapping) set to EGP. The DCN-GATEWAY host, connected between ARPANET, DCNET, FORDNET and UMDNET, would contain both protocol modules and would run GGP with ARPANET neighbors and EGP with another test gateway on DCNET, probably DCN6. The existing GGP protocol module and virtual-host process has already been changed to support multiple gateway protocols.

Once the neighbor acquisition, reachability and update message formats, and basic protocol functions have been tested, testing with another implementation, presumably MIT, is recommended. Testing our implementation with either the FORDNET or UMDNET DCNET clones also can occur without danger of disrupting outside traffic. Later, we would like to explore the issues raised by distributing the EGP functionality as discussed above.

6. EGP ARCHITECTURE ISSUES

The following discussion explores certain issues unresolved in the current EGP specification (RFC-827), which arose while designing the test implementation of an EGP gateway for the DCN Fuzzball system. At this time, many of these issues need to be discussed and resolved among the participants in the special-interest group charged with EGP specification.

6.1. EGP System model

There is some confusion about the definition of an autonomous system and its connectivity constraints. A precise statement of what is believed to be the intended model follows.

An autonomous system (henceforth simply system) is a set of gateways. Every gateway belongs to exactly one system. Two gateways are connected with respect to system s if they both belong to s and share a net in common. The gateway-connected relation for each system is reflexive, symmetric, and transitive, and thus an order relation. Its transitive closure must be an identity, so that the set of nets included is not partitioned. Two systems are connected if there is a pair of gateways, one in each system, that share a net in common or are connected directly by a point-to-point link that is not considered part of any net. The system-connected relation is an order relation, and its closure must also be an identity. Thus, every net is reachable from all gateways in all systems.

Gateways belonging to a particular system communicate routing information using an interior gateway protocol (IGP), an example of which is the Gateway-Gateway Protocol (GGP). One or more designated gateways in each system communicate routing information to designated gateways in other systems using the Exterior Gateway Protocol (EGP).

EGP is spoken between a pair of peer gateways, each belonging to a different system and connected in the sense above. These are called direct neighbors and, presumably, each participates in its own IGP to exchange routing information with other gateways in its system. The routing information exchanged between these systems may include information derived from other systems via EGP. In particular, the routing information may indicate that traffic for some nets should be directed to gateways other than those particular EGP peers, including those belonging to other systems. These other gateways are called indirect neighbors.

6.2. Topological Constraints

The topology of the systems has been specified to be tree-structured, with the "core" system at the root of the tree, in order to avoid potential routing loops. There is, however, no implied restriction on the net topology within each system.

A net is internal to a system if all gateways sharing that net are in that system and external otherwise. The set of all

internal nets, together with their gateways, can be considered as constituting a single net whose internal structure is invisible outside the system.

The tree-structure restriction was designed to avoid potential routing loops between systems. Thus, each system can share, at most, one external net with any other system and there are no cycles involving external nets. Strictly interpreted, the restriction would forbid more than one set of EGP peers connecting two systems sharing the same net.

There is some evidence that this is not a workable or even tenable restriction. Systems very likely will grow appendages that may form loops. The alternative can only be a registered topology, which does not seem feasible for the research community. In addition, not only must the topology be tree-structured, but the topology can be changed only after all old routing information has been purged from all routing tables. Past experience indicates that there is no way to assure this, other than administrative control of all gateways.

The assumption is that the systems can (but not necessarily will) determine routing via their neighbor systems using a universal metric. In such a case, loops are broken by "counting to infinity," which takes time when polling rates are low. However, loops can be avoided in the first place by employing a hold-down, which would operate to inhibit inclusion of a net in an EGP net-reachability message for a specified period following the time the path to that net was determined to be down. The hold-down interval may have to be relatively long, since it must be at least as long as the time to propagate routing changes to the far corners of the Internet.

We have thus exchanged the tree-structure restriction for a set of specified parameters that must be taken as characteristic of the Internet. One of these parameters is the maximum diameter of the Internet (which places some restrictions on the topology); a value of about eight might be right. A second parameter is the maximum hold-down interval--two to ten minutes may be necessary. Finally, every system that might be part of a loop would be expected to propagate routing changes in a timely manner and to comply with the hold-down requirement.

6.3. Routing

There should be a universal metric understood by all gateways. This does not necessarily mean that each IGP must use this metric, but it does mean that the metric is understood in all EGP communications. The metric must in fact be a metric; that is, it must be reflexive and symmetric, and must obey the triangle inequality. Simple hop-count is suggested.

While some systems may elect to constrain internal routing procedures, a consistent set of neighbor gateways and distances should be presented to neighboring systems. No attempt should be made to bias the routing decisions of the neighboring systems. The easiest way to do this in the present implementation is to extend the GGP metric everywhere, so that EGP routing decisions can be made in the same way as GGP. This does not imply that a gateway must believe the EGP routing information. In fact, it may elect to use (and report to other gateways) interior paths rather than allegedly shorter exterior paths.

A gateway (or host) can obtain a list of nets and first-hop gateways from a consenting EGP gateway on its net. There could be several neighboring systems sharing the net, each of which has such a gateway. The lists obtained from each gateway should contain equivalent information; however, different first-hop gateways may be suggested for a particular net. The understanding in such cases is that each of these first-hop gateways is associated with the same distance to that net.

6.4. Neighbor Acquisition

EGP neighbors evidently must share a common net, because otherwise they could not establish a way to determine a route between themselves. In general, there must be one or more EGP representatives for each system sharing a common net. Presumably, each representative would be an IGP participant containing an EGP protocol module capable of sustaining simultaneous EGP operations with representatives in other systems.

As yet, no architecture has been developed for finding either IGP or EGP neighbors on a particular net. In current GGP all neighbors are direct neighbors, and the protocol is sung between each pair of peers separately. Thus, it is sufficient for only one member of each pair to know the address of its neighbor apriori. However, this can lead to situations where a gateway may not discover all its neighbors on a particular net, resulting in suboptimal routing. The problem could be remedied through redirects, but at present this is not done.

Sufficient information is conveyed in EGP to allow each peer to discover at least those direct and indirect neighbors along the minimum-hop path to every other net known to the IGP of either system. These neighbors must not be considered potential EGP neigbors, even if they belong to another system. Thus, suboptimal routing can occur in EGP as well.

In order to minimize overheads, it is desirable to structure the host-gateway interaction so that no state information need be maintained at the gateway. This can be done by simply sending an NR-poll message to the selected EGP gateway, upon which the gateway will return an NR-update message. Further polls should not be necessary, unless it is determined that one of the gateways has become unreachable as reported by local-net or high-level protocols. It is not necessary to send NA-requests or HELLO messages unless the host is connected to another net (i.e., it is, in fact, a gateway). A gateway should not incorporate routing information received from another gateway into its routing data base unless it has satisfied the reachability criteria.

Redirects should be sent to gateways as well as hosts. It may happen that a gateway in one system may redirect a gateway or host to a gateway on another system. The redirect mechanism can be expected to operate much faster than the routing-update mechanism in either EGP or (existing) GGP. The effect of interactions between this kind of routing information and that conveyed by EGP may be an interesting topic for further study.

It is particularly important to refine the model for NR-updates. An initial approach is to simply extend the GGP metric and neighbor selection into EGP, which would certainly simplify the gateway design. The intent is for the EGP neighbor to receive all the information necessary to construct a routing matrix as if it were direct GGP neighbors of each indirect neighbor. This could be done by simply sending suitably edited copies of the routing matrix; however, that leads to a considerable amount of redundant and probably useless information.

Another approach is to send a copy of the routing vector, together with the associated first-hop gateway, to each net. This is much more compact, but does not have the correct distance, because the distances are relative to the sending gateway and would be incorrect from the neighbor's point of view.

We conclude that the right thing to do is to send a modified routing vector (re-sorted by gateway address, of course) together with the first-hop gateways. The modification amounts to subtracting one from the distance value for every net associated with a first-hop gateway other than the sender. In effect, the sender has done all the route computing for the receiver; the receiver must only store the vector and use it. If other EGP peers are sending these data to the receiver as well, the receiver need only pick the one with the smallest distance, exactly as if an ordinary GGP update were being sent.

These issues will be discussed during the next quarter with the intent of reaching closure soon. The test EGP implementation is now in service; however, the above issues are not yet resolved. It is expected that as these issues are resolved, the test implementation will evolve correspondingly.

7. REFERENCES

1. Mills, D.L. Final Report on the COMSAT Internet Program. COMSAT Laboratories, January 1983.

2. Hinden, R., and A. Sheltzer. The DARPA Internet Gateway. DARPA Network Working Group Report RFC-823, Bolt Beranek and Newman, September 1982.

3. Rosen, E.C. Exterior Gateway Protocol (EGP). DARPA Network Working Group Report RFC-827, Bolt Beranek and Newman, October 1982.

Appendix A. DCNET NETWORK OPERATIONS

The following sections describe the data structures and protocols used by the DCNET to facilitate automatic routing, time synchronization, and fault detection and to maintain connectivity with the other hosts and nets of the catenet. This is a revision and expansion of the material in Section 4 of Reference [1] (see Section 7 above). Of particular importance to the discussion in Sections 5 and 6 above are Sections A.4 (Host Updates) and Section A.5 (Net Updates). Sections A.1 through A.3 serve as an introduction to these sections.

A brief description of the process and addressing structure used in the DCNET follows. A DCNET physical host is a PDP11-compatible processor that supports a number of cooperating sequential processes, each of which is given a unique 8-bit identifier called its port ID. Every DCNET physical host contains one or more internet processes, each of which supports a virtual host given a unique 8-bit identifier called its host ID. Of the four octets in the internet address, only the third (class-A/B addresses) or fourth (class C addresses) is significant for DCNET host addressing and indicates the host ID of a virtual host. Each DCNET physical host is identified by a unique host number only for the purpose of detecting loops in routing updates, which establish the minimum-delay paths between the virtual hosts. By convention, the physical host number is assigned as the host ID of one of its virtual hosts.

Each virtual host can support multiple internet protocols, connections, and in addition, a virtual clock. Each physical host contains a physical clock that can operate at an arbitrary rate and, in addition, a 32-bit logical clock that operates at 1000 Hz and is assumed to be reset each day at 0000 hours UT. Not all physical hosts implement the full 32-bit precision; however, in such cases the resolution of the logical clock may be somewhat less. The date representation is in RT-11 format and is incremented when the logical-clock is reset.

A link to a foreign net is associated with a pseudo-host, sometimes called a gateway, which is assigned a unique host ID. The physical link associated with a gateway is identified with this host ID as part of the configuration procedure. In all other cases, the links connecting the various DCNET hosts can be distributed in arbitrary ways, as long as the net remains fully connected. If full connectivity is lost due to a link or host fault, the virtual hosts in each of the surviving segments can continue to operate with each other and, once connectivity is restored, with all of the segments.

Routing of datagrams from a physical host to each of the virtual hosts in the net is determined by its Host Table. This table contains estimates of roundtrip delay and logical-clock offset for all virtual hosts in the net. For the purpose of computing these estimates, the delay and offset of each virtual host relative to the physical host in which it resides is assumed zero. In addition to the delay and offset information the Host Table contains timestamp, leader, and routing information as described below.

The delay and offset estimates are updated by HELLO messages exchanged on the links connecting physical-host neighbors. The HELLO messages are exchanged frequently, but not so often as to materially degrade the throughput of the link for ordinary data messages. A HELLO message contains a copy of the delay and offset information from the Host Table of the sender, as well as information to compute the roundtrip delay and logical-clock offset of the receiver relative to the sender. In some cases the HELLO message contains information to update the Net Table of the receiver as well.

The Host Table is updated by HELLO messages from each neighboring physical host and in certain other cases. The updating algorithm is similar to that used in the ARPANET and in other places, in that the roundtrip delay calculated to a neighbor is added to each of the delay estimates given in its HELLO message and compared with the corresponding delay estimates in the Host Table. If a delay computed in this way is less than the delay already in the Host Table, the routing to the corresponding virtual host is changed accordingly. The detailed operation of this algorithm, which includes provisions for host up-down logic and loop suppression, is summarized in a later section.

The portable virtual-host structure used in the DCNET encourages a rather loose interpretation of addressing. In order to minimize confusion in the following discussion, the term "host ID" will be applied only to virtual hosts, while "host number" will be applied to the physical host, called generically the DCNET host.

A.1. DCNET Addressing

The DCNET uses a three-level addressing structure including nets, subnets, and hosts. In class-A, class-B and class-C addresses, the net structure is defined by the Internet addressing specifications and consists of one, two, and three octets respectively. In class-A and class-B addresses, the second octet is interpreted as a DCNET subnet number and the third octet as the host ID. In class-C addresses, the fourth octet is interpreted as the host ID. In class-A and class-B

addresses, the fourth octet presently is not significant for routing within a DCNET subnet. DCNET subnets can be interconnected freely with each other and with other nets conforming to the Internet specifications. However, automatic routing is effective only at the subnet level, since HELLO messages are exchanged only between hosts on the same subnet. Some DCNET services, such as time and date synchronization, are effective across the boundary connecting two DCNET subnets.

Gateways are used between nets and subnets and between subnets and subnets. However, DCNET gateways are not necessarily hosts in themselves, but virtual hosts that share resources with other virtual hosts in the same physical host. The present implementation supports automatic routing at the gateway-gateway level and is compatible with the standard Internet gateway implementation, but it does not support the transmission of reports to the internet monitoring system.

The DCNET addressing structure is compatible with all three classes of Internet-address formats. When operated as a class-A net, a collection of DCNET subnets appears as a collection of hosts, notwithstanding the subnet structure itself, which is invisible outside the net. When operated as a class-B net, each DCNET subnet appears as a separate net. When operated as a class-C net, each DCNET host appears as a separate net and is itself responsible for subaddressing.

A.2. Network and Host Tables

There are two tables in every DCNET host that control routing of Internet Protocol (IP) datagrams: the Network Table and the Host Table. The Network Table is used to determine the pseudo-host (gateway) on the route to a foreign net, while the Host Table is used to determine the link, with respect to the DCNET host, on the route to a virtual host. These tables are maintained dynamically using updates generated by periodic HELLO messages. In addition, entries in either table can be changed by operator commands.

The Network Table format is shown in Figure A.1.

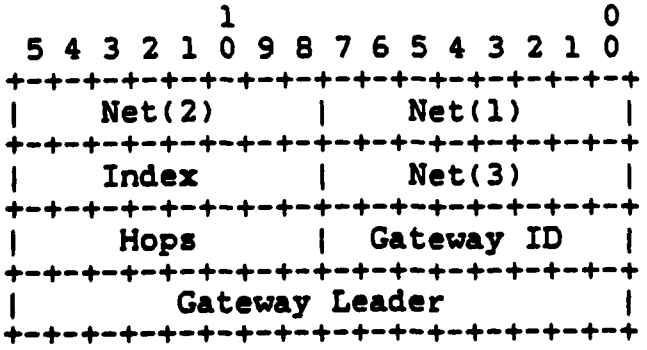


Figure A.1. Network Table Entry

The "Net" fields define the class A/B/C net numbers. The "Index" field is used by the distributed updating algorithm (see later sections). The "Gateway ID" field contains the host ID of the first gateway to the net and the "Hops" field the number of gateways to it, as determined by the distributed updating algorithm. The "Gateway Leader" field contains the (byte-swapped) local-net leader for the gateway on an adjacent net. This field presently is used only for ARPANET gateways and contains the host and IMP address of the neighbor gateway to the net. Network Table contains an indefinite number of entries and is terminated by a zero word immediately following the last entry.

The Host Table format is shown in Figure A.2.

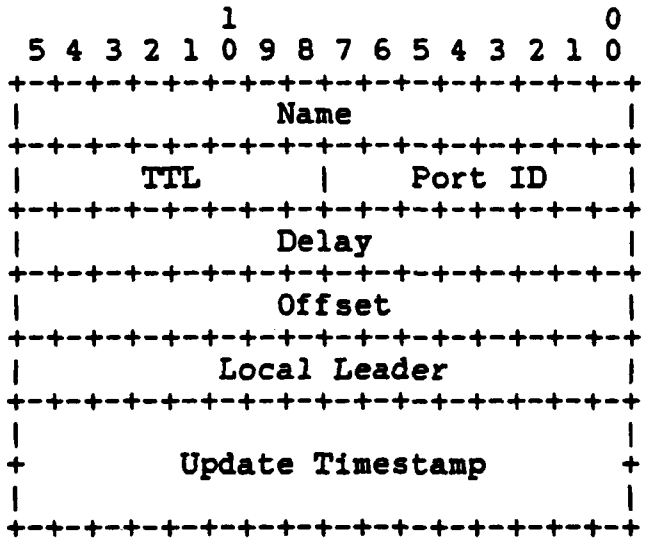


Figure A.2. Host Table Entry

The ordinal position of each Host Table entry corresponds to its host ID. The "Name" field contains a short (RAD50) name for convenient reference. The "Port ID" field contains the port ID of the link output process on the shortest path to this virtual host, and the "Delay" field contains the measured roundtrip delay to it. The "Offset" field contains the difference between the logical clock of this host and the logical clock of the local host. The "Local Leader" field contains information used to construct the local leader of the outgoing packet, for those nets that require it. The "Update Timestamp" field contains the

logical clock value when the entry was updated last, and the "TTL" field contains the time (in seconds) remaining until the virtual host is declared down.

All fields except the "Name" field are filled in as part of the routing update process, which is initiated upon arrival of a HELLO message from a neighboring DCNET host. This message takes the form of an IP datagram carrying the reserved protocol number 63 and a data field, as shown in Figure A.3.

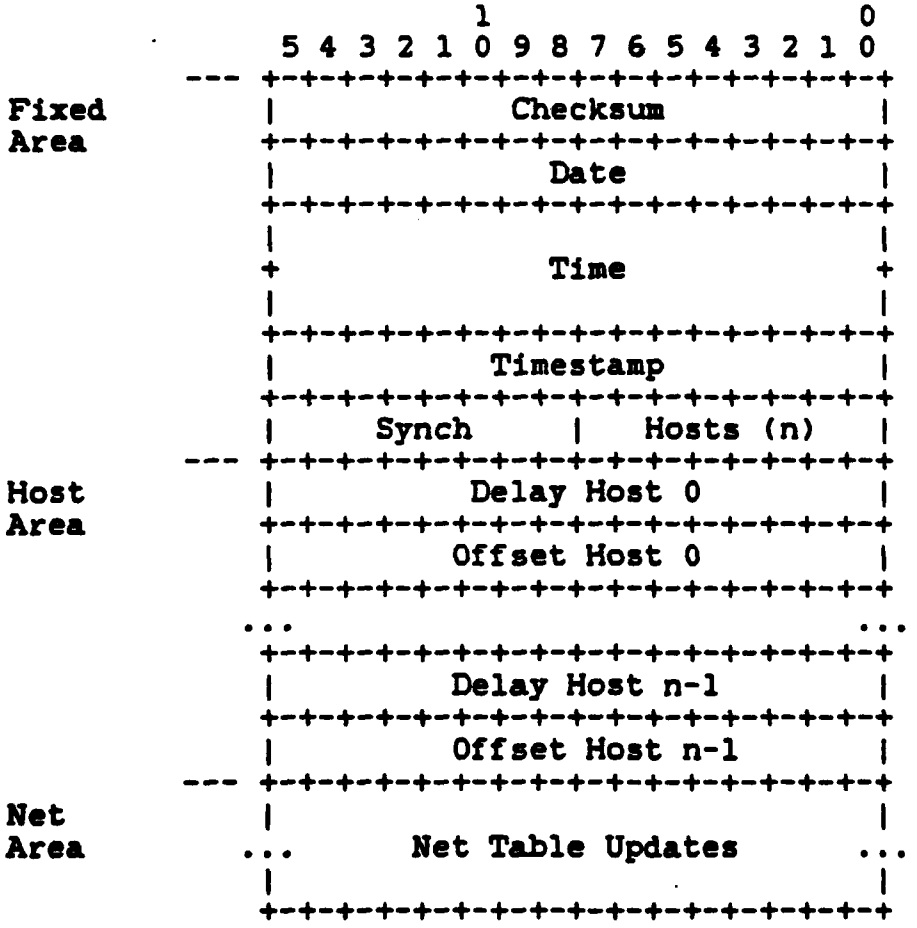


Figure A.3. HELLO Message Format

There are three HELLO message formats, which are used according to the length of the message. One format, sent by a DCNET physical host to a neighboring host that does not support the DCNET local-net protocols, includes only the fixed area shown above. A second format, used when Net Table information is changing, includes the fixed and host areas and, in addition, the net area. The third format, used in all other cases, includes only the fixed and host areas.

The net update information consists of 16-bit sequence number, followed by a number of variable-length entries in the form of a one-to-three octet net number, followed by the "Gateway ID" and "Hops" Net Table entries of the sender. The net area is delimited by the length of the datagram, as determined from the IP header.

Note that all word fields shown are byte-swapped with respect to the ordinary PDP11 representation. The "Checksum" field contains a checksum covering the fields indicated. The "Date" and "Time" fields are filled in with the local date and time of origination. The "Timestamp" field is used in the computation of the roundtrip delay (see below). The "Synch" field presently is unused. The "Delay Host n" and "Offset Host n" fields represent a copy of the corresponding entries of the Host Table as they exist at the time of origination. The "Hosts (n)" field contains the number of entries in this table.

A.3. Roundtrip Delay Calculations

Periodically, each DCNET physical host sends a HELLO message to its neighbor on each of the communication links common to both of them. For each of these links the sender keeps a set of state variables, including a copy of the source-address field of the last HELLO message received. When constructing a HELLO message the sender sets the destination-address field to this state variable and the source-address field to its own address. It then fills in the "Date" and "Time" fields from its logical clock and the "Timestamp" field from another state variable. It finally copies the "Delay" and "Offset" values from its Host Table into the message and constructs the "Net Table Updates" information as required.

A host receiving a HELLO message discards it if the checksum fails. It then checks whether the source-address field matches the state variable containing the last address stored. If not, the link has been switched to a new host, so the state variables are flushed and the link forced into a recovery state. The host then checks whether the destination-address field matches its own address. If so, the message has been looped, roundtrip delay information is corrected, and the host and net areas are ignored. If not, the host and net areas of the message are processed to update the Host and Net Tables.

Roundtrip delay calculations are performed as follows. The link input/output processes assigned each link maintain an internal state variable, which is updated as each HELLO message is received and transmitted. When a HELLO message is received, this variable takes the value of the "Time" field plus the current time-of-day. When the next HELLO message is transmitted,

the value assigned the "Timestamp" field is computed as the low-order 16-bits of this variable minus the current time-of-day. Following transmission, the state variable is reset to zero.

When a HELLO message with a nonzero "Timestamp" field is received, the roundtrip delay is computed as the low-order 16-bits of the current time-of-day minus the value of this field. If this field is zero, then either the neighbor host has never received a HELLO message from the local host, or the neighbor host has not received a HELLO message during the interval between two successive HELLO message transmissions. In order to assure the highest accuracy, the calculation is performed only if the length of the last transmitted HELLO message (in octets) matches the length of the received HELLO message.

The above technique renders the calculation independent of the clock offsets and intervals between HELLO messages at either host, protects against errors that might occur due to lost HELLO messages, and works even when a neighbor host simply forwards the HELLO message back to the originator without modifying it. The latter behavior, typical of non-DCNET gateways, requires a loop-detection mechanism so that correct calculations can be made, and so that spurious host updates can be avoided.

A.4. Host Updates

When a HELLO message that results in a valid roundtrip delay calculation arrives, a host update process is performed. This consists of adding the roundtrip delay to each of the "Delay Host n" entries in the HELLO message in turn and comparing each of these calculated delays to the "Host Delay" field of the corresponding Host Table entry. Each entry is then updated according to the following rules:

1. If the link connects to another DCNET host on the same net and the port ID (PID) of the link output process matches the "Port ID" field of the entry, then update the entry.

2. If the link connects to another DCNET host on the same net, the PID of the link output process does not match the "Port ID" field, and the calculated delay is less than the "Host Delay" field by at least a specified switching threshold (currently 100 milliseconds), then update the entry.

3. If the link connects to a foreign net and is assigned a pseudo-host ID corresponding to the entry, then update the entry. In this case only, use as the calculated delay the roundtrip delay.

4. If none of the above conditions are met, or if the virtual

host has been declared down and the "TTL" field contains a nonzero value, then no update is performed.

The update process consists of replacing the "Delay" field with the calculated delay, the "Port ID" field with the PID of the link output process, the "Update Timestamp" field with the current time of day, and the "TTL" field by a specified value (currently 120) in seconds. If the calculated delay exceeds a specified maximum interval (currently 30 seconds), the virtual host is declared down by setting the corresponding "Delay" field to the maximum and the remaining fields as before. For the purposes of delay calculations, values less than a specified minimum (currently 100 milliseconds) are rounded up to that minimum.

The "Offset" field also is replaced during the update process. When the HELLO message arrives, the value of the current logical clock is subtracted from the "Time" field, and the difference is added to one-half the roundtrip delay. The resulting sum, which represents the offset of the local clock to the clock of the sender, is added to the corresponding "Offset" field of the Hello message, and the sum replaces the "Offset" field of the Host Table. Thus, the "Offset" field in the Host Table for a particular virtual host is replaced only if that host is up and is on the minimum-delay path to the DCNET host.

The purpose of the switching threshold in (2) above and the minimum delay specification in the update process is to avoid unnecessary switching between links and transient loops, which can occur due to normal variations in propagation delays. The purpose of the "TTL" field test in (4) above is to ensure consistency by purging all paths to a virtual host when that virtual host goes down.

In addition to the updates performed as HELLO messages arrive, each virtual host in a DCNET host also performs a periodic update of its own Host Table entry. The update procedure is identical to the above, except that the calculated delay and offset are taken as zero. At least one of the virtual hosts in a DCNET host must have the same host ID as the host number assigned the DCNET host itself, and all must be assigned the same net number. There are no other restrictions on the number or addresses of Internet processes resident in a single DCNET host. It should be appreciated that virtual hosts are truly portable and can migrate about the net, should such a requirement arise. The host update protocols described here ensure that net routing procedures always converge to the minimum-delay paths via operational links and DCNET hosts.

A.5. Net Updates

The Net Tables in the various physical hosts provide information to route datagrams to the appropriate gateway for forwarding into a neighbor net. The UPDATE distributed-update algorithm, described in this section, is designed to adjust the entries in these tables to contain consistent information. It uses the net area of the HELLO messages to do this, but only if the entries are inconsistent. In the steady-state case when the entries are consistent, this area is not used and, therefore is not transmitted.

The Net Tables are intended to be close copies of one another, and to be updated as new information is received from the external gateway system as the result of a routing change. Ordinarily this happens when a link to a neighbor net changes state or when a GGP routing update is received from a foreign neighbor gateway and processed by the GGP protocol module in one of the local-net virtual hosts. The new information then is incorporated by a host that "owns" a particular entry (i.e., net) into its Net Table and then propagated to all other copies by the UPDATE algorithm described here.

The UPDATE algorithm operates in conjunction with the HELLO algorithm, which determines the connectivity and routing of the local network. The HELLO algorithm operates using periodic HELLO messages transmitted periodically by a host to each of its neighbors. Information used by the UPDATE algorithm is piggybacked on these messages; however, the two algorithms are otherwise independent.

The UPDATE algorithm operates on a distributed data structure in which each host maintains a separate copy of the routing table, identified by a sequence number S. In addition, each host keeps the last sequence number R(i) received from neighbor i, together with a control bit b(i). When a HELLO message is to be sent to neighbor i, the host computes the difference R(i) - S. If this difference is nonzero, the sequence number S is included in the message and followed by a copy of the routing table. This is called a routing update. If the difference is zero and b(i) is nonzero, only S is included and the routing table is not. This is called a routing acknowledgment. If the difference is zero and b(i) is zero, no routing information is included in the HELLO message. In all cases, b(i) is set to zero as the HELLO message is sent.

A host receiving a HELLO message from neighbor i first processes the local-network routing information. If from this information it is determined that the neighbor or the link connecting it is down, no further processing is done. If this is not the case, and routing information is included in the message, the host computes the difference R(i) - S. Subsequent processing

depends on this difference and the presence of a routing table in the update.

Case 1: Routing update. Neighbor i believes its routing table to be more recent. The host proceeds as follows:

R(i) - S < 0. Do nothing. Neighbor i's routing table is, in fact, less recent and will be updated when the next HELLO message is sent to it.

R(i) - S >= 0. Set b(i) to one and perform a table-update operation (see below). If no discrepancies are found, set S = R(i). Otherwise, set S = R(i) + 1. Neighbor i's routing table is at least as recent and thus updates the host.

Case 2: Routing acknowledgment. Neighbor i believes its routing table to be current and is simply acknowledging the last update sent by the host. The host proceeds as follows:

R(i) - S < 0. Do nothing. In this case the host has received a more recent update from another neighbor since the last update transmitted to neighbor i. Neighbor i's routing table is thus less recent and will be updated when the next HELLO message is sent to it.

R(i) - S = 0. Do nothing. Neighbor i's routing table agrees with the host.

R(i) - S > 0. Set b(i) to one. This case can happen only if the host has crashed and recovered with an invalid (old) sequence number. Recovery will be initiated when the next HELLO message is sent to neighbor i.

The table-update procedure incorporates a vector of updates U received from a neighbor into the host routing table T as follows. The entries u(i) of U and t(i) of T correspond one-to-one. In addition, each entry t(i) includes a lock bit p(i), which is set to one if the host owns that entry. During the table-update procedure, each entry u(j) is compared to its corresponding t(i). If unequal, a discrepancy is said to exist. In addition, the value of u(j) replaces the value of t(i), but only if p(i) is zero.

It is expected that every entry of the routing table (copies of which are maintained and coordinated by the UPDATE algorithm) will be owned by exactly one host. The operation of the algorithm is designed so that a host owning an entry can change its value at any time in its own copy, with these changes propagating automatically to all other copies. Thus, a host making such a change is required only to increment its sequence

number S each time a change is made. When a number of such changes are made in a relatively short time, there will be a period during which some hosts will have the new information and some the old, and even cases where the old temporarily replaces the new. However, these transients will be attenuated with time, with the result that all copies eventually will be consistent and will contain the new information.

Upon first coming up a host needs to get an initial copy of the routing table from one of its neighbors. It does this by setting its sequence numbers S to zero and R(i) (for all i) to any nonzero value. In addition, if a host updates (wraps around) its own S, to zero, S is incremented again to one. An S value of zero, treated as a special case, is less than any other value; thus, the first update from a host coming up always appears as old and causes the receiving host to transmit a current copy of its routing table.

It is not difficult to construct an informal proof that, if the algorithm converges (i.e., S = R(i) (for all i) for all hosts in the net), all routing tables contain identical information. It is more difficult to show that the algorithm always converges in a finite number of steps for any initial configuration of table values. In typical DCNET configurations of up to a dozen hosts with network diameters up to four, the algorithm always has converged within a few HELLO intervals.

A.6. Timeouts

The "TTL" field in every Host Table entry is decremented once a second in normal operation. Thus, if following a host update another update is not received within an interval corresponding to the value initialized in that field, it decrements to zero, at which point the virtual host is declared down and the Host Table entry set as described above. The 120-second interval used currently provides for at least four HELLO messages to be generated by every neighbor on every link during that interval, since the maximum delay between HELLO messages is 30 seconds on the lowest-speed link (1200 bps). Thus, if no HELLO messages are lost, the maximum number of links between any virtual host and any other is four.

The "TTL" field is initialized at 120 seconds when an update occurs and when the virtual host is declared down. During the interval this field decrements to zero immediately after being declared down, updates are ignored. This provides a decent interval for the bad news to propagate throughout the net and for the Host Tables in all DCNET hosts to reflect the fact. Thus, the formation of routing loops is prevented.

The IP datagram forwarding procedures require decrementing the "time-to-live" field in the IP header once per second, or at each point where it is forwarded, whichever comes first. The value used currently for this purpose is 30; therefore, an IP datagram can live in the net no longer than that number of seconds. Thus, this is the maximum delay allowed on any path between two virtual hosts. If this maximum delay is exceeded in calculating the roundtrip delay for a Host Table entry, the corresponding virtual host will be declared down.

The interval between HELLO messages on any link depends on the data rate supported by the link. As a general rule, this interval is set at 16 times the expected roundtrip time for the longest packet to be sent on that link. For 1200-bps asynchronous transmission and packet lengths to 256 octets, this corresponds to a maximum HELLO message interval of about 30 seconds.

Although the roundtrip delay calculation, on which the routing process depends, is relatively insensitive to net traffic and congestion, stochastic variations in the calculated values ordinarily occur due to coding (bit or character stuffing) and medium perturbations. In order to suppress loops and needless path changes, a minimum switching threshold is incorporated into the routing mechanism (see above). The interval used for this threshold, as well as for the minimum delay on any path, is 100 milliseconds.

END

FILMED

3-84

DTIC