(12)

UCLA

OFFICE OF ACADEMIC COMPUTING

"Continued Development of Internet Protocols under the IBM OS/MVS Operating System"

FINAL TECHNICAL REPORT

TR46

Robert T. Braden

January 25, 1985

Sponsored by

Defense Advanced Research Projects Agency (DoD)
ARPA Order No. 4823

Under Contract MDA903-83-C-0435 issued by
Department of Army, Defense Supply Service-Washington,
Washington, DC 20310

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER OAC/TR46 ✓	2. GOVT ACCESSION NO. AD-A167180	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) "Continued Development of Internet Protocols Under the IBM OS/MVS Operating System"	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert T. Braden	8. CONTRACT OR GRANT NUMBER(s) MDA 903-83-C-0435
9. PERFORMING ORGANIZATION NAME AND ADDRESS Office of Academic Computing 5628 Math Sciences Addition C0012 - UCLA Los Angeles, CA 90024	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Order 4823
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209	12. REPORT DATE January 25, 1985 13. NUMBER OF PAGES 45
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution Unlimited	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)	
18. SUPPLEMENTARY NOTES	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer Communication Protocols. ARPANET Control Program. Internet Protocols. TCP. IP. TELNET. FTP. SMTP. Performance Measurement.	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) UCLA has developed and distributed the ARPANET Control Program (ACP), host software for the DoD Internet Protocols to execute on an IBM 370 under OS/MVS. The ACP supports TELNET, FTP and SMTP. This report describes the protocol features and services of ACP Release 1.5 and presents some performance figures.	

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102 LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

OFFICE OF ACADEMIC COMPUTING

University of California at Los Angeles
405 Hilgard Avenue,
Los Angeles, California 90024

"Continued Development of Internet Protocols under the IBM OS/MVS Operating System"

FINAL TECHNICAL REPORT

TR46

Robert T. Braden

January 25, 1985

Sponsored by

Defense Advanced Research Projects Agency (DoD)
ARPA Order No. 4823

Under Contract MDA903-83-C-0435 issued by
Department of Army, Defense Supply Service-Washington,
Washington, DC 20310.

SUMMARY

This document is the Final Technical Report under DARPA contract MDA903-83-0435, entitled "Continued Development of Internet Protocols under the IBM OS/MVS Operating System". Under this contract, the contractor has developed and distributed network software for the DoD Internet protocols, to execute on an IBM 370 mainframe under the OS/MVS operating system.

The central component of this network software is a subsystem called the ARPANET Control Program or ACP. The ACP supports the principal user-level protocols -- TELNET for remote login, FTP for file transfer, and SMTP for electronic mail.

This report describes the features implemented at all protocol levels of the ACP (Release 1.50) and presents the results of performance measurements on the code. Finally, there is a discussion of ACP facilities which will require further development.

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

This document is the Final Technical Report under DARPA contract MDA903-83-0435, entitled "Continued Development of Internet Protocols under the IBM OS/MVS Operating System". This contract was in effect from 9/1/83 through 1/21/85.

Under this contract, the UCLA Office of Academic Computing (OAC) has performed development work on network software to support the DoD Internet communication protocols on an IBM 370 mainframe using the OS/MVS operating system. The central component of this software is a subsystem called the ARPANET Control Program or ACP.

The specific objectives of the contract were to:

* Update the prototype TCP/IP implementation within the ACP.

  OAC had developed a prototype TCP/IP implementation for an IBM MVS host under a previous DARPA contract[1] as part of the Internet research effort on the design of these protocols. The prototype implementation was grafted onto the ARPANET host software developed at OAC beginning in 1970.

  At the start of the present contract, the Internet protocols (which had just become DoD standards) had evolved beyond the state of development of the MVS prototype. A number of protocol features were missing from the prototype (e.g., options, ICMP, and UDP), while others were out of date (e.g., FTP had not been updated to the revised specification for TCP).

  Under the present contract, the prototype was to be updated to current protocol specifications and converted to production-level software.

* Implement Important User-Level Protocols

  In particular, FTP and SMTP were required for full function.

* Improve the performance, reliability, maintainability, and exportability of the MVS ACP.

---

[1] Contract MDA903-74C-0083.

DARPA desired the MVS code to be available to government sites and vendors, for use on the Defense Data Network (DDN). This implied a giant step in removing obsolete code, clarifying interfaces, simplifying the installation of the ACP, and documenting its structure and operation.

At the end of the contract period, major progress had been made towards all these objectives. In particular:

* The ACP has been completely overhauled and the obsolete host-host protocol removed, creating a pure TCP/IP communication package.

* Extensive documentation has been prepared on the ACP.

* Over 25 copies of the ACP distribution tape have been shipped in six releases of the software.

* The ACP is in operation on three DDN hosts, with more expected. Several vendors are making the ACP the basis for their products.

This report contains two sections. Chapter 2 contains a general overview of the ACP and a specific list of features at the present stage of development. Chapter 3 describes a preliminary performance study on the ACP.

We conclude this introduction with a brief history of the development of the ACP prior to the beginning of the contract.

The ACP is the product of 15 years of development and evolution of an ARPANET communication package for an IBM mainframe. In 1970, OAC began development of the ACP for its IBM 360/91 under the operating system OS/MVT [Braden77]. The most important changes in the ACP during the succeeding 15 years have included the following:

* Hardware Change

  The ACP, originally written for an IBM 360 CPU, was later moved to an IBM 370 with an expanded instruction set.

* ARPANET Access Protocol Change

  The ARPANET access protocol changed from 32-bit leaders (8-bit host numbers) to 96-bit leaders (24-bit host numbers).

* Operating System Change

  OAC changed its operating system from OS/MVT to OS/MVS. OS/MVT supported batch processing and a time-sharing system called TSO, using real memory. Although TSO under MVT was largely compatible with batch processing, the batch and interactive environments were quite distinct. MVT/TSO swapped users in and out of fixed partitions of real memory.

In contrast, each MVS job, either batch or interactive, executes in a separate 16M-byte virtual address space, managed with demand-paging. MVS/TSO is integrated with batch processing from the viewpoint of the operating system interfaces.

Installing OS/MVS had little effect on the much of the ACP. However, the ACP interfaces to the operating system and the interprocess communication mechanisms were completely changed.

* TCP/IP Development

As we mentioned earlier, OAC participated in the DARPA Internet research program which led to the present TCP and IP protocols. Development of a prototype implementation of TCP/IP for OS/MVS began in 1978. The prototype implementation was inserted into the ACP in parallel with the existing host-host protocol [FeiPos78] implementation, with a nearly-compatible transport-service interface for the user-level protocol modules [Braden79].

* Full-Screen 3278 Support

Interactive terminal support under the ACP was extended to allow full-screen 3278 operation with both User and Server TELNET. In full-screen operation, the hardware data stream used to drive the IBM 3278 terminals is sent transparently across the Internet, using the binary mode of TELNET.[2]

* NSW Development

OAC participated in the National Software Works (NSW) development [MCA77,Braden76,BraLud76,BraLud80]. Under DARPA and Air Force research contracts, OAC developed software to make the IBM system an NSW "Tool Bearing Host". This work included implementation of the transaction-oriented protocol MSG within the ACP [RivLud77].

The ACP which resulted from all these changes had accumulated a lot of history and some unnecessary complexity. It was poorly documented and contained features which were specifically tied to the UCLA environment. Its installation required the updating of parameters in many modules and a formidable sequence of assemblies and linkedits. The present contract has addressed these problems.

---

[2] This work was supported by the Federal Systems Division of IBM.

Chapter 2

STATUS OF ACP DEVELOPMENT

The UCLA ARPANET Control Program or ACP is a communication subsystem for the DoD Internet protocols, executing on an IBM 370 mainframe under the OS/MVS operating system.

The ACP includes all the Internet-specific protocol code -- the local network I/O driver, code for the host-host protocol TCP/IP, and the programs for the user-level protocols. The ACP supports the standard user-level protocols:

* TELNET for remote login,

* FTP for file transfer, and

* SMTP for electronic mail.

The ACP can be installed under either MVS/SP or MVS/XA with no operating system modifications; interprocess communication is accomplished with IBM's ACF/VTAM.

Most of the ACP, including the TCP/IP code, is written in 370 Assembly Language. However, the SMTP modules and a few other components are written in "C", and the user interface programs that execute under TSO are written in PL/1. The "C" code was compiled using the C/370 compiler distributed by AT&T Bell Laboratories.

The ACP is in the public domain, with two exceptions:

* The sources for the full-screen 3278 extensions to the User and Server TELNET programs are controlled by the Federal Systems Division of IBM.

* The sources for some of the "C" library routines are part of the proprietary C/370 compiler distribution, and can be released only to sites with a valid C/370 license from AT&T.

2.1 ACP COMPONENTS

Figure 1 shows the Internet protocol hierarchy implemented by the ACP. We will briefly review these protocols.

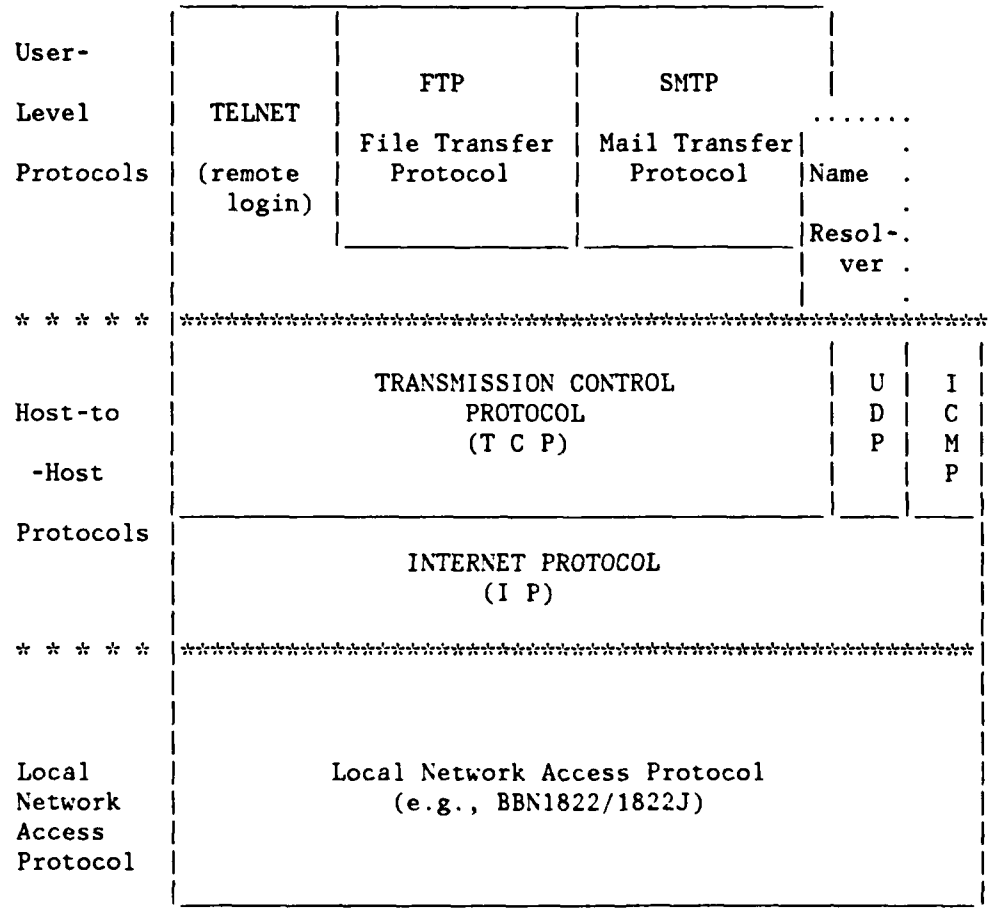


Figure 1: ACP Protocol Hierarchy

* IP -- Internet Protocol

IP provides datagram service in an internetwork environment, sending "Internet datagrams" between hosts which may be attached to different networks, linked by packet-switching hosts called "gateways" [Postel81b]. An Internet datagram consists of an IP header followed by data.

From a host viewpoint, the IP protocol provides two principal functions: (1) Internet host addressing, and (2) fragmentation and reassembly of Internet datagrams, in order to accomodate networks with diverse packet sizes.

IP does not provide error control; depending upon the properties of the networks and gateways, a transmitted packet may be lost, delivered out of order, or delivered in duplicate.

* TCP -- Transmsission Control Protocol

TCP is a reliable end-to-end protocol for transmitting data between processes over connections (virtual circuits) [Postel81a]. TCP plays a central role as the transport service protocol used by most user-level protocols.

1. TCP sends data in messages called segments, each of which begins with a TCP header and is sent as an Internet datagram using IP.

2. TCP delivers data segments to a user reliably and in order. The data in these ordered segments logically form an undelimited stream of 8-bit data bytes or "octets".

3. TCP provides flow control on each connection, using a windowing mechanism. The sequence space is fine-grained -- each octet of the data stream is numbered.

4. TCP uses checksums to ensure end-to-end reliability. The receiver sends acknowledgments of correctly-received data, and the sender does timer-based retransmission of unacknowledged data.

5. TCP creates full-duplex connections whose ends are labeled with 16-bit numbers called ports. Thus, a TCP connection is defined by the set of four address parameters:

   (<local host address>, <local port>,

     <remote host address>, <remote port>).

   TCP allows the same local port on a given host to participate in any number of connections whose remote ends have differing (<remote host address>, <remote port>) pairs.

Thus, a server host's well-known port can participate in any number of TCP connections, as long as the user host's (host,port) pairs are each unique.

6. A TCP connection is inherently full-duplex, even if an application needs only a simplex connection. Furthermore, a TCP connection is allowed to be half-open indefinitely. Thus, a close request (<FIN>) only signals the end of data transmission in one direction; data flow can continue in the other direction until a matching <FIN> is sent. The connection will be fully closed and deleted only when both ends request its close.

* ICMP -- Internet Control Message Protocol

ICMP is really an extension of IP, carrying routing, congestion control, and error reports to hosts [Postel81c].

* UDP -- User Datagram Protocol

UDP provides datagram service between two processes.

UDP does not define connections as does TCP. However, UDP does have 16-bit port numbers just like TCP, and sending or receiving a UDP datagram therefore requires the same set of four address parameters that define a TCP connection. As a result, the UDP implementation is a lot like that of TCP, except UDP is much simpler.

* User-level Protocols

The three principal user-level protocols are TELNET (remote login), FTP (file transfer), and SMTP (electronic mail) [Postel81d, Postel81e]. See section 2.5.

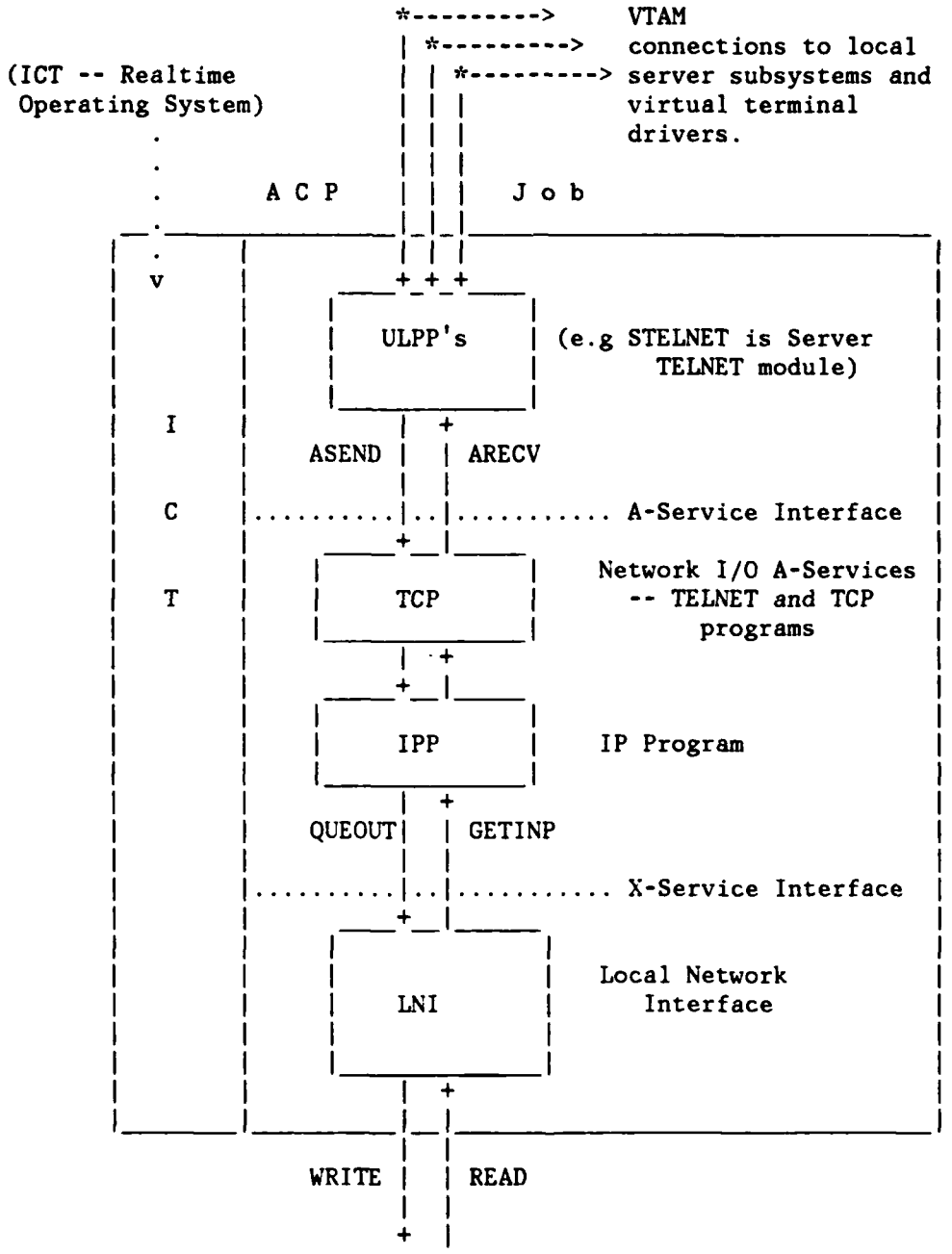
* Domain Name Resolver

A domain name resolver [Mockap83] will be required, but it is not included under the current contract.

We now begin to describe the ACP, which is organized in correspondence with the protocol hierarchy of Figure 1. Figure 2 shows the principal components of the ACP job.

* ICT Subsystem

The ACP normally executes as a batch job under the control of an internal real-time (sub-) operating system called ICT. ICT does multiprogramming, creating "pseudo tasks" or "ptasks" that are really coroutines. Its non-preemptive round-robin dispatching mechanism is often called a "commutator".



EXCP-level I/O to IMP Interface

Figure 2: ACP Job Components

The ACP uses a fixed set of ptasks to implement the local network access and host-host protocol layers. Additional ptasks are started dynamically to execute the appropriate user-level protocol modules for each active user or server session.

ICT provides a set of (sub-) system calls for operating system functions -- e.g., module loading, storage allocation, and dynamic file allocation. These system calls, which are actually subroutine calls through a transfer vector, are called "P-services" [Stein84].

Technical Report TR35, "Technical Overview of the UCLA ARPANET Control Program for OS/MVS using Internet Protocols TCP/IP" [Braden85a] includes an introduction to the facilities of ICT. Technical Report TR38, "ICT Version 2 -- Pseudotask Services and Macro Instructions" [Stein84] contains information on the operation and installation of ICT, plus a detailed description of all the P-Services of ICT.

* Local Network Interface (LNI)

The Internet protocols TCP and IP are used end-to-end, and in turn they use the transport mechanism provided by each network which is traversed. Hence, at the lowest level in the protocol hierarchy (see Figure 1 again), the ACP must handle the network access protocol (in ISO terminology, the link and network levels) for the particular network to which it is locally attached.

An ACP component called the "Local Network Interface" or LNI is the device driver for the hardware interface to connect the host to a network which is part of the Internet. The LNI is described in section 2.2 below.

* Host-Host Protocol Processing

IP and each of the higher-level host-to-host protocols (e.g., TCP and UDP) are handled by distinct protocol modules. ICMP processing is performed in the IP module (also known as the IPP or "Internet Protocol Program").

The user-level protocol programs open and close TCP connections and send and receive data using a set of subroutine calls within the ACP; these "A-Services" provide a tranport-service interface. The A-services are fully documented in TR21A, "Programming User-Level Protocol Processes for the ARPANET ACP" [TR21A].

Later sections of this document describe the current IP and TCP implementations.

* User-Level Protocol Processes (ULPP's)

The ACP includes a set of programs to implement specific user-level protocols, e.g., TELNET, FTP, and SMTP.

These "User-Level Protocol Processes" or "ULPP's" move data streams between the Internet (using the A-service calls) and the appropriate server subsystem in the host, The ULPP's are generally "protocol transformers" that match the Internet protocols to the IBM conventions.

Section 2.5 describes the various user-level protocol implementations.

* Interprocess Communication Mechanism -- ACF/VTAM

To perform its communication function, the ACP must transfer data to and from user programs or server subsystems, each of which is executing in its own address space. Thus, the ACP software requires some facility for cross-domain (or "interprocess") communication within the local system.

Under OS/MVS, IBM provides an interprocess communication mechanism named ACF/VTAM ("Advanced Communication Function/ Virtual Telecommunication Access Method") [IBMvtam]. ACF/VTAM creates connections or "sessions" between entities called "logical units" or LU's.

The original function of VTAM was to provide virtual-terminal capability, dynamically connecting terminal drivers ("secondary LU's") to server subsystems or "applications" ("primary LU's"). However, ACF/VTAM provides direct access to the underlying transparent interprocess communication facility of VTAM. The ACP uses ACF/VTAM to provide both virtual terminal support and interprocess communication [Rivas84].

Figure 3 shows the relationship of the ACP to VTAM.

1. The ACP acts as a secondary LU to allow a remote user to access a local server subsystem, e.g., TSO or CICS. The ACP interfaces to VTAM as a secondary LU using an OAC-developed package called the "Virtual Terminal Facility" or VTF,[3] which provides a simple get/put interface [Ludlam85]. The VTF creates virtual terminals, simulating both IBM 3767 (buffered typewriter) terminals and IBM 3278 displays.

   Consider a remote-login session through the ACP to a local server subsystem. A server ULPP passes data between the Internet and the VTF, performing the necessary transformations on character set and protocols. The data passes over a VTAM session to the appropriate server subsystem.

---

[3] In earlier documentation, the term "VLT" or "Virtual Line Terminal" was used instead of "VTF". VTF implies either a VLT or a VCRT (virtual CRT).

2. Conversely, the ACP acts as a primary LU to provide the internal communication path allowing a local user to gain terminal access to the Internet.

   An ACP process (ptask) named "VTAMAPPL" acts as a primary LU to VTAM, as if the ACP were itself a server subsystem [Rivas84]. A real or virtual 3767 or 3278 terminal, connected to VTAM as a secondary LU, can establish a session with VTAMAPPL as shown in Figure 3. A ULPP is again used to translate and transform the data as necessary.

User and server protocols in the ACP are discussed more in Section 2.5. The use of VTAM by the ACP is described in document TR37 [Rivas84].

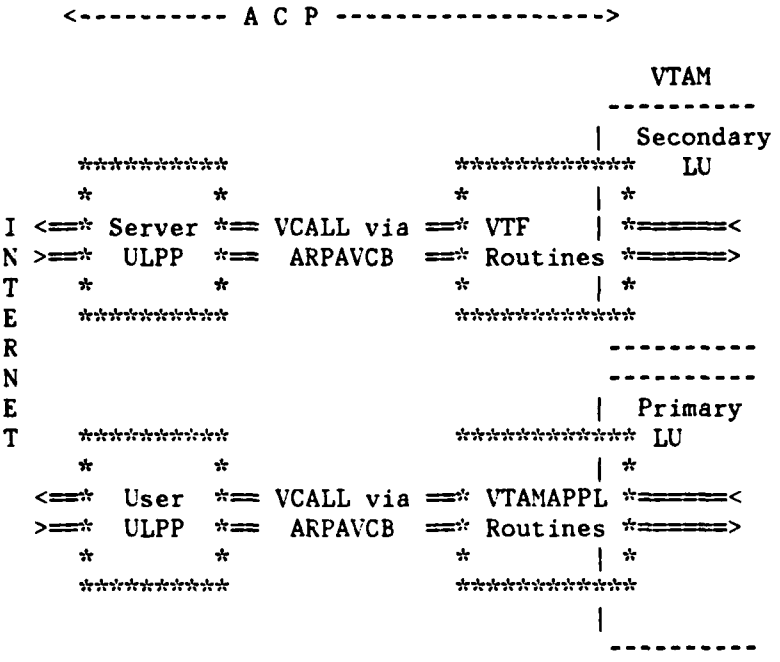


Figure 3: Terminal Sessions Using the ACP and VTAM

One facility which is not presently included in the ACP (but easily could be) is direct access to TCP connections from application programs in other address spaces.

This would use VTAM connections as transparent pipes to the ACP, and a new (but trivial) ULPP that simply passed binary data between VTAM and

the TCP connection. It would require a tiny control protocol out-of-band from the TCP stream, to specify the TCP address, pass status, etc. This facility has never been added simply because there has been no use for it. Only a newly-written network-oriented application would be able to take advantage of such a facility.

Some operating systems incorporate TCP into their kernel code in such a way that a TCP connection becomes simply a virtual file. Full integration of TCP connections in this manner with the normal sequential I/O mechanism of MVS would perhaps be a feasible goal. However, it not be desirable, because the ACP would then require serious operating system changes and be unmaintainable.

2.2 LOCAL NETWORK INTERFACE

The ACP as distributed by OAC contains a Local Network Interface (LNI) module named IMP1822. This module supports:

* The 1822 IMP-host protocol [BBN1822], and

* EXCP-level I/O operations to one of the two alternative channel interface boxes:

  1. The ACC-IF-IMP/370 channel interface built by Associated Computer Consultants, Inc. of Santa Barbara. This interface is restricted to a Byte Multiplexor channel.

  2. The HDH Host/IMP interface based upon an IBM Series 1, using interface boards built by Channel Systems, Inc. of Santa Barbara. This interface may use either a Block or a Byte Multiplexor channel.

We call these the "ACC" and the "CSI" interfaces, respectively.

The IMP1822 module has a number of important features:

* It handles the ARPANET internal flow control mechanism ("counting RFNM's") [BBN1822]. When output from the MVS TCP exceeds the limit of 8 outstanding packets to an ARPANET host, further packets to that host are queued until the RFNM count is reduced.

* Deadman Timeout Support

  It is capable of sending host-IMP NOP messages at an appropriate frequency when the interface is idle, to reset a "deadman" timer in the interface. This allows a suitably-designed channel interface to report to the network when the ACP is not functioning.

* Software Loopback

It can internally loop network output back into network input, so the ACP can be tested in isolation from the local netowrk. In particular, the ACP can be executed under control of the TSO interactive debugger TEST, allowing breakpoints to be used.

* Raw Packet Interface

  It provides a "raw packet" interface, allowing packets to be sent and received directly at the network level. The input side of the raw packet interface, called the "NMC Intercept", provides a general packet filter to select the packets to be monitored.

  The raw packet interface is used by the TSO processor ACPEEP to produce a real-time packet trace. ACPEEP has extensive filtering and formatting capabilities, providing an extremely useful diagnostic tool.

There are a number of deficiencies in the present Local Network Interface implementation.

* It does not support the logical addressing extension to the 1822 protocol, known as "1822L" [BBN1822L].

* The local network protocol functions are not cleanly separated from the rest of the ACP. Modifying the existing Local Network Interface or implementing a new one for a different local network access protocol would affect code in a number of ACP modules.

* There is no provision for multiple LNI instances to drive multiple hardware interfaces. Hence, it cannot handle multi-homing or parallel interfaces to increase bandwidth.

* Input packets are passed from the Local Network Interface to the IP code with an unnecessary buffer copy.

We hope to address these problems in a follow-on contract.

2.3 IP IMPLEMENTATION

Within the ACP, the Internet Protocol IP is implemented by the Internet Protocol Program or IPP. The IPP must support a number of different higher-level protocols, each of which is implemented by a corresponding higher-level protocol module (HLPM). The IPP accepts from the HLPM's segments of data to be sent, and it passes to the HLPM's the complete segments which have been received [Braden85b].

The IPP must implement the data transport functions of IP:

* Internet addressing

* Routing transmitted datagrams

* Reassembling Internet datagrams

* Demultiplexing reassembled datagrams

* Handling ICMP Redirect messages.

In addition to these IP functions, the IPP performs some control functions such as:

* Dynamically loading and deleting HLPM load modules and their resource pools

* Providing a timing service for HLPM's

* Creating new ULPP's in response to incoming and outgoing logger requests.

The design choices in the IP and TCP implementations were intended to support operation of a large IBM system using Internet protocols.

* The design favors a large number of simultaneous connections, with "fairness" criteria controlling the sharing of the buffer pools.[4]

* The IPP uses a hash table to perform demultiplexing of incoming packets. When no reassembly is required, a single hash lookup demultiplexes a packet directly to a particular TCP connection. This requires a controlled violation of the strict IP/TCP layering [Clark82a].

The various features of the IP protocol are currently implemented in the following manner.

* Fragmentation:

  The OAC implementation does not fragment, requiring its TCP to split outgoing data into segments of size suitable for the particular local network. Again, this requires a controlled layering violation.

  The lack of IP-level fragmentation will only be a problem if the ACP is adapted for use on a local network with maximum packet size less than 576. Then the use of TCP segments for fragmentation would increase the effective overhead due to TCP headers; also, UDP would require fragmentation for datagrams which did not fit into a single network message.

---

[4] Note that the existence of per-connection limits on the number of packets being sent has a desirable side-effect: it provides an implicit back-pressure on TCP when the local network flow control stops taking packets for a given remote destination.

* Reassembly:

The code performs reassembly of fragmented datagrams.

The present ACP reassembly code builds a linked list external to each reassembly buffer. It could have used a more efficient algorithm which builds the data structure within the reassembly buffers [DClark82a].

* Routing:

The first-hop gateway is chosen by the simplest (but most robust) algorithm:

> The ACP contains a preset table of initial gateway choices, in order of preference. A new locally-initiated connection is directed to the currently-preferred gateway; ICMP Redirects will correct this choice when necessary.
>
> If the preferred initial gateway is declared down by the ARPANET, the next in the list is made the preferred gateway; the list is used thus in a round-robin fashion.
>
> For a remotely-initiated call, however, the gateway from which the first packet came is used as the initial routing choice.

If the first packet arrives with source-routing, the return route in the packet is used as a source route for packets in the opposite direction.

Packets may be sent with source routes; a strict source route makes a fixed choice of first-hop gateway, which cannot be changed by a Redirect.

* ICMP:

This discussed below.

* Options:

Both loose and strict source route options may be either sent or received.

The security, timestamp, and return-route options are not yet supported (they are accepted but ignored).

* Identifier selection:

The IP module uses globally-unique identifiers for transmitted segments, independent of destination.

* Reassembly timeout:

  The IP module uses a fixed value (30-60 seconds), independent of the time-to-live field. Packets are discarded if their time-to-live field is zero.

* Gateway "pinging":

  The IPP assumes it is on the ARPANET and depends upon Host Unreachable messages when a neighbor gateway goes down. It does NOT probe its neighbor gateways to discover whether they are still up.

Most ICMP processing is performed within the IPP. The ICMP implementation has the following features:

* Redirect messages are acted upon.

* Unreachable and Source Quench messages are passed to the higher-level protocol (i.e., TCP).

* Echo Requests are turned into Echo Replies.

* ICMP Error indications are received and logged.

* Timestamp and Information Request/Reply are not currently implemented.

The IP input routine will send an ICMP Parameter Problem messages for certain packet errors.

2.4 TCP IMPLEMENTATION

The ACP implementation of TCP is contained in the load module TCPMOD, which is a particular instance of a HLPM (higher-level protocol module).

Specific features of the TCP protocol are handled in the following manner:

* Retransmission:

  Successive retransmission timeouts use "exponential backoff", starting with twice the measured round-trip time. Round-trip time is an exponentially-weighted average of the intervals between initial packet transmission and complete acknowledgment. This algorithm has become known as the "RSRE algorithm".

  The TCP retransmits slowly into zero window, as required by the protocol. It does not repacketize data in the retransmission queue.

* Window strategy:

The TCP uses a conservative strategy, never advertising a receive window larger than the space available in the circular buffer.

* ACK generation:

The TCP generally sends an <ACK> in response to the receipt of a non-empty packet. However, if received data can be sent immediately to a waiting process (ULPP), TCP tries to defer the ACK until the window has enlarged again.

As the user process removes bytes from buffer, an optimizing algorithm determines when to generate <ACK>'s to inform the sender of the enlarged window. As a result, the advertised window may sometimes be LESS than the available buffer space.

* Push:

The "Push" bit in data being sent forces TCP to forward the last data to the remote host. Since the Push bit is not a record marker, the ACP may collapse successive Push bits, through a process we call "promotion":

> If there is data queued for output but not yet fully-packetized that has the Push bit on, and if the user process issues a new ASEND call specifying PUSH, the earlier Push bit will be "promoted", i.e., moved to the new end of the queued data.

If the ACP receives TCP data containing the Push bit, that fact is made available to a user process. This could be useful in the User and Server TELNET ULPP's, for example, to aggregate data for transmission through VTAM. Unfortunately, current higher-level protocol implementations on other Internet hosts do not allow this optimization to be effective -- FTP and SMTP streams (ideally) never push data (except the last packet), while TELNET implementations tend to push every packet, even when packets are part of a burst of data.

* Data Aggregation

The algorithm used to packetize data to be sent by TCP aggregates data from successive ASEND calls, whenever the send window size and the Push bit allow it to do so. This provides efficient TCP transmission even if the ULPP is sending the data in small (un-Pushed) chunks.

* Urgent:

Urgent may be sent and received by a user process.

* Initial Sequence Number:

The Initial Sequence Number (ISN) for a new connection is derived from the system clock, as required by the protocol.

* Maximum Segment Size Option:

Receipt of the Maximum Segment Size option will cause larger packets to be sent when possible (the data is available and the window permits it).

It is recommended that the ACP be configured with TCP reassembly buffers at or near the maximum size of an ARPANET/MILNET packet (1007 bytes). If the reassembly buffer size is greater than 576 bytes, the ACP will send the Maximum Segment Size option when it opens a connection, to make use of the configured buffer size.

* New Sessions:

The ACP is effectively listening at all times on the "well-known" ports for supported server protocols. Receipt of a SYN for one of these ports will cause the incoming logger in the IPP to spawn a ULPP to service the connection.

In addition, a ULPP can issue a LISTEN for a partially-specified foreign socket.

* Performance

Like the IP layer, the TCP layer is designed to handle a large number of connections. It therefore attempts to minimize the number of timer interrupts by TCP, since these events create significant operating system overhead. Essentially, a timer is used only to trigger retransmission of un-ACK'd data.

This design decision is not without its drawbacks. Since it does not set a timer on input, the TCP implementation in the ACP tends to send more ACK's than necessary. A timer could also be used to improve the "silly window syndrome" (SWS) behavior [Clark82b] of the TCP implementation.

* Connection Errors

The Internet protocols TCP and IP are designed to support communication over paths with very unpleasant characteristics, including long delay, high packet loss rate, and massive failures of intermediate packet switches. We say that these protocols are "robust and survivable", to meet the requirements of a military environment.

One implication of these characteristics is that TCP connections can experience "soft" failures, in which transmission is halted but recovers fully after a (possibly long) period of time. During the failure period, the sending host will generally be receiving asynchronous ICMP error messages such as Destination Unreachable.

The proper treatment of the various flavors of connection errors is vital if the robustness and survivability of the transport mechanism is to be carried into the user protocols. For example, the desirable response to a particular mode of connection failure may be different if the user process is an FTP server than if it represents a human user doing remote login over a TELNET connection.

In the ACP, asynchronous connection-related error messages from ICMP or the local network are passed up to the TCP level for analysis and reporting. The TCP level passes "soft errors" up to the user level for further analysis and decision.

The user level can query the detailed nature of any connection errors using the ASTAT ERROR service. If the error occurs during the opening of a TELNET connection, an asynchronous exit to the ULPP conveys the connection error status.

The User TELNET program generally displays information on soft connection errors to the user, so the user can make the decision on when to give up. Server programs, however, allow a limited number of soft errors and then abort the session.

2.5 UDP

The User Datagram Protocol (UDP) is implemented in the module UDPMOD.

2.6 USER-LEVEL PROTOCOLS

Finally, we discuss user-level protocols, levels above TCP and IP, which provide the services to users. Figure 4 shows the user services currently supported by the ACP. Most of these protocols are based at least partly on TELNET, the Network Virtual Terminal protocol of the Internet.

2.6.1 Server TELNET

The TELNET Server may be used for access to any server subsystem[5] which drives a supported terminal type (see below) via ACF/VTAM. At UCLA, this includes TSO as well as the WYLBUR system marketed by Online Business Systems.

---

[5] A server subsystem is sometimes called an "application" in IBM terminology.

The supported (virtual) terminal types are:

* IBM 3767 typewriter terminals

* Locally-connected non-SNA IBM 3278 terminals

Either of these may be driven from a Network Virtual Terminal (NVT) to provide line-at-a-time operation for the remote user. The virtual 3278 may also be used in transparent full-screen mode from a remote IBM MVS or VM system.

When the remote user opens a TCP connection to the "well known" TELNET Server port (23), he/she is connected to a TELNET Server process (ULPP) in the ACP. If the user proceeds to LOGON to TSO, for example, the TELNET Server ULPP invokes the VTF which uses ACF/VTAM to make a cross-address-space connection to the virtual terminal handler for TSO. The ULPP makes all necessary conversions of code and protocols.

The TELNET Server also implements a few "pre-LOGON" services within the ACP. These include:

* HELP display

* NEWS display

* NETSTAT

  The NETSTAT program provides status information regarding the ACP. For system programmers, an alternate entry called SYSSTAT is provided that enables the ACP control functions in NETSTAT. The TELNET Server requires a local "LOGON" before allowing access to SYSSTAT.

  For detailed information on NETSTAT commands, see the TR42, "Intallation and Operation Guide" [TR42].

* ACTEST

  ACTEST is the ICT interactive debugger. It requires a local LOGON from the TELNET Server and is restricted to system programmers.

At the present time, there is an incompatibility between MVS and VM in their implementations of full-screen 3278 operation. When a user on VM contacts the MVS ACP on port 23 and selects a potentially-full-screen service such as TSO, VM is unable to negotiate full-screen operation at that point. To circumvent this problem, the distributed ACP is configured with a special contact port (1023) which connects VM users directly to MVS TSO to obtain full-screen 3278 operation. The ACP configuration can trivially be expanded to assign similar ports to other server subsystems.

User-Level Protocol	Service	Notes
TELNET Server	Access to TSO or other application service.	Applic. must support 3767 or local non-SNA 3278 through ACF/VTAM.
TELNET Server	Help, news, ACP status display.	Implemented within ACP.
TELNET User	Local user perform remote login to another Internet host ("passthru").	Access from: * Local 3278 (SNA/not), * Local 3767 * TSO command "TELNET".
FTP Server	File transfer server.	Contained completely within ACP job.
FTP User	User initiate file transfers between any two Internet hosts.	TSO command "FTP".
SMTP Server	Receive SMTP mail.	Print it, or pass thru JES2 to UCLAMAIL system.
SMTP User	Send SMTP mail.	Spool mail file from JES2 queue and send to Internet.

Figure 4: ACP Services

2.6.2 User TELNET

User TELNET allows local access to the Internet via the ACP. Local access can use either of two paths, which have different properties.

* Using the TELNET (or FTP) command processors under TSO.

  The TSO TELNET processor supports both line-by-line and full-screen terminals, but both are mapped into line-by-line NVT operation to the remote host.

There is a problem with this program from a human-factors viewpoint: TSO supports only half-duplex locked-keyboard operation, making access to character-by-character hosts somewhat awkward.

This program is written in PL/I, and it is a maintenance headache [TR42]. To use it, an installation must have the PL/I runtime library, an IBM Program Product.

On the other hand, this program has some useful features, e.g., saving typescripts and multiplexing simultaneous sessions.

* Directly from VTAM-supported 3767 or 3278 terminals.

Although the ACP 3278 terminal manager does not have all the features of the TSO TELNET program, it can access the 3278 with a true full-duplex unlocked-keyboard protocol.

The 3278 can either be mapped into line-by-line operation as a Network Virtual Terminal, or can operate in transparent 3278 full-screen mode to access a remote IBM MVS or VM server.

A desirable goal would be to upgrade the direct terminal entry module (VTAMAPPL) to provide all the features of the TSO TELNET program, and then drop the latter from further use.

2.6.3 File Transfer Protocol

The ACP supports the File Transfer Protocol (FTP).

* FTP Server

The FTP Server implementation is one of the most complete on the ARPANET; it includes block transmission mode, restart, and a comprehensive set of host-dependent options (such as DCB, VOL, and SPACE parameters). It also supports "anonymous" login for retrieval of public files.

Server FTP executes entirely within the ACP, using the dynamic allocation provided by the PDYNAL service of ICT.[6]

A thorough description of the FTP Server and all its features is contained in TR44, "The DoD Internet File Transfer Protocol in the UCLA ACP" [BraRiv84].

---

[6] At present, the ICT dynamic allocation can only be used with OS/MVS. It could be enhanced to work with MVT by replacing the SVC 99 calls with a suitable user-provided allocation routine.

* FTP User

  The user FTP command under TSO allows an OAC user to move a file from any ARPANET host A to any other host B, where A and B both have FTP SERVER programs; that is, it uses the "third party model". It operates by opening User TELNET connections to both FTP servers and sending the appropriate commands to each.

  This approach is very general but not extremely convenient for the common case that A or B is local. A more convenient user interface to FTP should be written.

2.6.4 Electronic Mail

Some key components for supporting electronic mail using the SMTP protocol have been developed experimentally, but require further refinement before they can be distributed.

* SMTP Receiver

  The SMTP Server SSMTP receives SMTP mail and spools it into a JES2 SYSOUT class. This program is written in "C" and adapted from a VAX UNIX[7] mail program written by the UCLA Computer Science Department.

* SMTP Sender

  USMTP was written as part of the ACP development under the present contract. Like the receiver, it is written in "C". It assumes that mail to be sent is available in cataloged datasets with a specific DSNAME prefix.

* Outgoing Mail Spooler

  This program, called SPOOL#3, copies output from a JES2 virtual 3770 printer into cataloged datasets. It is used to pass outgoing mail files to the SMTP sender. SPOOL#3 is written in BAL and uses the VTAM interface package called the "Virtual Remote Batch Terminal" or VRBT [Ludlam81]. The VRBT provides a virtual terminal (VTAM secondary LU) for a virtual IBM 3770 remote batch terminal.

For full-facility electronic mail service, an installation will need an appropriate user mail system. The SMTP modules in the ACP were designed to interface through JES2/NJE with a user mail system called UCLAMAIL. UCLAMAIL was also developed at OAC, and has interfaces to both BITNET and to the ACP.

---

[7] UNIX is a trademark of AT&T Bell Laboratories.

Without a user mail system, an ACP installation will be able to receive messages on the line printer, but not to send messages.

2.6.5 Remote Batch

Previous to the conversion of the ARPANET to TCP, OAC received batch jobs over the ARPANET using a remote job entry protocol called NETRJS.[8]

The NETRJS protocol provided a TELNET connection for a "remote operator", allowing the user to sign on as a (virtual) remote batch terminal. It had a user command language for obtaining job and system status and for controlling the operation of the session. Additional simplex connections were opened for transmitting virtual card reader, printer, and/or punch streams. Transmission could be in ASCII or EBCDIC and optionally use data compression.

The NETRJS server in the ACP interfaced as a virtual 3770 remote batch terminal to IBM's Job Entry Subsystem 2 (JES2) [IBMJES] through ACF/VTAM, using the VRBT [Ludlam81].

It would be feasible to convert the NETRJS server to TCP. However, this will not be useful unless NETRJS user programs are made widely available under a common operating system (e.g., UNIX).

2.6.6 Common TELNET Routines

The ACP includes a standard set of TELNET routines which are used by all ULPP's which need the TELNET protocol [Braden84]. These routines provide the ULPP's with a GET/PUT interface for sending and receiving character streams using the TELNET protocol. Internally, the TELNET routines call the connection I/O A-Services such as ASEND and ARECV.

The common TELNET routines perform a number of functions:

* Character Translation

  There are standard character translation tables in the ACP, used by all programs which need ASCII-EBCDIC translation. There is also a mechanism to allow a particular ULPP to override the default translation tables.

---

[8] See "NETRJS Protocol", NIC 42423 in the ARPANET Protocol Handbook [FeiPos78].

However, not all translation is 1:1. The most important case is the mapping between the ASCII CR LF sequence and the EBCDIC NL character. The TELNET routines also implement a rich set of escape sequences to handle the "problem mappings" between EBCDIC and ASCII.

* Output buffering

  The TELNET output routine provides a ring of buffers for the ULPP, and it handles the TCP "Push" function.

* Option Negotiation

  These routines implement the state machine to do option negotiation and sub-negotiation.

* TELNET Commands

  The TELNET commands are implemented.

* Urgent Processing

  These routines implement the discarding of data implied by the TCP Urgent facility, scanning for a Data Mark.

* Binary Mode

  These routines support binary transmission mode for sending data transparently, with the escape character X'FF' doubled.

* Tab Expansion

  The TELNET input routine can optionally expand a HT (tab) character using logical tab stops in every Nth column. The default for N is 8, but a ULPP can set N to any value.

2.6.7 Miscellaneous Services

The ACP contains three of the 'little' services (Echo server, Discard server, and Character Generator) for both TCP and UDP.

2.7 OPERATION AND MAINTENANCE

The ACP includes a number of useful tools for operation and maintenance. Interfacing between the IBM world and Internet worlds is not simple, and subtle problems may arise at all levels of protocol.

These tools include the following:

* Log Files

  The ACP produces a historical record of its activities, errors, and unexpected events, which it writes into a set of log files. These log files are normally printed with the ACP job output, and can be scanned in the running ACP using the ISPF processor of TSO.

  For example, the receipt of an erroneous or unexpected packet from a remote host will cause both the error text and a hex dump of the offensive packet to be sent to the log.

* NETSTAT

  This Server TELNET processor provides useful displays of connection and session status, and also a display of the recent messages written to the log files (which cannot be seen in ISPF, because they are still hidden in JES buffers).

* ACPEEP

  ACPEEP is a program running under TSO to format an ACP packet trace in real time onto the terminal. It has extensive filtering capabilities, so it can trace packets for a specific host and/or a specific protocol level -- local network, IP, TCP, or TELNET. For example, in the IP mode all IP datagrams are shown with the IP header formatted. In TCP mode, the TCP header is formatted.

  ACPEEP also has a number of parameters to control the display format for the data -- ASCII or EBCDIC conversion; display in "dump format" or as byte streams with escape sequences; and control of the amount of data to be displayed from each datagram.

  ACPEEP is written in PL/I, and uses VTAM to obtain raw datagrams from the ACP job.

* Connection Tracing

  There are several ways to capture a trace of the data on a TCP connection into a log file. It is possible to format the trace immediately and send it to a log file. Alternatively, the recent data on a TCP connection can be saved in a circular buffer and only formatted to the log file if a severe protocol error occurs.

  The SMTP programs similarly keep a circular trace buffer of their transactions. They dump this buffer to the log only if an SMTP protocol error occurs.

There is a "trace trap" mechanism, which will trigger an immediate log trace for a TCP connection to a selected combination of remote host address, local port, and/or remote port numbers.

* Connection Statistics

  Extensive statistics on each connection are printed in one of the log files whenever a TCP connection closes.

* Histograms

  The ACP has a built-in mechanism for building histograms of interesting distributions, and a histogram display command in NETSTAT.

  For example, histograms are kept of the (queueing) delays in sending packets out the local network interface and of TCP acknowledgment times.

* ACTEST

  The interactive ICT debugger program, mentioned earlier, is a powerful tool for diagnosing ACP bugs and for installing new modules.

  For debugging, the ACP can be run under TSO TEST, using the software loopback mode of the LNI. Both TSO TEST and ACTEST are then available. This testing can be done without disturbing the production ACP, using alternate ACB names defined for VTAM. The TSO interface programs TELNET, FTP, and ACPEEP all have parameters to select these alternate names.

Chapter 3

PERFORMANCE OF THE ACP

This Chapter describes the results of performance measurements of the ACP in operation on an IBM 3033 CPU under MVS/SP.

These measurements are particularly concerned with full-screen remote login using the transparent 3278 full-screen operation, since any large-scale use of the UCLA ACP is more likely to use full-screen than line-at-a-time operation.

A complete description of the performance of a communication package like the ACP would consider several different performance measures:

* CPU Usage

  Here we are concerned with the amount of mainframe CPU time that is consumed to drive a given number of concurrent sessions.

* Delays

  The ACP must provide acceptable delays to users at the maximum load level.

* Throughput

  For some applications (e.g., file transfer) the total throughput is more significant than delay.

The measurements reported here are mainly concerned with CPU usage, for several reasons. First, its measurement is simplest and most unambiguous. Secondly, in the regime we are concerned with, CPU time should be the primary limitation to supporting a large number of users. If we assume an 1822 IMP connection of 100KBits per second, this bandwidth should be far from saturated by the terminal load we are considering. As a result, we do not expect significant queueing delays for users, even with a large number of concurrent sessions.

Our measurements are focused on answering the following specific question: how much CPU time will be required in the ACP to support a population of N users, employing the ACP for full-screen remote login with TCP/IP?

In our measurements, we separated the CPU utilization into three categories:

* TCP Performance

* VTAM Interface and TELNET Performance.

* ICT Performance.

Each of these will be described in the following sections.

These measurements are limited in a number of respects.

* They were mostly made on a single session carrying N times the traffic of a real session, rather than setting up N sessions and measuring the aggregate. The assumption was that the ACP processing has only secondary dependence on the number of sessions. In fact, measurements on N simultaneous sessions reported later (see e.g., Figure 7) support this assumption fully.

* We considered only full-screen operation for which the ACP design is best suited -- because it uses large packets.

* We used a single average interaction rate, using parameters based upon typical measured user interaction rates [McKay84]. Specifically, we assumed an average interaction inputs about 200 bytes and outputs about 1500. In projecting performance in the last section, we will assume a rate of 0.02 transactions per second per terminal.

* We made only loopback measurements; i.e., in every case the UCLA 3033 was both source and sink for the data. That means the results contain a mixture of user and server operation. In making projections in the final section, we will make the untested assumption that the two ends make equal contributions to the total CPU time.

In spite of these shortcomings, we believe the results are consistent and reliable and are indicative to the real performance which can be expected. Future work on performance will serve to justify or disprove this belief.

We will present only a small sample of the data. In general, the results which we give are representative, and the measurements showed a great deal of consistency from day to day and run to run.

Our CPU measurements were based upon the sum of job step CPU time and SRB CPU time.

3.1 NETWORK PERFORMANCE

We assume that the CPU time required for sending and receiving data on a TCP connection may be written in the form:

CPU = A*S + B*N Seconds

Here S is the total number of segments processed, N is the total number of bytes of data transmitted, and A and B are constants defining the performance.

In general, our measurements are consistent with this formula, and we have been able to measure values for A and B which seem reliable.

Recent measurements have all used the following scheme:

* A traffic generator program (UTPTEST) is invoked as a ULPP, and told to connect to the discard server (port 9) on the UCLA host. This is accomplished from user TELNET by simply asking for "CCN,9;UTPTEST".

* UTPTEST opens a TCP connection to port 9 and sends a known stream of packets. At the end of this stream, it does a long PWAIT TIME to allow the connection statistics to be examined in NETSTAT. While packets are flowing, we monitor the CPU time and I/O count.

* Data is sent using ASEND and received using ARECV; the TELNET programs are not involved. Thus, the processing includes TCP and all layers below.

Figure 6 shows some typical results.

Note that in these tests the TCP acts perfectly, sending exactly one ACK for each data segment. An "ACK" is regarded as a segment containing zero data; thus, we take the total segment count S as:

S = #Segs + #ACKs.

If we plot CPU Time versus S for the first four values, we find a convincing straight line whose slope yields a value for the constant A (since N is constant). Similarly, we can determine values for B from either the intercept of this straight line, or from the second and fifth lines of the table. One gives B=1.5*E-6, the other gives 1.9E-6.

ASEND Size (bytes)	Total Data (bytes)	CPU Time (secs)	#IOR	#Segs	#ACKs
1*74	100,000	10.24	8155	1354	1354
3*74=222	100,000	3.44	2749	453	453
10*74=740	100,000	1.26	862	138	138
50*74=3700	100,000	1.03	713	111	111
3*74=222	200,000	3.59	2750	453	453

Figure 6: Typical TCP Performance Data

From these results and a number of similar tests, we obtained an empirical result for the 3033 CPU time needed to both send and receive S segments containing N data bytes:

LOOPED-BACK TCP CONNECTION

(Send+Receive)

CPU = (3.8E-3)*S + (1.9E-6)*N Seconds

That is, there is a cost of 3.8 milliseconds of processing time per segment, plus nearly 2 microseconds per data byte.

Note again that this includes BOTH sending and receiving; until we make measurements between two hosts we cannot separate the two.

Note that the S term dominates in every case. Even under optimum conditions, N/S does not rise above 200 data bytes per segment. The N term is never more than 10% of the total CPU time.

In an attempt to understand the surprisingly high per-packet cost, we repeated these tests with the ACP running in software loopback mode (IMP=NO). In this mode, the I/O operations to the IMP are simulated so there are no I/O waits. This will reveal how much of the CPU time is due to OS/MVS dispatching the ACP address space after I/O waits have completed. The result is:

CPU = (1.05E-3)*S + (2.4E-6)*N Seconds

Note that sending and receiving a segment requires three I/O event completions: the WRITE, the READ, and the READ for the RFNM (which is simulated even in IMP=NO mode).

That leaves in excess of 1 millisecond of CPU time per segment that we cannot blame on OS/MVS.

The next obvious question is: how much does the TCP checksum contribute? We repeated these measurements with IMP=NO and with the TCP checksum code dummied out. The result was:

CPU = (0.87E-3)*S + (1.2E-6)*N Seconds

That implies that the checksum contributes half of the per-byte processing, and 20% of the per-segment processing. The latter is a surprise; it would be desirable to make further measurements to corroborate this result and understand why the segment processing appeared to depend upon the checksumming.

Finally, we consider how the CPU time increases with the number of concurrent sessions K. From the design of the ACP we would predict that the CPU time would increase very slowly with C; the CPU time would thus depend almost entirely upon the total N and S, independently of how the processing is distributed across simultaneous sessions.

In particular, two important ACP processes that depend upon K are:

* Demultiplexing Incoming TCP Packets

  A single hash-table lookup demultiplexes each packet, creating a time increasing only very slowly with K.

* Dispatching a ready ptask under ICT

  ICT was designed to avoid a linear search of ptasks to determine the next one to dispatch, using instead a queue of "Ready" ptasks.

In order to verify this, we ran the network performance test described above using K simultaneous sessions, for a variety of values of K. That is, there were K traffic generator ptasks, K discard server ptasks, and K TCP connections. Each traffic generator ptask used an ASEND size of 50*74 = 3700 bytes, sending approximately 111 segments and receiving an equal number of ACK's. The results are shown in Figure 7.

K	CPU Time (secs)	#IOR	
1	1.03	713	(from previous table)
2	1.79	1390	
10	9.28	6987	
32	30.39	22325	

Figure 7: Performance with K Sessions

The CPU time is increasing faster than linearly; however, extrapolating linearly from K=10 to K=32 would predict 29.47 CPU seconds; the actual value is only 3% larger than this! We therefore feel justified in measuring the resource usage for a single session carrying K times the traffic of a real session and then interpreting it as a result for K sessions.

3.2 VTAM INTERFACE PERFORMANCE

Remote login through the ACP uses either one of two interface packages to interface to VTAM: VTF (Secondary LU) or VTAMAPPL (Primary LU). VTAM provides the virtual terminal communication across address spaces.

Of course, VTAM itself consumes CPU time, but we assume this depends only upon the terminal traffic, not on the identity of the programs generating/consuming that traffic. For example, we assume that VTAM needs the same CPU time, for either a real local terminal or the ACP acting as a Seconday LU accessing a given application program. Thus, we are interested only in the additional ACP CPU time needed to interface to VTAM.

We chose to measure a single composite VTAM scenario that again uses internal loopback and therefore lumps together user and server operation. Future measurements should be made of the two modes of operation separately. We also included a network path in the scenario, so to find the VTAM CPU we must deduct the TCP costs derived from the results of the last section.

The actual scenario is as follows (see Figure 8):

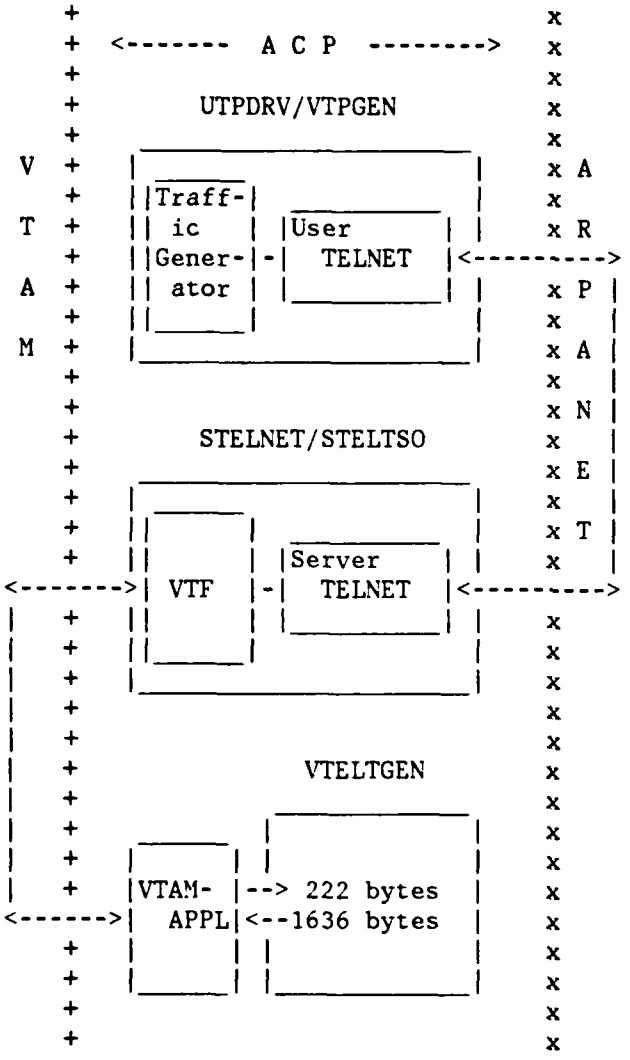


Figure 8: Schematic of VTAM/TELNET Loop Test

* A test driver program (which is a slightly modified VTELNET) called UTPDRV is started as a ULPP ("CCN;UTPDRV"). This program makes a TELNET connection to the local TELNET Server.

* In the TELNET Server, we issue the command "ARPA CCN;VTELTGEN". This makes a loopback connection through the VTF into VTAM as a Secondary LU, and connects back to the ACP as a Primary LU. There is invokes another modified version of VTELNET, called VTELTGEN.

* An escape command to UTPDRV ("@GO") enters a traffic generator (VTPGEN) which then conducts the test.

* VTPGEN generates a stream of 222 byte packets representing keyboard input. This packet goes out to the IMP and back into the STELNET/STELTSO ULPP's. Then it goes through the VTF to VTAM and back out to VTAMAPPL and then to VTELTGEN.

* Each time VTELTGEN receives a 222-byte packet, it sends back to VTAMAPPL/VTAM a 1536-byte packet representing a 3278 output screen. This data travels through VTAM, Server TELNET, the IMP, and finally back to VTPGEN, which just discards it.

Thus, this test involves two loop-backs: one through the IMP, and the other through VTAM. The measured CPU time is the total of the entire path, and represents the aggregate of a user session and a server session simultaneously.

It is possible to conduct this test with the entire path in NVT/virtual 3767 mode, or the entire path in full-screen 3278 mode. We give one typical result for the 3767 mode, but we are really interested in the full-screen case. Note that the particular choice of data sizes (222 bytes in, 1536 out) is appropriate only for full-screen operation.

Figure 9 shows data on this test. In every case, the (222 bytes in, 1536 bytes out) sequence was repeated 50 times.

Case	CPU Time (secs)	#IOR	S (user->srver)	N (user->srver)	S (srver->user)	N (srver->user)
1	3.18	1681	252	11100	302	76900
2	2.30	1084	153	11100	202	76900
3	2.75	1571	240	11100	280	76900

Figure 9: VTAM/TELNET Loopback Measurements

The three cases shown in Table 9 used different conditions, as follows:

* Case 1

VTPGEN was generating a packet every 0.5 seconds, so there was time to receive each screen response before the next keyboard stimulus was sent. While the resulting packet sequence was very orderly, we discovered that the maximum packet size through the IMP was only 512 bytes. This was due to the choice of parameters for the TELNET write buffer pool (4 buffers of 128 bytes each). It can be seen that for each interaction there were 5 packets sent in one direction and 6 in the other (this includes ACK's).

* Case 2

In this case, we increased the TELNET write buffers to 512 bytes each. A packet trace shows that the 1536 bytes in a screen is always sent in a maximum-sized ARPANET message (964 data bytes), followed by a smaller message for the rest. The total of seven segments per interaction is essentially optimum, and indeed the CPU time is the lowest of any test.

Note that this test is going at 2 interactions per second, which is the predicted rate for 100 terminals. However, our double-loopback test exercises both user and server sides, so it is comparable to 100 user and 100 server sessions simultaneously.

The total bandwidth required is 3400 bytes per second, and 10 segments per second. This is well within the capability of the 1822 interface and the ACP, so no queueing took place.

* Case 3

In this case, we increased the rate of sending stimulus packets from 2 per second to 10 per second. This created a demand for throughput that exceeded the capacity of the TCP connection, so the packet trace shows somewhat chaotic behaviour and sub-optimal behaviour of TCP in packetizing and acknowledging the data. As a result, the segment count and the CPU time increased.

We can now apply the empirical formula from the preceding section to subtract from these CPU times the cost of the TCP connection. The results are:

Case	Net CPU
1	0.90
2	0.77
3	0.59

We do not fully understand the variation in this table. The CPU time here is presumably due to VTF, VTAMAPPL, and the TELNET

processing. Note that in full-screen mode TELNET is sending binary data; this requires a TRT and an MVC(L) for each buffer, but no character translation.

However, we will take the worst-case figure of 0.90 seconds and ascribe it to the VTAM interface programs. These programs were handling 100 segments in each test -- half containing 222 bytes and half containing 1536 bytes.

We will again make an unsupported assumption that this time is divided equally between the VTF and VTAMAPPL, and infer:

VTF + VTAMAPPL:
CPU = (9.0E-3) secs per segment

3.3 ICT PERFORMANCE

The results above included the ICT overhead to dispatch ptasks (pseudo-tasks), so we do not need a separate measurement. However, to understand the region of validity we need some idea of what this overhead is.

A simple test involves two ptasks, one PPOSTing the other that simply loops on a matching PWAIT. This showed that each PWAIT/PPOST/dispatch sequence requires 53 microseconds of CPU time.

3.4 CONCLUSIONS AND EXTRAPOLATION

We can extrapolate from the preceding results to predict the CPU time required to support L terminals logged in simultaneously in full-screen mode.

Assuming an interaction rate of 0.02 per terminal, the rate of CPU usage would be:

```
Usage= 0.02*L*( (3.8E-3)*P/2 + (1.9E-6)*1700/2 + 9.0E-3)
```

Here the three terms represent (roughly):

* Per-segment cost of TCP transmission for a single interaction, assuming P IP segments are needed. The measurements earlier showed P ranging from 7 to 10.

* Per-byte cost of TCP transmission for a single interaction containing 200+1500 data bytes.

* Per-segment cost for VTAM interface processing.

Working out the arithmetic (assume P=10), these three terms become (respectively):

CPU Usage= 0.02*L*(19*E-3 + 1.6*E-3 + 9.0E-3)

= 0.6E-3 * L

For 100 terminals, this predicts 6% of the 3033 CPU.

Another interesting consideration is the amount of the CPU overhead caused by OS I/O reqeuests and interrupts. We showed earlier that these contributed (3.8E-3 - 1.05E-3) to the S term; in the Usage result above, this difference is about 0.27E-3 * L. Thus, nearly half of the total CPU Usage is due to OS I/O overhead. We have been measuring full-screen operation, which emphasizes big packets. With line-by-line operation, smaller packets are likely to make the OS overhead completely dominant.

How can we reduce the OS overhead? The best approach would be to have a "smart" interface to the IMP, which would:

* Allow aggregation of outgoing packets

  A framing would be introduced into the byte stream to delimit packets, without using Device End for each packet.

* Aggregate incoming packets

  The interface could aggregate multiple incoming packets into a single read, again using framing to delimit packets. It is particularly important to aggregate RFNM packets, since a RFNM is associated with every packet which is sent, and is quite small.

Chapter 4

CONCLUSIONS

During the period of this contract, the UCLA ACP was developed into an exportable implementation of the DARPA Internet protocols for an IBM mainframe. This report detailed the features (and dis-features) of the current level of the ACP.

In summary:

* The ACP now includes essentially-complete implementations of all the protocols -- IP, ICMP, TCP, UDP, TELNET, FTP, and SMTP. In most cases, any protocol features which have been omitted are unlikely to be be required for an IBM mainframe.

* The ACP is quite easily installed on any MVS system with no OS/MVS system changes, and only the assembly of a configuration table.

* The ACP is accompanied by extensive documentation.

The successful evolution of the earlier ARPANET host software into the present ACP was possible because of the choice of a general design model. In particular, the internal realtime (sub-) operating system ICT provided efficiency, flexibility and simplicity to the ACP code.

Many discussions of efficient implementations of TCP/IP have made the assumption that the code must either be in the kernel or in user space, and that the latter choice is doomed to inefficiency. The ACP doesn't exactly fall into either category, although it does run entirely in user mode. The use of ICT and careful attention to efficient algorithms in the ACP, plus the record-oriented I/O structure of the IBM system, have allowed the ACP to create reasonably low overhead even though it runs in user mode.

We have reported the results of an initial suite of performance tests on the ACP code. Specifically, we were interested in the question: if there are N simultaneous TELNET sessions using 3278 full-screen operation, how much CPU time will be required on our IBM 3033 mainframe?

Tests with N varying from 1 through 32 showed that CPU time for TCP/IP is very nearly linear in N; the N= 32 case exceeded 32 times the N=1 CPU time by only 3%. In short, there are no linear searches over connections or processes in the ACP.

TCP/IP CPU time was consistently expressible in the form:

$$CPU = A*S + B*N$$

where S is the number of TCP segments and N is the total number of data bytes both sent and received.

This preliminary set of tests had one important limitation: all tests used loop-back from UCLA to UCLA. Thus, the results always lumped together both user and server ends of the same TCP connection. If you assume these make equal contributions, the following results can be divided by 2 to get the overhead for a single TCP connection on a 3033 mainframe:

> Protocol processing up through the TCP level required A= 3.8 ms per segment sent and received, and about 2 microseconds per data byte sent and received. Of the latter, about 0.8 microseconds per byte was due to check summing.

An elaborate loopback test was performed through User and Server TELNET, and through ACF/VTAM, so the result for N sessions really corresponds to N server plus N user sessions.

The result showed that processing up through TCP/IP is small compared to TELNET and VTAM processing. Another test with the ACP running in software loopback mode without an IMP showed that about half of the total ACP CPU time is due to MVS overhead in processing I/O and timer interrupts.

The tentative conclusions of this study were:

* If you assume the rate of 0.02 transactions per second, all ACP processing will take 6% of a 3033 CPU for 100 simultaneous TELNET sessions (now we have divided by 2, assuming user and server sessions are equivalent).

* If you send single-character packets (as UNIX systems and TAC's often do), you will impact the IBM system very badly.

* You could improve the performance significantly by using a "smart" 1822 interface that aggregates/disperses packets using a single READ or WRITE on the IBM channel. However, for reasonable data aggregation (line at a time, or full-screens), there is not much performance to be gained by moving TELNET/TCP/IP processing into a Front End.

In the future, a number of areas of the ACP need to be addressed.

* More Performance Testing

  The tests reported here have a number of limitations and a few inconsistencies that should be explored. In addition, the testing should be extended to include the cost of the various user-level protocols.

Under the guide of the past and future performance tests, some performance improvements should be made in the ACP.

1. It may be possibl to further improve the performance of the TCP implementation, decreasing the number of ACK's which are sent and improving the Silly Window behavior. The challenge is to accomplish these improvements without using excessive CPU overhead for timers.

2. A "smart" local network interface which aggregates packets should be tried.

3. A facility for dynamically expanding the TCP buffers for a very-long delay path should be developed.

* Revised Local Network Interface

  It will be desirable to write new LNI modules for different local networks and interfaces. At the current ACP level (Release 1.50), this is difficult because the LNI is not cleanly separated from the rest of the ACP. The LNI design and interfaces need to be rationalized and documented. It is also highly desirable to provide for multiple hardware interfaces, to allow multi-homing and multiplexing of interfaces.

* Name Resolver

  An Internet name resolver should be implemented and installed in place of the current host name tables. This will require some reorganization of the host lookup A-Services and of some of the ULPP's which use these services.

* Remote Job Entry

  Future users of the ACP are going to want to perform remote job entry to and from remote hosts.

We believe that the present ACP provides a general and flexible environment which can be adapted to a variety of special application communication needs. It also provides the generality to allow easy extension to new protocols or to add new features to existing protocols.

Chapter 5

ACKNOWLEDGMENTS

It is a pleasure to acknowledge the dedicated efforts of OAC system programmers Lou Rivas, Neil Ludlam, and Denis DeLaRoca. Without their contributions, we could not have reached the present state of development of the ACP. The masterful design and elegant implementation of ICT/V2 by Michael Stein was also a vital contribution.

Finally, the TCP/IP developments since 1978 have been built upon the firm design foundation for the ACP, laid by Steve Wolfe and Stuart Feigin nearly 15 years ago.

REFERENCES

BBN1822 BBN, "Specifications for the Interconnection of a Host and an IMP". Report 1822, Bolt Beranek and Newman, Cambridge, Massachusetts, revised May 1978.

BBN1822L Malis, Andrew G., "The ARPANET 1822L Host Access Protocol". RFC878, December 1983.

Braden76 Braden, Robert T., "The National Software Works". Technical Report TR9, Office of Academic Computing, UCLA, December 9, 1976.

Braden77 Braden, R., "A Server Host on the ARPANET". Fifth Data Communications Symposium, Snowbird, Utah, September 1977.

Braden79 Braden, R., "An IBM 360/370 Implementation of the Internet and TCP Protocols -- Design Specification". Technical Report TR20, Office of Academic Computing, UCLA, December 1979.

Braden84 Braden, R., "TELNET Routines in the UCLA ACP". Technical Report TR43, Office of Academic Computing, UCLA, November 1984.

Braden85a Braden, R., "UCLA ARPANET Control Program for OS/MVS Using Internet Protocols TCP/IP -- Technical Overview". Technical Report TR35, Office of Academic Computing, UCLA, (revised) January 1985.

Braden85b Braden, R., "Interface Specification for Programming a Higher-Level Host-Host Protocol Using Internet Protocol". Technical Report TR19, Office of Academic Computing, UCLA, (revised) December 1985.

Braden86a Braden, R., "The Local Network Interface in the OS/MVS ARPANET Control Program". Technical Report TR51, Office of Academic Computing, UCLA, January 1986.

Braden86b Braden, R., "Datagram Protocol Implementations Within the UCLA ACP". Technical Report TR55, Office of Academic Computing, UCLA, January 1986.

BraLud76 Braden, R., and Ludlam, H., "A Tool-Bearing Host in the National Software Works". Technical Report TR10, Office of Academic Computing, UCLA, April 1, 1977.

BraLud80 Braden, R., and Ludlam, H., "Final Technical Report, National Software Works Contract". Technical Report TR27, Office of Academic Computing, UCLA, December 1980.

BraRiv84 Braden, R., and Ludlam, H., "The DoD Internet File Transfer Protocol in the UCLA ACP". Technical Report TR44, Office of Academic Computing, UCLA, June 1984.

Clark82a Clark, D., "Modularity and Efficiency in Protocol Implementation". RFC817, MIT, July 1982.

Clark82b Clark, D., "Window and Acknowledgment Strategy". RFC813, MIT, July 1982.

FeiPos78 Feinler, E., and Postel, J., eds., "ARPANET Protocol Handbook". NIC 7104, published for the Defense Communications Agency by SRI International, Menlo Park, California, revised January 1978.

IBMJES International Business Machines Corporation, "OS/VS2 MVS System Programming Library: JES2". GC23-0002 File No. S370-36

IBMvtam International Business Machines Corporation, "ACF/VTAM General Information: Concepts". GC27-0463 File No. S370-30.

Ludlam84 Ludlam, H. C., and DeLaRoca, D. W., "Virtual Terminal Facility (VTF) Support Package". Technical Report TR26, Office of Academic Computing, UCLA, (revised) October 1984.

Ludlam81 Ludlam, H. C., "Programmers' Guide to the UCLA Virtual Remote Batch Terminal Support Package". Technical Report TR25, Office of Academic Computing, UCLA, February 1, 1981 (latest revision: January 1986).

MCA77 MCA staff, "The A-Level System Specification for the National Software Works". Massachusetts Computer Associates, Inc., Wakefield, Mass., May 15, 1977.

Mockap83 Mockapetris, P., "Domain Names -- Implementation and Specification". RFC883, November 1983.

McKay84 McKay, D., Private Communication.

Postel81A Postel, J., "The DoD Standard Transmission Control Protocol". RFC793, September 1981.

Postel81B Postel, J., "The DoD Standard Internet Protocol". RFC791, September 1981.

Postel81C Postel, J., "Internet Control Message Protocol". RFC792, September 1981.

Postel81D Postel, J., "File Transfer Protocol". RFC765, September 1981.

Postel81E Postel, J., "Simple Mail Transfer Protocol". RFC821, August 1981.

RivLud77 Rivas, L., Ludlam, H., and Braden, R., "Implementation of the MSG Interprocess Communication Protocol". Technical Report TR12, Office of Academic Computing, UCLA, May 1977.

Rivas84 Rivas, L., and Ludlam, H., "ACF/VTAM Usage by the ARPANET Control Program". Technical Report TR37, Office of Academic Computing, UCLA, (revised) October 1984.

Stein84 Stein, M., Rivas, L., and Ludlam, H., "ICT Version 2 Pseudotask Services and Macro Instructions". Technical Report TR38, Office of Academic Computing, UCLA, (revised) December 16, 1984.

TR21A Braden, R., Rivas, L., and Ludlam, N., "Programming User-Level Protocol Processes for ARPANET ACP (REVISED)". Technical Report TR21A, Office of Academic Computing, UCLA, Revised November 1984.

TR42 Rivas, L., Braden, R., and DeLaRoca, D., "Installation and Operation Guide for the UCLA ARPANET Control Program -- Release 1.50". Technical Report TR42, Office of Academic Computing, UCLA, Revised December 1984.