Bolt Beranek and Newman Inc.

Report No. 4856

HP3000 Internet Software: Final Document

January 1982

Prepared for:
Defense Advanced Research Projects Agency

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) HP3000 Internet Software Final Document		5. TYPE OF REPORT & PERIOD COVERED Final Report 6. PERFORMING ORG. REPORT NUMBER 4856
7. AUTHOR(s) Jack Sax		8. CONTRACT OR GRANT NUMBER(s) MDA903-80-C-0214
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 10 Moulton Street, Cambridge, MA 02238		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE January 1982 13. NUMBER OF PAGES 90
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Defense Supply Service - Washington Rm 1D 245, The Pentagon Washington, DC 20310		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release/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)

Transmission Control Protocols (TCP); Internet Protocols (IP); HP3000 Intrinsic; ARPANET Intelligent Network Processor (INP).

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

This report is the final document for implementing TCP on an HP3000 computer connected to the ARPANET.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

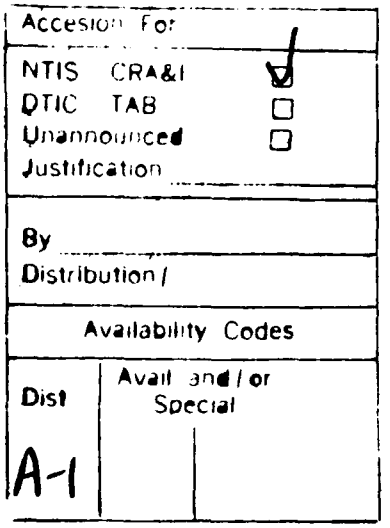
Report No. 4856

Bolt Beranek and Newman Inc.

HP3000 Internet Software:

Final Document

January 1982



Prepared for:

Defense Advanced Research Projects Agency

Table of Contents

1 Preface

This report is the final documentation of the software used to implement the Internet protocols on the Hewlett Packard HP3000 Series III computer system. Specific protocols implemented include a Transmission Control Protocol (TCP), Internet Protocol (IP), File Transfer Protocol (FTP), and TELNET Protocols.

The reader is assumed to be familiar with the purpose of these protocols and have a user-level understanding of the MPE operating system.

2 Introduction

The Internet protocols are implemented in five layers with each layer performing specific functions. Protocol layers one through four represent the system layers which are responsible for moving a message reliably from one host to another. The fifth protocol layer consists of a number of applications protocols which determine the content and meaning of the messages exchanged.

Protocol levels one and two are X.25 LAP link access protocols. These protocols are implemented in microcode on the Intelligent Network Processor (INP) interface available from Hewlett-Packard. Since the X.25 LAP protocols are different from the standard 1822 IMP Host protocols, a special X.25 IMP interface is used to link the HP3000 with the ARPANET. The interface divides standard 1822 packets into a number of X.25 frames and transfers each frame separately. The diagram in Appendix A shows the hardware configuration used to link the HP3000 to the ARPANET.

The next two protocol layers consist of the DOD standard Internet Protocol (IP) and the Transmission Control Protocol (TCP). The Internet protocol provides communication between Hosts on different networks via gateways between the networks. The Transmission Control Protocol provides reliable transmission between Hosts and performs some Host-to-Host flow control.

Both the TCP and IP protocols have user level interfaces which allow the applications protocols to send TCP byte streams and IP datagrams. These interfaces are implemented through a set of system commands which can be called from any user program.

The initial implementation includes three application layer protocols. One of these is the File Transfer Protocol (FTP) which allows a user to move files from one computer system to another. The second and third application layer protocols are User and Server TELNET. User TELNET gives the user a remote terminal capability by taking the characters from the local input device and sending them to the foreign host, and returning characters from the foreign host to the local output device (typically a terminal). The Server TELNET acts as a pseudo-Teletype, with incoming network messages providing TTY input, and TTY output being sent to the network. The operating system treats the Server TELNET pseudo-Teletype like an ordinary terminal.

Most of the protocol software is new code. the major exception being the INP microcode which is supplied by Hewlett Packard. The programs are written in HP's Systems Programming Language (SPL), which resembles PASCAL and allows intermixing of assembly code and compiled code. In addition to new code. implementation required one change to the MPE operating system code. This change will be incorporated into HP's standard

operating systems releases.

3 Current HP3000 Structure

This section describes the HP3000 system with an emphasis on the features that affect the network software design. The description includes both the processor hardware and the operating system.

3.1 Processor Features

The HP3000 CPU is a medium speed machine which uses a stack architecture. It executes uncomplicated instructions in one to two microseconds. Code and data are separate and thus all code is re-entrant. There are approximately 38 hardware registers which make up the state of the processor; most of these are associated with the stack (data) and the current instruction address (code).

Memory is divided into segments. A segment is a contiguous block of memory of any desired length up to 32K words. Individual segments are swapped in and out of memory as needed. Memory paging, a scheme which uses fixed size memory chunks as the basis for memory swapping, is not used in the HP3000. A segment may be designated as code or data by the operating system.

3.2 Network Interface Hardware

The interface unit between the HP3000 computer and the ARPANET machines is HP's Intelligent Network Processor (INP). This device consists of two boards located in the HP3000 main cabinet. It is a microprogrammed interface unit whose microcode is down-line loaded by HP3000 software. HP supplies the microcode to make the INP obey the X.25 LAP protocol and the device driver necessary to access the INP.

The INP is connected to a BBN C/30 IMP via an RS422 cable.

3.3 Operating System Software

The operating system for the HP3000 is known as the Multiprogramming Executive System (MPE). It offers both batch and interactive job capabilities and allows multiple concurrent users of either type. It offers a file system which manages files on disk, magnetic tape and/or punched cards. Some I/O devices, such as the line printer, have spooler programs built in to the system.

User programs are run as processes within MPE. Each process has associated with it a code segment and a stack (data) segment. In privileged mode it may run in "split-stack mode", where it is allowed to have two data segments. The most common use of split-stack mode is to access tables in the operating system

during system calls.

The design of MPE is greatly influenced by the HP3000 hardware architecture. MPE's organization relies heavily on operations which incur little processor overhead while avoiding operations which incur large amounts of processor overhead. The most striking example of this is the MPE's dependence on user processes for a large number of what would ordinarily be considered system functions. MPE avoids the use of "system" processes to perform these functions.

The design organization is a direct result of the stack architecture of the HP3000. The large number of status registers which must be saved when a new process is invoked makes process switching a very expensive operation. The time needed to perform a procedure call into a new segment of system code is typically less than the time to switch context from one process to another. Writing efficient code for this machine has thus led to organizing the system as relatively independent "utility" routines callable by the user rather than as a collection of separate processes which manage I/O devices and system utilities. These operating system calls, named Intrinsics, are implemented as subroutine calls into system code segments. The program segments which implement the Intrinsics run in a privileged mode which allows them to directly access system tables and I/O device tables.

One notable side-effect of this design is that system resources such as I/O devices are assigned to only a single program and are not normally shared. This approach has allowed the system programmers to create a complex operating system without tackling the problems of interprocess communication and resource sharing. As will be discussed later, it also has a significant effect on protocol software design.

3.4 Input/Output

Input/Output operations typically consist of two steps. The first step is initiation of the desired operation. This involves checking to insure that access to the device is allowed (software protection), and issuing I/O instructions to the device to initiate the desired action. This step usually occurs as a result of an intrinsic call to the device handler code and thus is executed on the user's stack. The second step is the operation completion handling. This may occur using either the Interrupt Control Stack (ICS) or the System Control Stack, neither of which is the user's stack. The choice of which stack to use depends on the specific device's function.

A consequence of this system design is that "system code" tends to be executed using the data stack of the first user process needing the function. If process 1 wants to do an I/O operation, it invokes a system procedure which knows how to

manage that I/O device. If process 2 now wishes to invoke the same device, and if the device is capable of supporting more than one request concurrently, it invokes the same routine. To avoid multiprocessing hazards in issuing I/O commands, the system procedure first checks to see if it is the first invocation of itself -- if not, it queues the request and exits; if it is, it proceeds to issue the I/O instructions. If the request was queued, it is assumed that the first process will detect the newly queued request and process it also. The first process is thus performing system functions for the second and all later processes, and will be charged run time for doing their work. In practice, we do not expect this to be significant, but in theory, the first process could run indefinitely, even if its own request had long since completed.

3.5 Interprocess Communication

Interprocess communication under the current version of MPE is a problem. Only two techniques are currently available and neither of them is really satisfactory.

One technique that may be used is that of the logical device. It is chiefly used to accomplish multiplexing of physical devices. This facility is implemented by creating a new entry in the system's Device Information Table, and by creating a set of procedures which act as a device handler. The handler

will be run in privileged mode.

Like other system device handlers, the procedures to manage the device are invoked directly by the user process. and the user's stack is used by the system code. This has the advantage of speed, since it avoids some process context switching.

There are a number of drawbacks to this technique. First. the Device Information Table entry must be maintained as if it were a real hardware device. This requires knowledge of all the MPE internal functions that might access this table. Furthermore. since these tables are system internal, they are subject to change with each new release of MPE. Use of the table requires Privileged Mode. Bugs in the code would have a greater chance of crashing the system. The greatest drawback is that logical devices are still under development at HP, and are more than usually likely to change over time.

A new operating system feature. not yet released officially, that has been written for MPE is an interprocess communication method known at HP as message files. These correspond to UNIX ports, and allow unrelated processes to communicate with one another. Each message file has one or more "reader" processes and one or more "writer" processes. During use. these files act as FIFO queues.

Message files are implemented using the file system. Read, write. and query commands are all patterned after the file system commands. The message file code is designed so that if readers and writers stay more or less in synchronization, disk I/O will not be needed. However. if the writers get far enough ahead of the readers, the message file will start being spooled out onto disk.

Message files are to be introduced as user level functions by HP, and, as such. their use will not change with new releases of the operating system. Code for this feature has already been implemented at HP and is available with both MPE III and the future MPE IV. They appear to be relatively easy to use and do not require knowledge of the internals of the operating system. Their chief drawbacks are that a process context switch is required between writer and reader, and that some file system overhead is incurred.

Timeouts, as seen in message files, are another new HP function that will be available. The older version of timeouts simply suspended the process for a fixed amount of time. but did not allow the process to be awakened by the completion of an I/O event during its sleep. The new version is equivalent to setting a timer whose alarm may be awaited with the same IOWAIT intrinsic that awaits I/O completion. It allows a process to wait for either some I/O device operation completion or the passage of

some maximum amount of time. whichever occurs first. Alternatively. a timeout could be used to insure that waiting for a specific event will terminate if the expected event does occur soon enough. There will be both user level and system internal ways of accomplishing timeouts.

3.6 Existing INP Software

The code to drive the INP is part of the CS/3000 Communications Software package from HP. It contains code to send and receive packets via the INP and code to manipulate the Device Information Tables. The code also allows the user to down-line load microcode into the INP memory. It contains intrinsics to open and close the line and to read and write packets. The microcode allows the INP to support X.25 LAP protocols and also allows the INP to buffer up to eight 128-byte packets. These packets are read by CS/3000 as soon as possible in order to keep the INP from losing packets due to a lack of buffer space in the INP. This technique allows the INP to function as a full duplex device, even though the MPE operating system offers only a half duplex control mechanism in its software.

4 Protocol Software Architecture

The protocol software architecture is dictated by a set of design requirements and MPE operating system constraints. These requirements and constraints are summarized as follows:

- The new network software must be isolated from the existing operating system as much as possible. The isolation will allow any site to add or remove the network software with a minimum of effort. It will also make the network software less vulnerable to any changes HP makes to MPE.

- Efficient high speed network communications are extremely important because this TCP version will be used on a production rather than an experimental basis.

- One of the problems with MPE is that, though the operating system performs device assignment and access control for its I/O devices, the user process is responsible for operating the I/O device. MPE does offer intrinsics to operate common devices, but these are very low level operations. This I/O arrangement makes it difficult to control an asynchronous network interface. The protocol software architecture will therefore require at least one process which has exclusive control of the INP interface.

- One of the properties of these network protocols is that the message acknowledgments and retransmissions occur at a

relatively high level -- in the Transmission Control Protocol in layer four. A moderate amount of time passes from the time the originating TCP queues the message for transmission and the receiving TCP gets the message. In order to prevent acknowledgment delays which in turn cause the foreign host to retransmit data, the software architecture should minimize the amount of time it takes for incoming data to move through the 1822, IP, and TCP protocols.

- With many network users and many connections concurrently in use, the network software must be able to handle the problems of multiplexing use of the network interface hardware. The interface on which the multiplexing takes place must support a number of simultaneous users in such a way that the behavior of any individual user does not affect data throughput of the other users.

In order to meet all of the design requirements and constraints described above, the HP3000 protocol software is implemented in a set of processes (see diagram in Appendix B). One process which will be called the system protocol process is responsible for maintaining the INP interface as well as supporting the 1822, IP and TCP protocols. The rest of the processes, called applications protocol processes, support the user interactive network functions including FTP and TELNET.

The use of a single system protocol process is a key element in the protocol design. The system protocol process provides control over the INP interface by providing buffers and acting as multiplexer and de-multiplexer of network traffic to and from the INP. Use of a single process minimizes inter-protocol layer communication delays which in turn minimize the acknowledgment delays for incoming data. A single system protocol process makes it possible to use interprocess communication primitives to provide a uniform network interface for the applications level protocol processes.

User TELNET and User FTP protocols are to be implemented as ordinary user programs. They use the same system calls as any other network accessing program, but are written to provide a higher level command language for the user. As user programs, they execute in the user's address space with the privileges normally available to the user. The User TELNET and User FTP programs are re-entrant, with as many processes running this code as users wishing the service.

Server TELNET is a single process created as the system starts up or whenever the first need for it arises. De-multiplexing of Server TELNET inputs is accomplished via a pseudo-teletype driver. The driver acts as the interface between the Server TELNET process and the Teletype handler.

The interface between application protocol processes and the system protocol process is through a set of IP and TCP intrinsics. The intrinsics are designed to form a uniform interface between the user and the IP and TCP. Actual data communication between a user process and the system protocol process is done with a combination of message files and direct buffer-to-buffer transfers. Message files are used to pass flow control information while the actual data transfer is made by copying data between user buffers and system protocol buffers. The combination of message files and buffer copy is used to take advantage of the flexibility of message files and the data rates achieved by direct data copy.

5 System Protocol Software

Since this TCP/IP protocol implementation is to be used on a production rather than an experimental basis, the design effort has concentrated on the efficiency rather than the sophistication of the protocol software. At the same time. the software design does allow for the future enhancement. There are no inherent design limitations which will prevent implementation of the more sophisticated TCP and Internet features.

5.1 Implemented Features

The specific TCP and Internet features to be implemented include the following:

- multiple connections to multiple hosts,
- flow control at the 1822, Internet, and TCP layers,
- error recovery,
- fair allocation of resources among all connections,
- handling of urgent data,
- surviving incorrect packets,
- datagram reassembly,
- a user level interface for both the IP and TCP protocols,
- routing.

5.2 Software Architecture Overview

The system protocol software architecture reflects the need to avoid packet processing delays rather than a strict hierarchy between protocol layers. The system protocol software is implemented as a single process to allow the system protocol layers to share software resources for greater efficiency. The shared resources include subroutines which perform functions required by more than one protocol layer, and a common buffer pool to optimize storage resources and to allow efficient communication between protocol layers.

Network traffic through the system protocol process takes different forms including 1822 packets, datagrams, and TCP segments. These various forms are generically referred to as "packets." Packets are passed into the system protocol process from either an applications protocol process or the ARPANET interface. Packets from the ARPANET are passed into the system protocol process by intrinsic calls to the INP interface. User generated network packets are passed to the system protocol process by using a combination of message files and data buffers. Message files are used to transfer control and status information, while data transfer is done with buffer-to-buffer copies between the user protocol data segment and the system protocol data segment.

All read and write commands are done without wait to allow the system protocol process to simultaneously multiplex I/O channels and process network packets. I/O multiplexing is implemented through the IOWAIT intrinsic. The system protocol process issues an IOWAIT intrinsic after it finishes processing a data packet. The IOWAIT intrinsic returns the file number of the I/O channel associated with an I/O completion wakeup.

When the number of free buffers falls below a prescribed limit, an attempt is made to free buffers through data compaction. The attempt begins with a search for datagram fragments and unacknowledged TCP segments which waste buffer space by using only a fraction of the available space in each buffer assigned to them. This lack of efficiency can be particularly damaging because there is no guarantee that the data contained in the buffers will ever be processed. Wherever possible, datagram fragments are combined into a single datagram fragment and TCP segments are combined into a single segment to more efficiently utilize system buffers. Any buffers freed by this compaction process are returned to the free list.

Network packets from both the user process and the ARPANET are processed along one of a number of data paths in the system protocol process. The actual data path taken depends on the type of data packet and, in the case of TCP segments, the state of its associated network connection. Packet processing is performed by

a series of function calls which act as processing steps along the data path.

In order to avoid processing delays which can tie up system resources, each arriving data packet is processed through as much of the protocol software as possible. Processing of a packet is suspended only when the lack of some resource or some external event prevents further processing.

5.3 Control Structures

All of the status information both for individual network connections and for the system protocol software as a whole is kept in a set of control blocks as well as in a number of buffer list structures as shown in Appendix C. The control blocks include a general network resources control block, a foreign host control block for each foreign host connected to the local host. and send and receive control blocks for network connection. The list structures include a network buffer free list, a TCP buffer aging list, an Internet buffer aging list. and a timer queue list. The relationships between the various control blocks and the list structures are shown in the first diagram in Appendix C.

5.3.1 Network Resources Control Block

The Network Resources Control Block contains the information needed to maintain the network buffer free lists and aging lists. This information includes pointers to the network buffer free lists and aging lists and a count of the buffers in each of the lists.

The information contained in the Network Resources Control Block is used by the protocol software to control the distribution of network buffers among the various lists. The information is scanned at various times to determine the allocation or disposition of a particular network buffer. The determinations occur when new buffers are allocated from the free list and when buffers containing TCP segments are about to be acknowledged. Decisions are made based on the number of free buffers available and the priority of the task requiring the buffers.

5.3.2 Foreign Host Control Blocks

Foreign Host Control Blocks are used to maintain flow control within the 1822 protocol layer. The information contained in this block includes the foreign hosts IMP and HOST identification numbers, a count of the number of TCP connections to this host. a count of the number of outstanding 1822 packets

which have not been acknowledged with a RFNM, and the 1822 message id for each of the outstanding 1822 packets.

The counter is used to prevent transmission of too many 1822 packets to a single host. All transmission to the foreign host is blocked when the counter reaches the limit of eight outstanding 1822 packets to that host.

5.3.3 Connection Control Block

Each TCP connection has an associated control block. The control block contains data associated with the Transmission Control Block (TCB) along with other connection related information. Specific information included in the control block is shown in the diagram in appendix C.

5.3.4 Network Buffer Resources List Structures

Three list structures are used to maintain the network buffer resources shared by all of the sockets. These list structures include the free list and the two buffer aging queues.

The network buffer free list contains all of the network buffers currently available for use by any socket. These buffers are allocated when new data comes in from either the network or a user protocol process.

The Internet Aging Queue is a list of active buffers assigned to blocked datagram fragments and complete datagrams. These buffers are the first to be reclaimed when there are no free buffers available. The Queue is sorted according to datagram age. All buffers which belong to the same datagram are combined into a single list structure. The datagram list structures are linked into the Internet Aging Queue with the least recently updated datagram always at the head of the queue. When a new datagram fragment comes in it is moved to the end of the queue along with all of the other fragments which belong to the same datagram.

The TCP Aging Queue is a list of buffers which contains at least parts of unacknowledged TCP segments. These buffers can be reclaimed when there are no free buffers and no buffers on the Internet aging list. The Queue is sorted by socket. All buffers which contain data for the same socket are combined in a buffer list and each buffer list is linked into the queue. The queue is sorted by the age of the data associated with each socket. Data belonging to the socket which has been inactive for the longest period of time is placed at the head of the queue so it can be recycled first. When a user process reads data from a connection, all the network buffers still waiting to be read on that connection are moved to the end of the TCP aging list. This assures that data associated with an active TCP connection will not be recycled ahead of data associated with an inactive TCP

connection.

5.3.5 Timer Event Queue

The Timer Event Queue is a linked list of buffers which contains information about future protocol events. These events include the various transmission timeouts used by the protocols. Each event buffer is linked into two list structures, one master list structure includes all timer events for all connections the other list includes all timer events for a particular TCP connection. The event buffers are sorted in chronological order with the earliest event as the first element in each list. A special message file is used to schedule a interrupt for the first element in the master list. When the timer goes off the event is executed and the event represented by the second element in the master list is scheduled.

5.3.6 Data Message Buffer Structure

All data messages to and from the network are stored in the buffer structure shown in appendix C. The first buffer in the structure contains the 1822 header and the overhead information needed by the various protocol layers to process the message. The second and succeeding buffers contain the actual 1822 data which includes the TCP and IP headers as well as any TCP data.

In addition, each buffer has a one-word HDH header used to identify the buffer type and the amount of data in the buffer (see BBN Report 1822).

6 The User Process Interface to the TCP and IP

The interfaces between the user process and the TCP and IP protocol software are designed to meet two basic requirements. First, the interfaces must support high speed data transmission between the user and system protocol layers; this is especially important since these interfaces involve interprocess communication which could be delayed by excessive system overhead due to context switching and process scheduling. Second, the interfaces must isolate the system protocol process from any buffer overhead burdens caused by processing delays in the user process. System protocol process buffers are too valuable a resource to be locked into storing TCP segments or IP datagrams which are waiting for response from a user process.

High speed data transmission across the interfaces are achieved by copying data directly from buffers in the user process to buffers in the system protocol process. The direct transfer is implemented with the move-to-data-segment and move-from-data-segment instructions provided by the HP3000.

The system protocol process is isolated from delays in the user process by making the user process responsible for buffering TCP data segments and IP datagrams. Acknowledged incoming TCP segments, and TCP segments and IP datagrams waiting to be transmitted over the INTERNET, are stored in buffers in the user protocol process. This use of user buffers serves two functions:

it frees system protocol buffers from being locked into storing TCP and IP data. and also gives the user process some control of network connection throughput. Throughput control is accomplished by allowing each user process to choose the amount of buffer resources it dedicates to each connection.

6.1 Interface Intrinsics

The interfaces between the TCP and IP software and the user process are implemented through a set of intrinsics. These intrinsics allow the user process to create and use network connections with other processes on foreign hosts.

Twelve intrinsics provide the basic control functions needed to transfer data through the TCP and IP interfaces. The TCP intrinsics allow the user to create network connections with other processes on foreign hosts. Each connection consists of a pair of sockets as defined in the TCP protocol document. Connections are created with a TCP'OPEN intrinsic whose parameters define the sockets which make up the connection. After a connection is created, the user process uses the TCP'SEND and TCP'RECEIVE intrinsics to send or receive data. The TCP'STAT intrinsic allows the user to check the status of a connection.

Within the user process. connections are identified through the combination of a connection file number and a connection

buffer. The connection file number is returned by a successful TCP'OPEN call. The connection buffer is an integer array allocated by the user process. The buffer is initialized by the TCP'OPEN intrinsic and is then passed as the first parameter to all of the other TCP intrinsics. It is the responsibility of the user process to maintain the association between the connection file number and the connection buffer.

The IP intrinsics allow the user to send IP datagrams to other processes on foreign hosts. Communication is initiated with the IP'OPEN intrinsic whose parameters define the type of datagrams the user process will accept. While there is no such thing as an IP connection, the IP'OPEN intrinsic creates a pseudo connection to handle the datagram traffic to and from a process. Once the pseudo connection is created, the user process uses the IP'SEND and IP'RECEIVE intrinsics to send or receive data.

The TCP and IP interfaces are entirely asynchronous so that a user process can queue any number of read or write requests to a particular connection. The user process has three limitations in this regard: first. it must provide the buffers associated with each intrinsic call; second, the user process must keep track of which buffers are associated with each I/O call; and third, the user process cannot dequeue buffers until it has permission to do so from the system protocol process.

The user process uses a combination of the IOWAIT and the NET'COMPLETE intrinsics calls to keep track of I/O completion events. The IOWAIT intrinsic is invoked when the user process has completed processing all of the current data. When the IOWAIT intrinsic returns with a file number associated with a TCP or IP connection, the NET'COMPLETE intrinsic is called to handle the I/O completion. The NET'COMPLETE intrinsic uses the connection buffer to determine the cause of the I/O completion and then performs the appropriate I/O cleanup task and returns an I/O-type code to the user process.

This is a summary of the TCP (Transmission Control Protocol) and IP (Internet Protocol) intrinsics, their arguments, and the errors that may arise. A negative number returned by any of these intrinsics indicates an error; zero or positive numbers indicate successful completion. Standard HP notation is used.

```
  I                    IA      IP        IP
errnum := TCP'Abort(UIbuf,Rbufptr,Sbufptr);
```

UIbuf is the connection buffer detailed below.

Rbufptr will return either NULL or a pointer to the first receive buffer freed.

Sbufptr will return either NULL or a pointer to the first transmit buffer freed.

This intrinsic will abort the connection indicated by the contents of UIbuf. It will wait until the connection is aborted before returning. If the abort went smoothly 0 will be returned,

otherwise an error code will be returned.

```
   I                IA     DV  IV IV IV  IA        O-V
filenum := TCP'Open(UIbuf,FHIA,FP,LP,AP,baddr);
```

UIbuf is a connection block whose format is shown below.

FHIA is the Foreign Host Internet Address.

FP is the Foreign Port number.

LP is the Local Port number.

AP is a word indicating the type of open:
- 0 Active open (initiate contact with foreign host)
- 1 Partial listen (nonspecific wait for foreign connection)
- 2 Active listen (wait for specific foreign connection)

BADDR is an optional argument that may specify a buffer for the reception of data. If supplied, the buffer size will be used as the initial window size for the connection. If not supplied, the initial window size will be zero.

TCP'Open initiates a TCP connection on an unassigned port. When this intrinsic returns, a connection has been defined in the system. but the connection may not yet be open to the foreign host. The value of FILENUM is the value that will be returned from IOWAIT when there is activity regarding the connection. It is up to the user program to keep track of this number.

If a connection cannot be opened, an error code will be returned instead of a file number.

```
  I                    IA      IA  LV
errnum := TCP'Send(UIbuf,baddr,eol);
```

UIbuf is the connection block described below.

BADDR is a single buffer of data to be transmitted. See below for buffer format.

EOL is the end-of-letter flag. If TRUE. the end of the buffer passed will be marked with end-of-letter.

TCP'Send will queue one buffer at a time for transmission over the connection indicated by UIbuf. If transmission is allowed and the buffer is successfully queued 0 will be returned; otherwise an error code will be returned indicating the reason for the failure. BADDR will be set to zero if the buffer has been taken by TCP'Send.

This intrinsic does not wait for data to be sent. Buffers whose data have been sent and acknowledged are returned to the user via the Net'Complete intrinsic.

```
   I                     IA     IA
errnum := TCP'Receive(UIbuf,baddr);
```

UIbuf is the connection descriptor shown below.

BADDR is a buffer for receiving data whose format is detailed below.

TCP'Receive will queue the buffer provided for data reception. The window size advertised to the foreign host will be incremented by the size of the buffer. If successful 0 is returned, otherwise an error code will be returned. BADDR will be set to zero if the buffer is taken by the intrinsic.

This intrinsic does not wait for data to be read. Buffers with incoming data are received via the Net'Complete intrinsic.

```
  I                    IA     IA
errnum := TCP'Status(UIbuf,Sarray);
```

UIbuf is the connection descriptor detailed below.

SARRAY is an array to which the status information is returned.

TCP'Status reads parameters describing the state of the connection into the array SARRAY. If successful, 0 is returned, otherwise an error code is returned. The format of the array is:

Sarray Offset	Symbolic Name	Symbol value	Contents
0	status'locport	0	Local port number
1	status'fport	1	Foreign port number
2-3	status'fhost	1	Foreign Host Internet address
4-5	status'Rwindow	2	Receive window
6-7	status'Swindow	3	Send window

```
  I                  IA
errnum := TCP'Close(UIbuf);
```

UIbuf is the connection descriptor block shown below.

TCP'Close will initiate closing of the transmit side of the connection. Further data transmission on this connection will not be allowed. The connection should not be considered actually closed until the appropriate code from Net'Complete has been received.

If closing the connection could be initiated, 0 will be returned, otherwise an error code will be returned.

```
   I                        IA     IA     IP   IP   I   I
fcncode := Net'Complete(UIbuf,msgbuf,Rbuf,Sbuf,Rbp,Sbp);
```

UIbuf is the connection descriptor block.

MSGBUF is an array (0:10) used to receive signals from the TCP process. The message received is interpreted by Net'Complete.

Rbuf is nominally a receive buffer pointer returned to the user, or NULL.

Sbuf is nominally a transmit buffer pointer returned to the user or NULL.

Rbp may be a byte pointer into the receive buffer Rbuf or unchanged.

Sbp may be a byte pointer into the buffer Sbuf or garbage.

Net'Complete is used to interpret the message from the TCP network process read by the user program's IOWAIT. Depending on the message, Net'Complete may want to signal the user that some event of interest has occurred. The function code returned indicates what event has occurred and also determines how the various arguments are to be interpreted. The function codes are:

ClosedTR Transmit and receive sides of the connection have both been closed. The connection no longer is defined in the system. and all attempts to access it will be invalid.

RBUF points to the head of a chain of receive buffers;

SBUF points to the head of a chain of transmit buffers;

If there are no buffers to be returned, the pointer

will be set to NULL. RBP and SBP are not modified. The buffer chains terminate with a NULL.

ClosedT The transmit side has been closed and the closing acknowledged by the foreign host. Data may continue to arrive on the receive side of the connection.

SBUF points to the head of a chain of transmit buffers. If there are no buffers to be returned, SBUF is set to NULL.

ClosedR The receive side has been closed. No further data will be received. Transmission of data may still be allowed.

RBUF points to the head of a chain of receive buffers. If there are no buffers to be returned, RBUF is set to NULL.

Note: When the "first" side closes, only CLOSEDR or CLOSEDT is returned as appropriate. When the "second" side closes, the system will note that the other side is already closed and return CLOSEDTR.

DSent One or more buffers has been transmitted to the foreign host and, if TCP, acknowledged as received. The buffer(s) are returned to the user.

SBUF In TCP, points to the head of a chain of transmit buffers. The chain is terminated by a NULL pointer. In IP, a single buffer (not a chain).

DRcvd Data has been received from a foreign host. If TCP, there will be at least one byte of new data. If IP, a datagram has been received.

If IP the intrinsic argument returned is

RBUF pointer to a buffer containing the datagram received. The length of the datagram will be found in ubo'datagramsize in the buffer overhead. The buffer is returned to the user.

If TCP the intrinsic arguments returned are

RBUF pointer to the buffer containing the first byte of new data;

RBP byte pointer to the first byte of new data in RBUF;

SBUF pointer to the buffer containing the first byte past the end of the new data;

SBP byte pointer to the first byte beyond the end of the new data.

Buffers from RBUF to SBUF are chained through ubo'link in the usual manner. The buffer pointed to by SBUF is still owned by the system. but all other buffers from RBUF to SBUF are being returned to the user. If there is urgent data. the urgent pointer in ubo'urgent of the buffer overhead words will indicate the last byte of such data.

ConAbort The network connection has been aborted. The connection is deleted from the system and all further operations on it are invalid.

RBUF points to the head of a chain of receive buffers;

SBUF points to the head of a chain of transmit buffers; if there are no buffers to be returned, NULL is returned.

ConOpen Connection open signal. The connection has been established with the foreign host and data transmission may now actually begin. If a passive listen was specified in TCP'Open, this signal means that it is now legal to queue data for transmission. No parameters are returned.

ConRefused Connection was refused. The connection attempt with the foreign host has been rejected. The user should close or abort the connection. No parameters are returned.

ConRetran Excessive retransmissions are occurring on the TCP connection. This is purely a notification. and no specific user action is required. Retransmissions will continue unless the user aborts the connection.

NOP No operation. Occasionally the network process will signal the user process for reasons purely internal to the network intrinsics. No action is required of the user program; no parameters are returned. The user program may return to the IOWAIT as if nothing had happened.

```
  I                        IV      BA
length := Net'Error(errorcode,line);
```

ERRORCODE is the error code returned by any of the other TCP/IP intrinsics.

LINE is a byte array into which a text string describing the error will be written.

LENGTH is the length of the error message in characters (positive count).

The error code returned from a TCP/IP intrinsic will be interpreted and a descriptive message written into LINE. The length of the message will be returned as the value of the intrinsic. If the error code is not recognized, a message to that effect will be returned in LINE.

For a list of the types of errors that may be encountered, see the section below entitled Network Intrinsic Error Codes.

```
  I                    IA    DV      IV
filenum := IP'Open(UIbuf,FHIA,protocolnum);
```

UIbuf is the connection block described below.

FHIA is the foreign host internet address, or zero.

PROTOCOLNUM is the internet protocol number to use.

IP'Open initiates an IP connection on an unassigned port. When this intrinsic returns, a connection has been defined in the system. and both sending and receiving may begin. If there is a problem opening the connection, the error will be returned via Net'Complete.

The value of filenum is the value that will be returned from IOWAIT when there is activity regarding the connection.

```
  I                  IA     IA
errnum := IP'Send(UIbuf,bfradr);
```

UIbuf is the connection block described below.

BFRADR is the address of the first word of the datagram to be sent.

IP'Send will queue one buffer at a time for transmission over the connection indicated by UIbuf. If the buffer is successfully queued, BFRADR will be set to zero. If a problem occurs, an error code will be returned.

The intrinsic does not wait for the buffer to be sent. Datagrams that have been sent are returned to the user via Net'Complete.

The internet source address (in'srcadr) will be set by the system. as will the internet header checksum. Any previous values in these fields will be lost.

```
  I                     IA     IA
errnum := IP'Receive(UIbuf.bfradr);
```

UIbuf is the connection descriptor block described below.

BFRADR is a buffer for receiving datagrams.

IP'Receive will queue the buffer for receipt of a datagram. If the buffer is successfully queued, BFRADR will be set to zero. An error code will be returned if a problem arises.

Datagrams are passed to the user one datagram per buffer. If the datagram is longer than the buffer provided, the excess bytes will be lost. The actual number of bytes placed in the buffer will be returned in UBO'DATAGRAMSIZE. If the datagram received is shorter than the buffer provided, the remainder of the buffer will be unused and unchanged.

This intrinsic does not wait for data to be read. Incoming datagrams are received via the Net'Complete intrinsic.

```
  I                    IA
errnum := IP'Close(UIbuf);
```

UIbuf is the connection descriptor block described below.

IP'Close will initiate closing of the IP connection. Buffers may not be queued for either sending or receiving on this connection after closing has been initiated. Data already queued for transmission, or datagrams arriving (if there are receive buffers queued), may still proceed. If closing could not be initiated, an error code will be returned; if successful, IP'Close will return zero.

A user program that uses the TCP/IP intrinsics should include the following lines of code near the beginning of the program.

```
$include comdefs.source.network
$include netdefs.source.network
```

The files contain the symbol definitions needed to use the intrinsics. The TCP/IP intrinsics may be defined with the intrinsic statement, such as

```
intrinsic (netintr.pub.network) tcp'open.tcp'close;
```

The symbols used in this document are the same as the ones defined in the files just described.

6.1.1 UIbuf format

The user program will need to have one UIbuf array for each open connection desired. The format of this array should be of no concern to the user. It is manipulated solely by the network intrinsics. Altering the contents of this array by other means could result in the process aborting or the connection being terminated. The user may read the contents of the array if desired.

```
integer array UIbuf(0:uib'len-1);
```

Offset	Value
uib'Cn	Connection block index number
uib'ICBseg	Interface Control Block data segment number
uib'tfrin	Resource Identification number for network
uib'vw	User stack verification word
uib'Smsgfile	File number of TCPMF.PUB.NETWORK
uib'msgfile	File number of private message file

6.1.2 MSGBUF format

This is an integer array (0:10) that is used in IOWAIT as the destination buffer for messages from the TCP network process to the Net'Complete intrinsic. It is a scratch region and has no useful data as far as the user program is concerned. The contents of this array should not be modified between the exit from IOWAIT and the call to Net'Complete, but may be modified any other time without harm.

One MSGBUF array will suffice for all connections combined. Since the array is used only between the IOWAIT and Net'Complete calls, it is, in effect, shared by all connections.

6.1.3 Transmit and Receive buffer overhead format

The buffer overhead for transmit and receive buffers is roughly identical. The same symbols are used for both. Buffers are linked into chains through the link word UBO'LINK. Buffer chains are terminated by a value of NULL in the link field.

Offset	Value
ubo'link	Link pointer to next buffer, or NULL.
ubo'len	Length and status flags. Subfields are:
ubo'inuse	Buffer-in-use flag;
ubo'eol	End-of-letter flag (TCP only);
ubo'byc	Buffer byte count (size of data area);
ubo'urgent	Urgent data indicator (TCP only);
ubo'datagramsize	Number of bytes in received datagram (IP only);
ubo'data	First word containing data.

The in-use flag is set and cleared by the network intrinsics for the convenience of the user program, but is not otherwise important. The EOL flag is not manipulated directly by the user program. The flag is changed by the TCP'Send intrinsic according to the EOL parameter. The flag is not used on the receive side. The byte count field indicates the size of the data area, in bytes, in the buffer. The count is not modified by the network intrinsics. When transmitting data via TCP, if the urgent pointer is greater than zero, the buffer is assumed to contain urgent data. If negative or zero, it is assumed the data is not urgent. In receiving TCP data, the urgent pointer indicates the number of bytes of data in the buffer that are considered urgent, starting from the beginning of the buffer. If zero or negative, the data is not urgent. When receiving IP datagrams, the total length in bytes of the received datagram will be placed in UBO'DATAGRAMSIZE. This field is ignored on the transmit side.

Offset UBO'DATA in the buffer contains the first byte of data space in the buffer. Assuming a pointer to the buffer is in

the variable BADDR, a byte pointer to the first data byte is

```
(@BADDR+ubo'data)&LSL(1)    or    @BADDR(ubo'data)&LSL(1)
```

The buffer size, in words, is given by the formula

```
ubo'data + (BADDR(ubo'byc) + 1) / 2
```

6.1.4 Network Intrinsic error codes.

The following error codes are used by all the network intrinsics. Error codes are always negative integers. The major error class is a multiple of -100, and the error subclass, if any, is a number from 0 to -99. The error mnemonics given below are listed in the file COMDEFS.SOURCE.NETWORK.

Error code	Meaning
NoNet	Network process or network interface block missing.
NoConnections	Cannot open another connection at this time. All available connection blocks are in use.
IllegalArg	An illegal argument was supplied. If the subclass is zero, an option-variable intrinsic was not supplied as a required argument. If the subclass is less than 10, it indicates which argument is bad. If the subclass is 10-99, the high digit indicates which argument has bad format, and the low digit indicates which item is at fault. Typical errors are buffers whose byte count is zero, or buffers with internal structure whose internal pointers are inconsistent or null.

NoMsgFile	Could not open one of the private message files.
CantWriteMF	Could not perform FWRITE on a message file.
SendClosed	Transmit side state does not allow sending of data at this time. This will happen if connection is closed or if opened with a passive listen and the ConOpen has not yet been received.
RcveClosed	The receive side is closed.
WrongCnType	A TCP intrinsic was called for an IP connection, or vice versa. If the subclass is zero, UIbuf passed describes a TCP connection; if the subclass is one, UIbuf describes an IP connection.
InternalError	The intrinsics have detected an inconsistency in something. Things may be in a very bad state. Report this to a systems person with as much detail as possible.
SignalLost	A TCP message to the user has been lost or mangled. This could occur if the MSGBUF from IOWAIT was overwritten before being passed to Net'Complete.
BufferAlreadyQd	The buffer passed was already on the list of buffers the system had queued. The buffer is not added to the queue again.
OpenRefused	The attempt to create a connection has been rejected. The arguments to TCP'Open or IP'Open are potentially legal, but request some network resource or activity that cannot be satisfied at this time.
NoConnection	The connection described by UIbuf does not seem to exist, or is not owned by this process. All operations on the connection will be refused.

Other error codes may be added as needed in the future.

6.2 Flow Control Across the Interface

Flow control across the interface between the user process and the TCP and IP protocol software is implemented through the use of message files. The message files act as control channels to transmit flow control and status messages between the user process and the TCP or IP. Each connection makes use of two message files. A general input message file is used to transmit control messages from user processes to the TCP and IP. All user processes share the same general input message file. Each connection also uses a private message file to convey control and status information from the system protocol process to the user process.

The control messages passed between the user process and the system protocol process are short data buffers. These buffers contain the message type along with other information associated with the particular command. The formats for each of the message types is shown in appendix D.

6.3 Interface Control Structures

Each network connection has an associated TCP or IP interface control block. These blocks include a set of pointers and data segment numbers used to keep track of buffers within both the user process and the system protocol process. The

pointers contain buffer addresses relative to the beginning of the stack data segment for each process. A diagram of the TCP and IP interface control blocks is given in appendix E.

The control blocks are maintained in a separate data segment shared by both the user and system protocol processes. The data segment is initialized by the system protocol process when it starts up. The initialization of the data segment divides it into a number of control blocks. Individual control blocks are initialized by the TCP'OPEN and IP'OPEN intrinsics. Responsibility for releasing the control blocks is shared among the TCP'CLOSE, TCP'ABORT, TCP'WAIT and IP'CLOSE intrinsics.

6.4 Windowing, Acknowledgment, and Retransmission

The receive window size and data segment acknowledgment are completely dependent on the number of buffers the user process allocates to a connection. The receive window size of a connection is always set to the amount of free buffer space the user process allocates to the receive side of a connection. Acknowledgments of incoming TCP segments are limited to those sequence numbers which fit in the receive window. Acknowledgments are sent as soon as data is copied from the system protocol buffers to the user protocol buffers.

The initial retransmission algorithm is extremely simple. The first retransmission of unacknowledged data occurs 3 seconds after the original transmission. The second retransmission occurs 6 seconds after the first. The third and successive retransmissions occur 15 seconds after the previous retransmissions. After the third retransmission fails, the user program is informed that an excessive number of retransmissions has occurred.

7 Protocol Software Buffering Scheme

Data buffer management is the most important component of the network protocol software. Data buffers perform the key functions of data storage and data communication within the protocol software. These functions have complex and conflicting requirements which must be balanced by the buffer management scheme. An understanding of the buffer management scheme therefore begins with an understanding of its requirements.

First, data buffers must be considered a scarce resource shared by a number of competing "interests" within the protocol software. These "interests" include the various protocol layers as well as individual network connections within the TCP layer. The major problem is how to effectively allocate buffer resources among these interests. This problem becomes particularly difficult when there is a shortage of buffers.

An examination of the buffer requirements shows that they fall into two categories. The first category includes those buffers used to support general network functions. This includes buffers used in the 1822 and Internet protocol layers. These buffers are assigned to move and store data in these protocol layers without regard to particular network connections. The second category includes those buffers used by the TCP protocol to support specific connections.

The distinction between the two buffer categories is important because buffer use within each category is controlled by a different set of events. The use of buffers assigned to the general network functions can be controlled by the system protocol software. Buffers are processed through the Internet and 1822 protocol layers without regard to the behavior of user processes and their effect on individual connections. Buffers assigned to the connection specific network functions in the TCP and higher level protocol layers are greatly affected by events which occur in user processes. The rate at which data is accepted from or transmitted to the ARPANET by user processes is totally unpredictable. This unpredictability makes it difficult for the system protocol process to effectively assign buffer resources to individual network connections.

Two buffer pools are used to separate those buffering functions shared by all network connections from the connection specific buffering functions. A network buffer pool, maintained by the system protocol process, is used to support the 1822 and Internet and some TCP buffering functions. A user buffer pool, maintained by each user protocol process is used to support connection dependent buffering functions for the TCP and higher level protocols.

7.1 Network Buffer Pool

The network buffer pool supports the following specific functions.

- movement of network packets from the INP driver 1822 and Internet protocol layers;
- storage of Internet datagram fragments in the Internet protocol layer;
- storage of unacknowledged TCP segments which do not fall into the current window;
- movement of network packets from the TCP layer through the Internet and 1822 layer to the INP driver.

The network buffer pool is maintained on the system protocol process stack where it can be accessed easily by the various system protocol layers. All of the buffers in the pool are the same size to minimize the amount of software overhead needed to maintain the buffers. The buffer size is matched to the maximum frame size (128 bytes) which may be transmitted over the X.25 link between the INP and the ARPANET IMP.

The size choice is the result of two constraints. First. the buffers used to catch incoming data must be large enough for the largest incoming network packet. The packets are transferred directly into memory by the INP hardware making it impossible for a packet to cross buffer boundaries. Second, the single size buffer scheme limits the amount of software overhead needed to maintain the buffer pool.

The single size buffer scheme does not waste buffer space because the buffer size is well matched with the data it processes. The 128 byte buffer size allows room for all of the protocol headers and a small amount of data. Messages with more data will use multiple buffers. The buffers are large enough to hold a significant amount of data, yet small enough to limit the waste caused by partially filled buffers.

No attempt is made to assign network buffers to any particular protocol layer or task. Buffers are allocated either when data is read from the ARPANET or when the TCP layer sends data out to the ARPANET.

7.1.1 Packet Compaction

When the total number of network buffers in the free list falls below a set value, a data compaction algorithm is invoked. This algorithm searches for partially filled buffers used to store Internet datagram fragments and unacknowledged TCP segments waiting to be transferred to a user process. These buffers are chosen because processing of the data in them is indefinitely suspended. Compaction of the data in these buffers allows some of the buffers to be released to the free list.

7.1.2 Buffer Recycling

A buffer recycling algorithm is invoked when the system protocol process runs out of free network buffers. The algorithm allows buffers to be reused even if they currently contain data. The mechanism works by identifying which data buffers can be reused without losing irreplaceable information. These buffers are sorted in a priority scheme which allows the least important buffers to be recycled first. The buffer recycling scheme prevents one socket from tying up too much of the network buffer resources. It also helps assure a supply of network buffers even under heavy load conditions.

The buffer algorithm scheme divides network buffers into three categories: free buffers, in-use buffers, and aging buffers. Free buffers are available for immediate use by any protocol layer and are maintained on a common free list. In-use buffers are buffers bound to messages currently being processed and cannot be used for any other purpose. Aging buffers are used in messages where processing is suspended for any number of reasons. These buffers are placed in one of two special lists arranged in order of decreasing age. That is, message buffers which have been blocked for the longest time are at the front of the queue, while the message buffers which were most recently blocked are at the end of the queue.

There are two points in the protocol software where messages may be blocked. The first point is in the Internet Protocol software. Fragmented datagrams cannot be passed on to the TCP and can be blocked indefinitely if one or more of the fragments which make up the datagram is lost. A duplicate datagram may eventually be transmitted leaving the fragmented datagram in a suspended state. The second blocking point is in the TCP software. Unacknowledged segments sent by a foreign host remain suspended in the TCP until they are transferred to a user process buffer. Any segments which are not transferred to a user process will remain blocked indefinitely.

Buffer recycling is implemented through buffer aging lists which are adjuncts to the buffer free list. When an incoming message is blocked, its buffers are attached to the end of one of two aging lists. Buffers bound to datagram fragments are attached to one aging list while buffers bound to TCP segments waiting to be read by user processes are attached to the second aging list.

The aging lists are periodically manipulated when a new datagram fragment comes in or when a user process receives some data from the TCP. Buffers associated with the particular datagram fragments or TCP segments are moved to the end of their respective aging lists. This helps assure that any data which has a chance of being used will not be thrown away.

The buffers bound to fragmented datagrams are recycled first because they are the most expendable. Blocked datagram buffers may be a part of datagrams which have been retransmitted and passed on to the TCP. When the blocked datagram buffers are exhausted the buffers bound to blocked TCP segments are used. These buffers contain the unacknowledged segments which have not been claimed by a user process. The assumptions here are that the user process will never claim these segments and that they are expendable.

7.2 User Process Buffer Pool

The user process is responsible for maintaining a set of fixed length buffers for passing the user data to the TCP. Each buffer must include a four-byte header along with 80 bytes of data space.

The first element of the header is used as either a byte count or a full buffer marker. The count is used by the TCPSEND intrinsic to indicate the number of data bytes in the buffer. The TCPRECEIVE intrinsic uses the buffer full marker to identify buffers which may be reclaimed by the user process.

The second element in the array header contains a list pointer. This pointer is maintained by the intrinsic software and should not be altered by the user process until the buffer is

released.

8 Data Flow through the Protocol Software

Data flow through the protocol software is effected through a series of tests and function calls. The tests check the type and processing state of each packet while the function calls perform specific operations on each packet. These operations include such things as creating or checking headers and queueing or de-queueing packet buffers.

Whenever possible, network packets are processed through all of the system protocol layers without interruption. This helps increase throughput by minimizing two important parameters. First, the amount of buffering required to process data is decreased because all network buffers associated with a packet are released when the packet has passed through the protocol software. Second, the time between the receipt of a packet from the ARPANET and the transmission of an ACK is reduced.

There are a number of instances when the processing of a packet can be interrupted within the system protocol process. This can occur when the lack of some resource or event prevents further processing. Examples of this are as follows:

- Internet datagram fragments waiting for reassembly;
- TCP segments from a foreign host waiting to be read by a user process;
- TCP segments from a user process waiting for window allocation before being transmitted to the ARPANET;

- TCP segments from a user process already sent to the ARPANET but waiting for an acknowledgment.

8.1 ARPANET to the User Level Data Flow

Data packets come in from the network via a DMA interface to the INP network processor. Incoming data is first transferred into the protocol process via network buffers passed to the CSREAD intrinsic which places a read request on the DIT queue of the INP. An arriving network packet is placed in the network buffer by the INP driver. The system protocol process is notified of each I/O completion through the IOWAIT intrinsic.

Processing of network packets begins when an IOWAIT call returns on completion of an CSREAD intrinsic. The first processing step is to link the network buffers which contain the pieces of an 1822 packet.

The next processing steps are performed by the 1822 protocol software. If this is a normal data packet the 1822 header is removed and the data packet is passed as a datagram to the Internet Software. The transfer is done by calling a sequence of Internet protocol routines with the datagram as a parameter.

The Internet software checks the datagram header for integrity and then tries to find the proper address for this datagram. If the datagram is not for the local host it is routed

to the proper ARPANET host and the network buffers are returned to the free list.

If the datagram is a fragment of a larger datagram it is linked to any existing fragments waiting to be processed. If the new fragment does not complete the incoming datagram, the fragment is placed in an aging buffer queue next to the youngest buffer in the partially complete datagram. At this point all processing on the incoming datagram is suspended until the rest of the datagram fragments arrive.

A complete datagram can be sent to one of two places. If the datagram is associated with a raw datagram interface, the datagram will be copied to the user buffers of the processes waiting for it. Datagrams associated with a TCP connection are sent to the TCP protocol software.

The TCP performs a number of functions on incoming segments: first the segment header is checked to see if it belongs to a known socket -- if it does, any acknowledgment information from the header is taken to update the socket status; next, the segment is checked to see if it falls within a window -- if it is not within the window (or a reasonable approximation thereof), the segment is discarded and its buffers are returned to the free list.

Accepted TCP segments are transferred into the user buffers. The transfer is initiated by the user process which provides a buffer through the TCP'OPEN or TCP'RECEIVE intrinsic. A command message sent via the general input message file is used to inform the system protocol process that a buffer is available. The system protocol process transfers as much of its segment as possible to the user buffer. The user process is then notified of the data transfer via the connection's private message file. Only the transferred portions of the segments are acknowledged to the foreign host. Any portions of segments which do not fit in the receive window are stored in the TCP aging queue.

The acknowledgment may be sent in a number of ways. If the same network connection has an outgoing packet waiting for transmission, the acknowledgment information is added to the outgoing packet. If there is no pending outgoing packet, a check is made to see if there is sufficient unacknowledged data to warrant an acknowledgment packet. If there is enough information, a separate acknowledgment packet is generated and transmitted out to the ARPANET as if it were a normal message. If the number of unacknowledged segments is insufficient to justify an acknowledgment packet, the pending acknowledgment bit in the TCB is set and a timer is started. If the timer runs out, an acknowledgment packet is sent regardless of the number of unacknowledged segments.

8.2 User Level to the ARPANET Data Flow

Transfer of data from the user process out to the ARPANET begins with either the TCP'SEND or IP'SEND intrinsic call. The intrinsic software sends a message to the system protocol process to inform it that it has data to send. The system protocol process tests the state of the connection to see if data transmission is feasible. The following are sufficient conditions for data transmission out to the ARPANET:

- enough data has collected to justify transmitting it to the foreign host;
- the user process has specified an EOL in the data transmission;
- there are fewer than eight outstanding 1822 protocol packets waiting for RFNMs to the foreign host;
- the waiting data falls within the foreign host's window.

If the state of the connection does not allow a transmission to occur, a request-to-send data flag is set in the connection control block. When the connection state changes due to some external event, a check is made to see if the new state allows the transmission of waiting data. An example of such an event is the arrival of a RFNM from a foreign host; in this case all of the connections to the foreign host are checked for data waiting for transmission. The connection with data which has been waiting for the longest time is processed first. An attempt is made to combine as many of the waiting TCP segments as possible

into one data transfer to increase the amount of data transmitted.

If there is nothing blocking transmission of the data, the TCP software allocates a buffer, creates the necessary TCP, Internet, and 1822 headers, and copies the data to be transmitted from the user buffer to the system's buffer. The TCP header will include any acknowledgment information for data received on the return socket associated with the connection.

In order to assure proper transmission of the TCP segment a retransmission sequence is started. A retransmission timer is started to wake up the protocol software when a retransmission is needed. If a timeout occurs, the segment is retransmitted as soon as the state of the connection allows it. The necessary conditions for a retransmission are the same as those for the original transmission. If the segment is partially acknowledged, the data left in the retransmission queue is only that data represented by the unacknowledged sequence numbers.

9 FTP Program User Document

The HP3000 FTP program is designed to transfer files between the HP3000 and some other host on the INTERNET. The user starts a file transfer session by invoking the ftp program with the command "rv ftp". Once the FTP program has been invoked, there are two steps involved in transferring files to and from a foreign host. First the user must establish a connection to the foreign host and identify himself with a user name and password. Once logged in, the user can invoke a set of commands described below to transfer files to and from the HP3000. During a session the user can connect to a number of hosts and transfer any number of files. The only limitations are that a user can only connect to one host at a time and can only transfer one file at a time.

It is important to remember that the user's access to files on both the local host (HP3000) and the foreign host is limited by the normal access control mechanisms on both systems. In using file name strings to identify files on either the local host or foreign host the user must use the standard naming conventions appropriate to the host on which the files reside.

9.1 FTP Commands

The following is an alphabetized list of the available FTP commands. All FTP commands consist of three or four characters.

The items enclosed in braces {} are parameter strings.

ACCT {account name} - This command is used specify an account number to the foreign host. The form of the account name will depend on which foreign host is involved. Many foreign hosts do not require account information.

APPE {file name} - This command will cause the FTP to append a local file to an existing file on the foreign host. The foreign host file name is specified as a parameter to the APPE command. The FTP program will prompt the user for the name of the local file.

CON {host name} - This command opens a connection to the foreign host specified by the host name parameter string. Only one connection can be active at any given time

CLOS - This command closes the connection to a foreign host.

DELE {file name} - This commands deletes the foreign host file specified in the file name parameter string.

LIST {directory name} - This command gets a listing of a foreign host directory. The directory name is specified as a parameter string. The directory information is printed out on the user's terminal.

NLST {directory name} - This command gets a listing of a foreign host directory and places it in a local file. The

FTP program will prompt the user for the file name of the local file.

OPEN {host name} - This command performs the same function as the CON command.

PASS {password} - This command specifies a user password to the foreign host. The password is needed to identify the user to the foreign host

QUIT - This command closes all connections to the foreign host and exits the FTP program.

RETR {file name} - This command retrieves a file from the foreign host and stores it on the local host. The command's file name parameter is used to identify the file on the foreign host. The FTP program will prompt the user for the name of the local file used to hold the transferred data. The FTP program will also prompt the user for the local file's record size in bytes and the number of records in the local file.

RNFR {file name} - This command specifies a file on the foreign host which is to be renamed. The file name parameter specifies the old name of the file. The command is used in conjunction with and before the RNTO command which specifies the new file name.

RNTO {file name} - This command specifies the new file name of a renamed file on the foreign host. The file name parameter specifies the new name of the file. The command is used in conjunction with and after the RNFR command which specifies the old file name.

STOR {file name} - This command is used to transfer a file from the local host to a file on the foreign host. The file name parameter is the name of the destination file on the foreign host. The FTP program will prompt the user for the name of the local file.

USER {user name} - This command is used to identify the user to the foreign host.

9.2 Example of an FTP Session

The following is an example of a typical FTP session in which data from a local file named "fum" is sent from an HP3000 to the host BBN-VAX in a file named "foo". The lines which begin with the ">" prompt character are user command lines to the FTP. The other lines are reponses from either the HP3000 User FTP or the BBN-VAX Server FTP. Lines which begin with an integer number are from the BBN-VAX FTP.

```
:rv uftp                    /user command which invokes the FTP
```

```
HP3000 USER FTP VERSION 0       /FTP herald and prompt message
>

> open bbn-vax                  /user command to open a connection
                                /to the host bbn-hp

trying to open a connection to bbn-vax /reassuring messages
connection open                        /from the User FTP

220 BBN-VAX Experimental Server FTP  /herald from bbn-vax

>USER jones                     /specify user login name

331 Enter PASS Command          /bbn vax accepts user name and
                                /asks for pass name

>PASS lslsls                    /pass command with password

230 User Logged in              /BBN-Vax tells user all is well

>STOR foo                       /user store local file into a file
                                /named foo on the foreign host
local file name: foo            /user specifies local file name
                                /after FTP requests it
data connection open            /User FTP indicates start of data
                                /transmission
125 Storing "foo" started okay /BBN-Vax has started receiving data

finished                        /User FTP has finished sending data
```

```
228 File transfer completed okay /BBN-Vax has received the data
close started                    /data connection is closing
close completed                  /data connection is closed
>quit                            /user closes FTP connection and
                                 /exits the FTP program
```

10 TELNET Program User Document

The HP3000 TELNET program allows a user to log into any other host on the INTERNET. The user starts a TELNET session by invoking the command "rv utel". Once the TELNET program has been invoked the user has access to a set of commands described below which allow him to connect to a foreign host and specify the conditions of that connection. A connection to a foreign host is established using either the OPEN or CON commands. Once a connection is established, normally typed characters are automatically sent to the command line interpreter of the foreign host. This allows the user to use the foreign host as if he were attached to it via a local terminal. The only exceptions to this are special command lines intended for the HP3000 User TELNET program. These command lines begin with the "^" character and contain one of the TELNET commands described below. A user can send a line which starts with a "^" character to the foreign host by beginning with two "^" characters.

Because the HP3000 only supports half duplex terminals, all characters typed by the user are echoed by the HP3000 rather than the foreign host. The characters are sent out over the network when the user types a carriage return. This feature prevents the HP3000 TELNET user from using interactive character-at-a-time programs on the foreign host. In addition, data transmissions from the foreign host take precedence over data typed by the

user. Any message from the foreign host will therefore obliterate all of the characters the user has typed since the last carriage return.

10.1 TELNET Commands

The following is an alphabetized list of the available TELNET commands. The items enclosed in braces {} are parameter strings. Any commands typed after a connection has been made to the foreign host must be preceded by a "^" character. The CAPITALIZED characters in each command are required; the lower case characters are optional.

AO - Commands a user program on the foreign host to stop printing. Most hosts including the HP3000 do not implement this command.

AYT - ARE YOU THERE? This is a desperation command sent to the foreign host which has not responded for some time. If the foreign host is an HP3000 it will respond with YES.

BReak - Sends a break command to the foreign host. If the foreign host is an HP3000 the command's effect will be the same as a break typed on a terminal directly connected to an HP3000.

COnnect {host name} {socket no} - Initiates a TELNET connection

to the foreign host specified in the host name string. The socket specification is optional and should not be used unless the foreign host has a TELNET on some non-standard socket.

CLose - Closes the TELNET connection to the foreign host. If the foreign host is dead or not responding, the connection will hang indefinitely. If the connection hangs, the user should use the QUIT command to force the connection to close.

Interrupt - Sends an interrupt command to the foreign host. The interpretation of this command is up to the foreign host. The command is equivalent to a break if the foreign host is an HP3000.

Open {host name} {socket no} - This command is the same as the CONNECT command.

Quit - Aborts a TELNET connection if one exists and exits from the TELNET program.

Sync - Sends a synch message to the foreign host. The interpretation of this message is up to the foreign host. If the foreign host is an HP3000 this message has no meaning and will be ignored.

^Y - Sends a control y to the foreign host. This command only

has meaning if the foreign host is an HP3000. Other foreign hosts will probably ignore it. Remember the control y is sent if the user types "^Y". Simply typing "Y" will send the character "Y" to the foreign host.

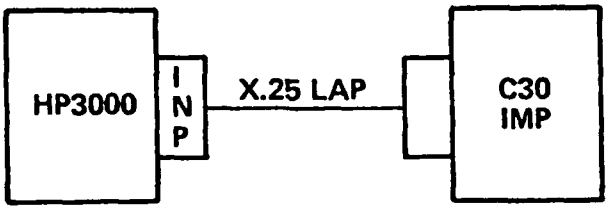
10.2 Example of a TELNET Session

The following is an example of a typical TELNET session in which a user connects to the host BBN-HP.

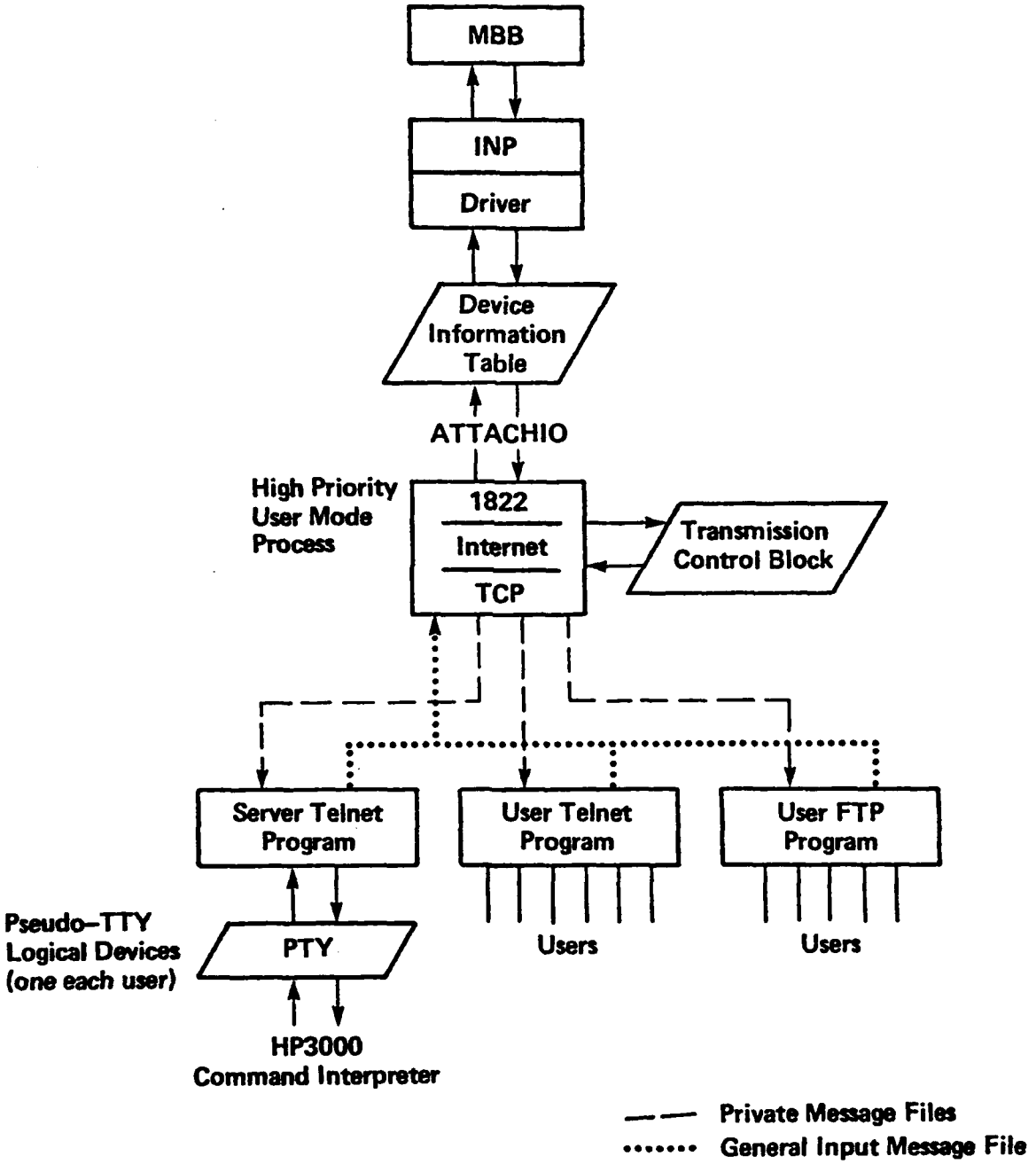
```
:rv utel                    /user invokes the TELNET programn

HP3000 USER TELNET version 0  /TELNET herald and prompt character
>open bbn-hp                 /user opens a connection to bbn-hp

trying to open a connection to bbn-hp /reassuring lines from
connection open                       /the TELNET program

Welcome to BBN-HP server TELNET /herald from BBN-HP
Login: jones                    /prompt for user name. user
                                /responded by typing jones
Password: ldldld                /prompt for password, user
                                /responded with a password

:HP3000 / MPE IV C.00.01  THU ... /herald from HP3000 MPE
                                  /operating system

:                                 /user prompt from MPE
```

```
                                    /operating system. user
                                    /is now free to type
                                    /MPE commands

^close                              /user closes connection

close starting                      /reassuring feedback form
close completed                     /TELNET

>                                   /prompt waiting for next
                                    /TELNET command

>quit                               /exit TELNET

:                                   /prompt from local host
```

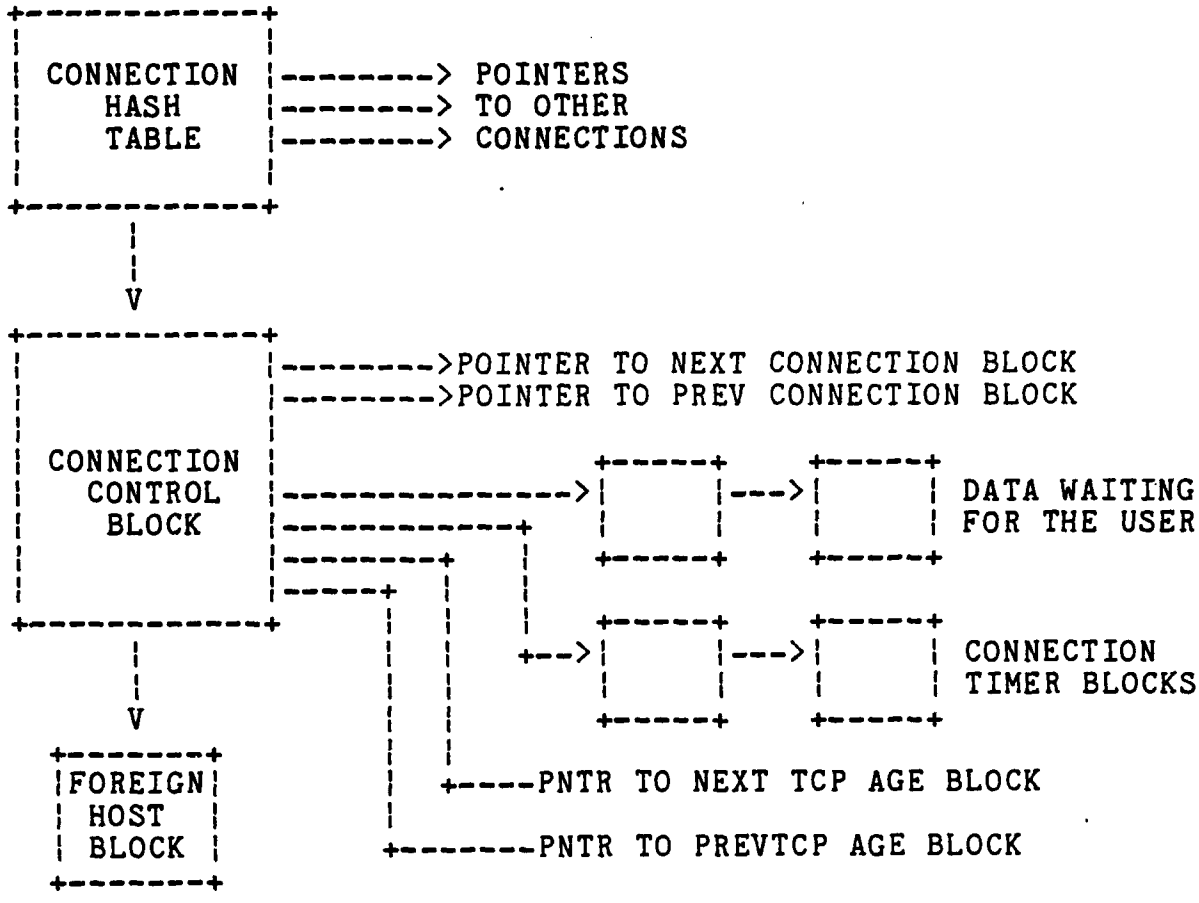
APPENDIX A - HP3000 to ARPANET Link



APPENDIX B - Protocol Software Organization



APPENDIX C - Control Structures

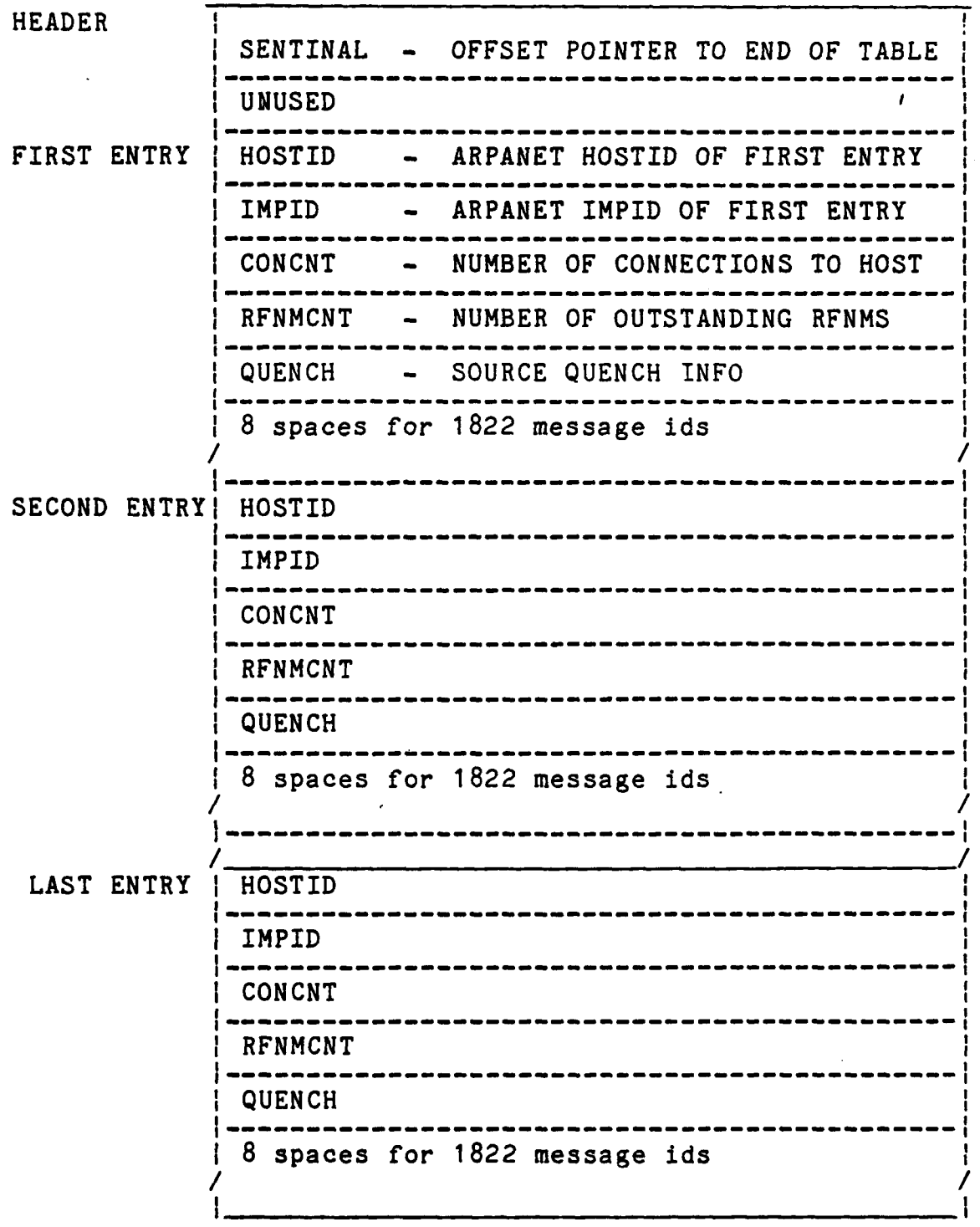


PROTOCOL CONTROL BUFFER AND LIST STRUCTURES

NETWORK RECOURCES CONTROL BLOCK

IPNTR	-	INTERNET AGE LIST POINTER
LIPNTR	-	END OF INTERNET AGE LIST POINTER
TCPNTR	-	TCP AGE LIST POINTER
LTCPNTR	-	END OF TCP AGE LIST POINTER

FOREIGN HOST TABLE

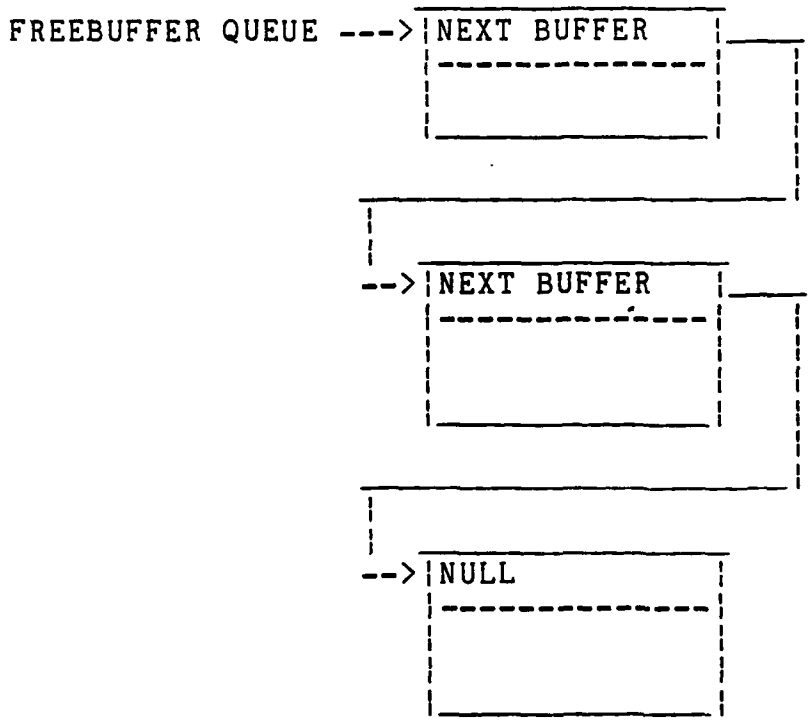


CONNECTION CONTROL BLOCK

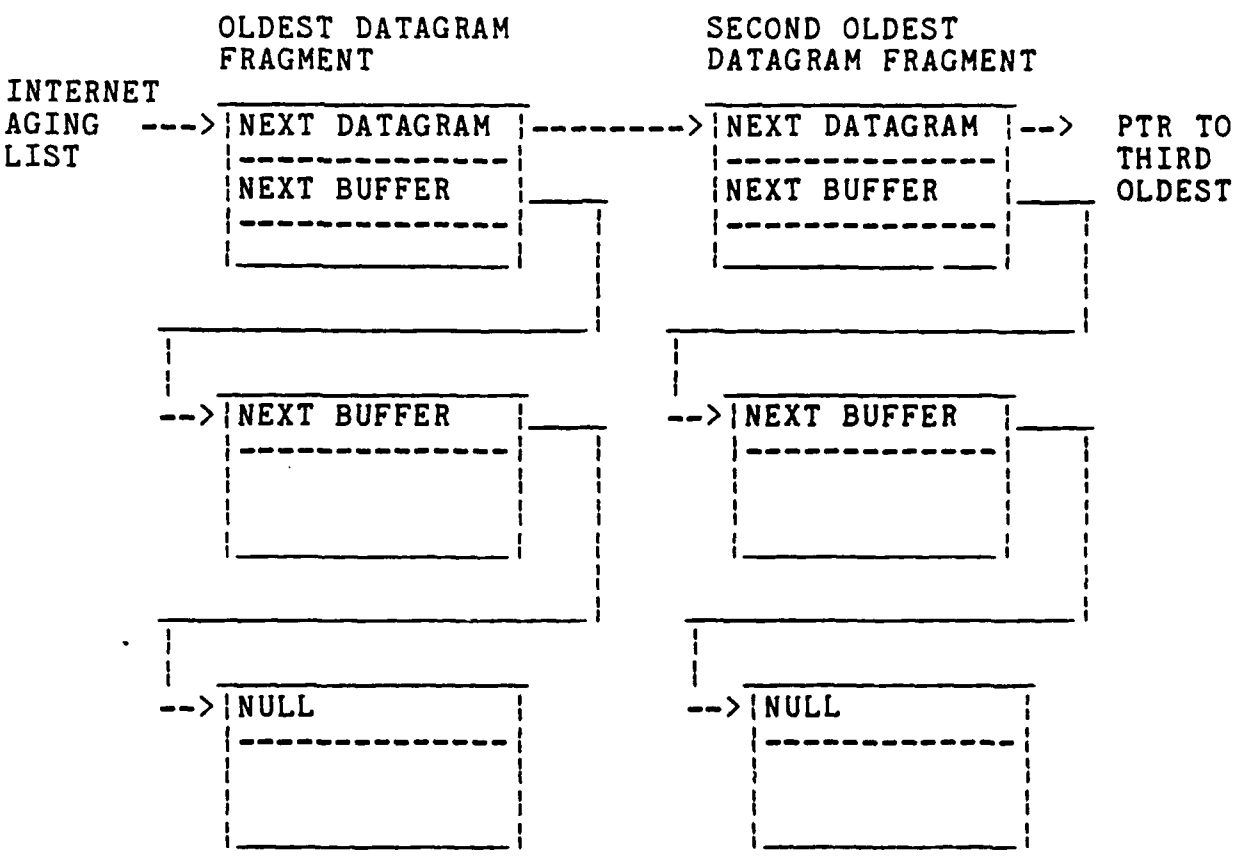
NEXTCON	- POINTER TO NEXT CONTROL BLOCK
PREVCON	- POINTER TO PREVIOUS CONTROL BLOCK
FORPORT	- FOREIGN HOST PORT
LOCPORT	- LOCAL HOST PORT
STATE	- CONNECTION STATE
CONNO	- TCP INTERFACE CONTROL BLOCK NO
MAXMSG	- FOREIGN HOST MAX MESSAGE SIZE
HASHVAL	- HASH FOR INDEX TO CONNECTION TABLE
PROTOMSG	- AREA FOR OUTGOING PROTOCOL BITS
OPENSTATE	- OPEN TYPE (LISTEN OR OPEN)
WAITREASON	- POINTER TO TIMER TABLE BLOCKS
OUTDATA	- POINTER TO DATA WAITING FOR NET
WAITDATA	- POINTER TO DATA WAITING FOR USER
TCP	- POINTER TO NEXT RECYCLE SEGMENT
PTCP	- POINTER TO PREV RECYCLE SEGMENT
FID	- CONNECTION PRIVATE MESSAGE FILE ID
UNUSED	
RTRANTIME	- RETRANSMISSION TIMEOUT VALUE
ARTRANTIME	- AVG RETRANSMISSION TIMEOUT VALUE
FHOSTENT	- POINTER TO FOREIGN HOST TABLE
SRCADDR (32 bits)	- SOURCE (FOREIGN HOST) ADDRESS

CONNECTION CONTROL BLOCK CONTINUED

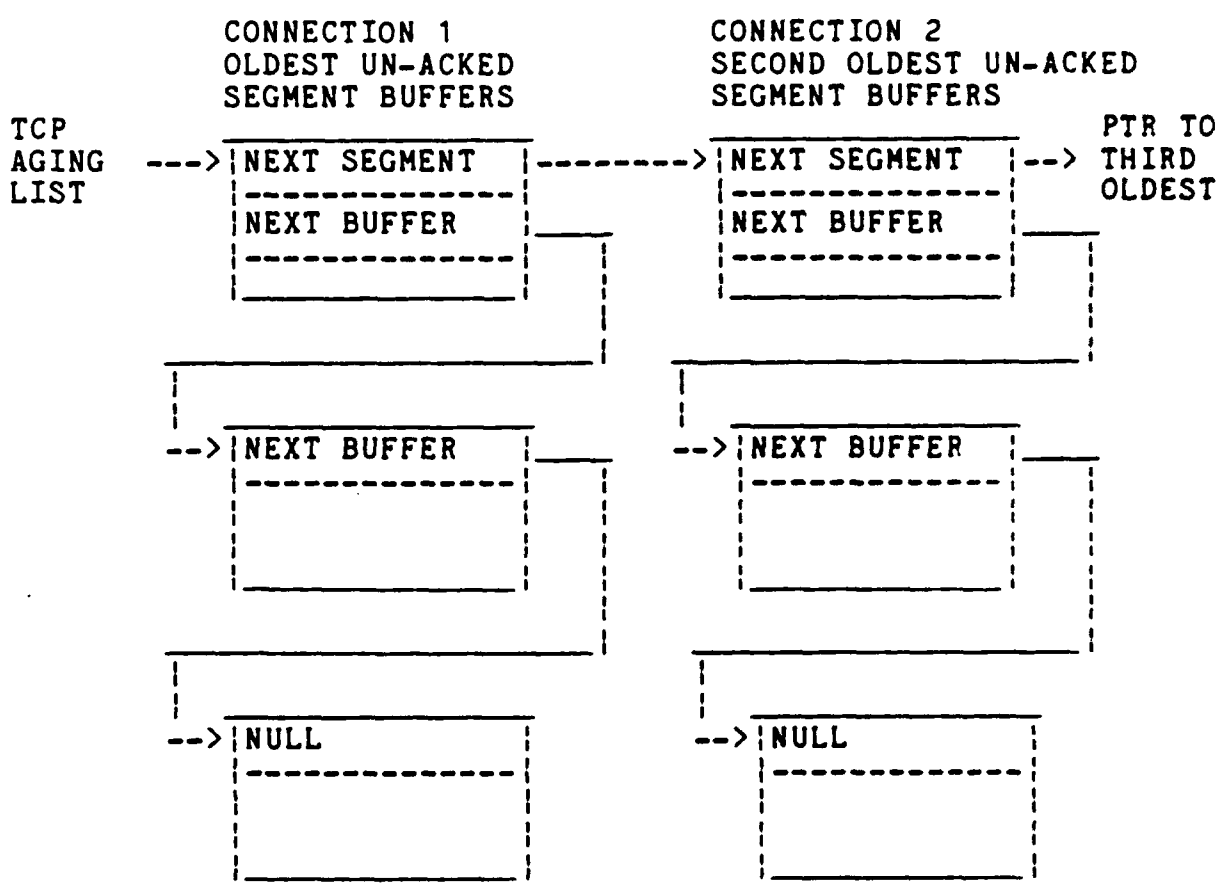
RCVNXT (32 bits)	- NEXT EXPECTED RECEIVE SEQUENCE NUMBER
RCVWND (32 bits	- RECIEVE WINDOW SIZE
RCVUP (32 bits)	- RECEIVE URGENT POINTER
UNUSED (32 bits)	
IRS (32 bits)	- INITIAL RECEIVE SEQUENCE NUMBER
SNDUNA (32 bits)	- FIRST SEQUENCE NUMBER SENT BUT NOT ACKED
SNDNXT (32 bits)	- NEXT SEQUENCE NUMBER TO BE SENT
SNDWND (32 bits)	- WINDOW ADVERTISED BY FOREIGN HOST
SNDUP (32 bits	- SEND URGENT POINTER
SNDWL (32 bits)	- SEND SEQUENCE FOR LAST WINDOW UPDATE
ISS (32 bits)	- INITIAL SEND SEQUENCE NUMBER
FINACK (32 bits)	- SEQUENCE NUMBER OF LAST FIN SENT
RTRANCNT (32 bits)	- NUMBER OF BYTES AWAITING RETRANSMISSION
FINRCV (32 bits)	- SEQUENCE NUMBER OF RECEIVED FIN



NETWORK BUFFER FREE LIST



INTERNET AGING LIST



TCP AGING LIST

TIMER QUEUE ENTRY

WAITTIME	WAIT TIME IN SECONDS
CONADDR	POINTER TO CONNECTION BLOCK
WREASON	REASON FOR WAITING
PRETIME	POINTER TO PREVIOUS TIMER ENTRY
NXTTIME	POINTER TO NEXT TIMER ENTRY
NEXTWAIT	PNTR TO NEXT ENTRY FOR THIS CONECTION
SEQCOUNT	NO. OCTETS IN ACK IF WAIT IS FOR ACK

DATA MESSAGE BUFFER STRUCTURE - HEADER BUFFER

HDH HEADER	HDH - HDH HEADER WORD
1822 HEADER	FORMFLAG - FORMAT FLAG WORD
	MTYPE - LEADER FLAG AND MESSAGE TYPE
	HANDLEHOST - HANDLING TYPE AND SOURCE HOST
	SIMP - SOURCE IMP
	MIDTYPE - MESSAGE ID AND SUBTYPE
	MLENGTH - MESSAGE LENGTH
CONTROL WORDS	IPSTRT - POINTER TO FIRST IP HEADER BYTE
	TCPSTRT - POINTER TO FIRST TCP HEADER BYTE
	DATAOFF - BYTE POINTER TO FIRST TCP DATA BYTE
	URGOFF - OFFSET OF URGENT POINTER IN TCP DATA
	FINFLAG - INDICATES LAST DATA FOR TCP CONNECTION
	SEGLENGTH - NUMBER OF TCP DATA BYTES
	TCPHDRSZ - SIZE OF TCP HEADER IN BYTES
	FRAG - POINTER TO NEXT DATAGRAM FRAGMENT
	PFRAG - POINTER TO PREVIOUS DATAGRAM FRAGMENT
	IP - POINTER TO NEXT DATAGRAM IN AGE LIST
	PIP - PNTR TO PREVIOUS DATAGRAM IN AGE LIST
	NEXTBUF - POINTER TO FIRST DATA BUFFER
	POINTERS FOR 8 MORE DATA BUFFERS
	FIRSTB (32 bits) - FIRST BYTE IN IP FRAGMENT OR FIRST SEQUENCE NUMBER IN TCP SEGMENT
	LASTB - LAST BYTE IN IP FRAGMENT OR

DATA MESSAGE BUFFER STRUCTURE - FIRST DATA BUFFER

HDH HEADER	HDH	- HDH HEADER WORD
IP HEADER	INTERNET	- VERSION, IHL, TYPE OF SERVICE
	TOTALSZ	- TOTAL LENGTH OF IP DATAGRAM
	ID	- IP DATAGRAM ID
	FLAGFRAG	- IP FLAGS AND FRAGMENT OFFSET
	TTLPROT	- TIME TO LIVE AND PROTOCOL
	CSUM	- IP HEADER CHECKSUM
	SRC (32 bits)	- IP SOURCE ADDRESS
	DEST (32 bits)	- IP DESTINATION ADDRESS
TCP HEADER	SPORT	- SOURCE PORT
	DPORT	- DESTINATION PORT
	SEQNO (32 bits)	- FIRST SEQUENCE NUMBER
	TCPACK	- ACKNOWLEDGED SEQUENCE NUMBER
	OFFSETFLAG	- DATA OFFSET AND TCP CONTROL FLAGS
	WINDOW	- ADVERTISED WINDOW AT SENDING HOST
	TCPSUM	- TCP CHECKSUM FOR HEADER AND DATA
	URPTR (32 bits)	- URGENT POINTER
	OPTSEC	- START OF TCP OPTIONS

APPENDIX D - Command Message Formats

General Format of Commands to TCP/IP Process

Offset	Contents
0	Command Type
1	Interface Control Block connection number
2-7	Arguments, if any

TCP/IP Message File Commands

Code	Meaning
0	Open TCP Connection (see below)
1	Close TCP Connection
2	Abort TCP Connection
3	Send TCP Data (see below)
4	Receive TCP Data (see below)
5	Close TCP/IP process logfile
6	Shutdown Network (see below)
7	Open IP Connection (see below)
8	Close IP Connection
9	Send IP Datagram
10	Receive IP Datagram

Arguments to Open TCP Connection Command

Offset	Meaning
2-3	Foreign Host InterNet Address (or zero)
4	Foreign Port Number
5	Local Port Number
6	Active/passive Listen Flag
7	Initial Window Size

Arguments to Send TCP Data Command

Offset	Meaning
2	Number of bytes to send

Arguments to Receive TCP Data Command

Offset	Meaning
2	# Bytes of receive buffer space added

Arguments to Network Shutdown Command

Offset	Meaning
2	Password value (numeric)

Arguments to Open IP Connection

Offset	Meaning
2-3	Foreign Host Internet Address (or zero)
4	Internet Protocol Number

All messages from the TCP/IP process to the user intrinsics consist of a single word which is the command number.

Command Numbers

Code	Meaning
100	TCP Connection has been opened
101	Connection Close request acknowledged
102	Connection Aborted
103	Data Sent
104	Data Received
105	TCP Receive Side Closed
106	Open Connection Request Refused
107	Foreign Host Refused Connection
108	Notification of Excessive Retransmissions

APPENDIX E - User Program Data Structures

User's Connection Control Block

Offset	Meaning
0	Connection Number
1	Interface Control Block Segment Number
2	RIN Number for Locking ICB tables
3	Verification Word
4	File number of Message File to TCP/IP
5	File number of Message File from TCP/IP

Overhead Words of Each User Buffer

Offset	Meaning
0	Pointer to next buffer (or zero)
1	bit 0: Buffer In Use Flag bit 1: End-of-Letter Flag bit 2-15: Buffer length in bytes
2	(if TCP) Urgent-Data Pointer (if IP) Received Datagram Size in bytes

TCP'Status Buffer Format

Offset	Meaning
0	Local Port Number
1	Foreign Port Number
2-3	Foreign Host's Internet Address
4-5	Receive Window Size in bytes
6-7	Transmit Window Size in bytes

Overhead Information

Offset	Description
0	Number of currently open connections
1	Flag to inhibit opening new connections
2	Recovering from net crash flag
3	ICB segment verification word
4	DST number of TCP/IP process stack
5	DB offset of TCP stack verification word
6	DB offset of TCP/IP connection table

A compile-time parameter NCON determines the number of connections that may be open concurrently. Beginning at offset 7 of the Interface Control Block, there are NCON words used as Connection-In-Use flags. The value of these flags is zero if the connection is not in use. and -1 if it is in use.

Following the array of Connection-In-Use flags, there are NCON connection descriptors. Each connection descriptor has the following format:

Connection Descriptor

Offset	Meaning
0	Pointer to head of oldest unreturned transmit buffer
1	Pointer to head of buffer containing data next to be sent
2	Pointer to next byte to be sent *
3	Number of bytes in current buffer remaining to be sent. *
4-5	Foreign Host Internet Address of next datagram to be sent **
4	Pointer to head of buffer containing data next to be acknowledged *
5	Pointer to next byte to acknowledge *
6	Number of bytes in current buffer not yet acknowledged *
7	Transmit side state
8	Receive side state
9	DB pointer to user control block
10	DST number of user's stack
11	bit 0: 0 if TCP; 1 if IP connection bits 1-15: User process number
12	Pointer to head of buffer where next data received will be written
13	Pointer to next byte of buffer space *
14	Number of free bytes in current buffer *
15	Pointer to head of oldest unreturned receive buffer
16	Pointer to first byte of data not yet returned to user program *

* variable used only by TCP connections
** variable used only by IP connections