f.G. (12)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ADA024820

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Note No. 75	2. GOVT ACCESSION NO. (14) TN-75	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Single-Connection TCP Specification (Preliminary Documentation).		5. TYPE OF REPORT & PERIOD COVERED (9) Technical Note
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) James E. Mathis (11) 25 Jan 76		8. CONTRACT OR GRANT NUMBER(s) (15) MDA903-76C-0093, ARPA Order 2494
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford Electronics Laboratories Stanford University Stanford, CA 94305 (12) 39p.		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 6T10
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Information Processing Techniques Office 1400 Wilson Ave., Arlington, VA 22209		12. REPORT DATE January 25, 1976 13. NO. OF PAGES 31
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office) Mr. Philip Surra, Resident Representative Office of Naval Research Durand 165, Stanford University		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this report)

Reproduction in whole or in part is permitted for any purpose of the U. S. Government

"A"

DISTRIBUTION STATEMENT A
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)

Interconnection, communication protocols, packet radio network, ARPANET

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

An implementation scheme is described for a single connection, user internetwork Transmission Control Program using the Cerf-Kahn protocol. It is designed for a dedicated micro-processor and supports low delay, low through-put interactive traffic. In this preliminary documentation a detailed implementation specification is presented in an ALGOL-like notation, along with a brief discussion of its functions, user interface, control variables and data structures.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Single-Connection TCP Specification
(Preliminary Documentation)

J. E. Mathis

Digital Systems Laboratory
Stanford University
Stanford, California 94305

January 25, 1976

Technical Note #75

DIGITAL SYSTEMS LABORATORY
Dept. of Electrical Engineering Dept. of Computer Science
Stanford University
Stanford, California

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2494, Contract No. MDA903-76C-0093.

Table of Contents

Section Page

Subsection

1 INTRODUCTION

This document describes a preliminary implementation scheme for a single-connection internetwork Transmission Control Program [TCP]. It is designed to operate on a dedicated small computer with minimal operating system assistance and to support low delay, low through-put interactive traffic. Although assumed to be attached to the ARPANET via a local or remote host Interface [1], only minimal changes are necessary for connection to the Packet Radio Network using the Channel Access Protocol [2].

Detailed knowledge is assumed of the internetwork TCP protocol and the reader is referred to the official specification [3] [4] for justification and further discussion of the details of the protocol.

NOTE: There have been several changes to the protocol that are not listed in the December 1974 revision [3]. Among these, addition of the Beginning Of Segment [BOS] bit and Timestamping in effect [T] bit in the control information word; addition of a 32 bit field for timestamping information and sending an ACKnowledgement for FIN requests. The final specification will be available by February, 1976.

2 DESCRIPTION of TCP FUNCTIONS

For maximum size reduction, only the single-connection user subset of functions are implemented. Unsupported functions are:

1) Unspecified sockets - On doing an OPEN, the user must fully specify the destination NET, TCP and PORT socket addresses, eliminating connections analogous to the NCP "listening" connection needed only by server and logger processes. (To permit a host to provide service serially through a single channel TCP, this function may be implemented leter.)

2) Re-assembly of fragmented segments - Packets must have both the Beginning Of Segment [BOS] and End Of Segment [EOS] control bits asserted. If not, on passing through a gateway, the segment was broken into fragments; it should be discarded without processing. The re-construction of segments has yet to be resolved; but it can be avoided by limiting segment length (by receive window size control) to less than the gateway fragmentation threshold. Though in the future, it may become necessary to implement it.

3) ECHO and TRASH special functions - Being solely for experimentation purposes, ECHO and TRASH special

functions are not implemented. Messages containing these special functions are ignored. (If the cost penalty, mainly storage capacity, is not too high, these functions will be implemented later.)

4) Timestamping - The timestamping control bit is ignored and the contents of the returning packet's timestamp field will be undefined.

5) Parameter change/Status socket - Being strictly single-connection, there is no Parameter change/Status socket. (Note - this socket is distinct from the Well Known Socket 0.)

Except for these restictions, the TCP insures end-to-end acknowledgement, error correction, duplicate detection, sequencing and flow control; providing the user process a reliable, error-free logical communications channel.

3 USER-TCP INTERFACE

Five primitives comprise the USER/TCP interface. It is intended that these routines be called via a subroutine jump or supervisor call and indicate command acceptance or rejection on exit. When complete, the user will be notified of the final disposition of OPEN, SEND and CLOSE requests. This allows user processing to proceed asynchronously and in parallel with TCP processing.

1) OPEN CONNECTION - Used to establish a connection, the OPEN primitive is passed the address of the foreign and local socket ids. If a connection already exists, the OPEN is rejected and an error returned to the caller. After checking the request and socket ids, the control variables are initialized and an Initial Sequence Number (ISN) is chosen. Since we can remember the last sequence number used on the previous connection, it is not necessary to choose a clock-based ISN, but rather just continue. (Cf. [3] section 4.3.1 for more on ISN selection.) After notifying the Net Output Process to send a SYN, control is returned to the caller. When the connection is ready for use, the user process is notified.

2) SEND LETTER - This call causes the data contained in the user buffer to be sent on the connection. The buffer address and length are stored in a common data area, the output process notified of work pending and control returned to the caller. After the

data has been packetized and sent (but not ACKnowledged), the user process is notified. Letter boundaries are ignored, every data packet has the End Of Letter [EOL] bit set. If the connection is not established, the data is queued for sending later. There can be at most one outstanding send. The mechanics of sending data is covered in section xxx.

3) RECEIVE LETTER - The user process is notified when data arrives on the connection and it is moved into user buffers by a RECEIVE. RECEIVE is called with a buffer pointer and maximum byte count and returns the actual byte count. Again, letter boundaries are ignored. After delivery to the user, an ACKnowledgement is sent to the sender. The exact details of moving data into the user buffer is covered later in section xxx.

4) INTERRUPT - A special control signal is sent to the destination indicating an interrupt condition. All unsent or unacknowledged data will be flushed. If the connection is not established, an error is returned.

5) CLOSE CONNECTION - This command causes the connection to be closed. If it is not open, an error is returned. Pending unsent or received data is flushed, no more accepted and a FIN sent to the remote TCP. Control is returned to the caller and the user is notified when the close is finalized. The exact process of closing a connection is covered later in section xxx.

4 TCP STRUCTURE

4.1 OPERATING ENVIRONMENT

The TCP is designed to operate under a very simple operating system structure. Each process has a process control table containing space for its run-time stack, status save area and an external event scoreboard. To signal a process of some event, the signalling process sets a bit in the called process' scoreboard. Each process is responsible for periodically polling its scoreboard and acting appropriately. After processing the signal, the process then clears the flag bit. Each process runs to completion and context switching happens only when a process explictly releases control. The only operating system primitive is one that causes the context to switch to the next active process. All processes run at the same priority level.

Using a scoreboard has three important restrictions. It is not possible to maintain temporal ordering of signals, multiple signals of the same type are condensed into one, and it is not possible to transfer any data along with the signal. The first two restrictions are not critical to the TCP implementation; indeed, a TCP implementation running under a normal message queueing operating system must go to some effort to remove extraneous signals resulting from process asynchrony. The third restriction requires putting data associated with an event in a "global" locaion known to both processes.

There must also be I/O devices and their associated device driver routines. It is assumed that the devices are interrupt driven, though programmed device polling is possible at reduced data rates. The following devices are needed:

1) Net input device -

2) Net output device -

3) Hardware timer -

4.2 NETWORK INTERFACE

To allow the TCP to be used with computer networks of different structure and interfacing requirements, all network dependent code is concentrated in three routines. While designed for the ARPANET and Packet Radio Network this partitioning should be adequate for most other network configurations. (The most obvious exception is the Very Distant Hosts in the ARPANET; which require an additional watchdog process to provide control functions for the IMP-HOST line protocol used (Cf. [1] appendix F).) The routines and their functions are:

INITIALIZENETWORK - Called on system initialization, this routine initializes the device driver routines and performs the HOST-NETWORK start-up sequence. It returns when the network is ready to deliver/accept messages to/from the host computer.

NETINPUT - Passed the address of a packet buffer, this routine initiates action to accept a message from the network. It performs the network-dependent processing of the NETWORK-HOST message header, e.g. in the ARPANET, it would verify that the message is of type 0 (regular packet) and not a special IMP-HOST message. Control returns to the caller when a valid message is received.

NETOUTPUT - Called with the address of a message to send, this routine performs the network-dependent formatting of the HOST-NETWORK message header. Transmission of the message into the network is started and control returned to the caller when the output is completed.

4.3 PROCESS INTERACTION

The TCP is composed of two processes, appropriate device drivers and a set of user-callable routines sharing a common data base, the Transmission Control Block. Each process in non-interruptable, running to completion, and communicates via signal flags. The TCP INPUT process handles incoming messages and either notifies the user process of new data received or signals the TCP OUTPUT process to send error packet or various control packets. The input process is the only process that receives data from the net and likewise, the output process is the only sender of data to the network. The TCP OUTPUT process, on command, sends error packets or control packet on request of the input process or data on request of the user process. It also is responsible for retransmitting unACKnowledged data periodically. The Network device driver also communicates via signals, notifying the caller of "done." The hardware timer interrupts the computer periodically and its device driver signals the output process for packet retransmission.

4.4 CONNECTION CONTROL VARIABLES

All of the information local to a specific connection is kept in a Transmission Control Block [TCB]. The following are the fields of the TCB and their length.

DHOST - (16 bits) The local PSN address of the destination host or gateway. For simplicity, it is assumed that the local host address is the same as the TCP address if in the local network. Otherwise, the destination NET id is used to determine the local gateway address.

DNET - (8 bits) The destination network Id. (Cf. [3] section 4.2.1 for list of assigned network ids.)

DTCP - (16 bits) The destination TCP id.

DPORT - (24 bits) The destination PORT id. Along with DNET and DTCP, they form the destination socket.

SNET - (8 bits) The network id of the local network.

STCP - (16 bits) The TCP id of the local TCP.

SPORT - (24 bits) The local PORT id. Along with SNET and STCP, they form the local socket number.

CONNECTIONSTATE - (8 bits) The actions performed by the TCP depend upon what has happened previously. There are six "connection states" in a user TCP. They are:

1) CLOSED - The connection, as such, does not exist.

2) SYNSENT - The user process has done an OPEN and a SYN has been sent to the foreign TCP in an attempt to establish a connection. We wait for the ACKnowledgement of our SYN before going to the ESTABLISHED state and notifying the user process that the connection is usable.

3) SIMULINIT - After sending a SYN to establish a connection, we received a SYN without an ACKnowledgement of our SYN from the foreign TCP. This represents an attempt by both ends to open the connection simultaneously. We send an ACKnowledgement of the SYN we received and initialize the connection dependent variables. We wait for the ACKnowledgement of our SYN before going to the ESTABLISHED state and notifying the user process that the connection is usable.

4) ESTABLISHED - The three-way handshake to synchronize the connection was successful and the connection is usable for data transfers.

5) FINWAIT - The user process has done a CLOSE and we have sent out the FIN. We wait for the FIN to timeout or to receive a FIN and ACKnowledgement of our FIN before going to the CLOSED state.

6) FINRECEIVED - We have received a FIN from the foreign TCP. The user is notified of the remote close and we send a FIN and ACKnowledge the receive FIN. We now wait for an ACKnowledgement of our FIN or its timeout before going to the CLOSED state.

RCVSEQ - (32 bits) The next sequence number expected

RCVWS - (16 bits) The receive window size.

INITSEQ - (32 bits) The initial receive sequence number used by the foreign TCP. This is used to detect old duplicates of the SYN that established the connection.

SNDSEQ - (32 bits) The next sequence number to send.

SNDWS - (16 bits) The send window size.

LASTWSEQ - (32 bits) The last sequence number used to update send window.

LSWEDGE - (32 bits) The left send window edge sequence number.

INPUTHEAD - (16 bits) The pointer to the head of the receive data reassembly ring buffer.

BUFFERPOINTER - (16 bits) The address of the start of the user's send buffer.

BUFBYTECOUNT - (16 bits) The number of bytes in the user's send buffer. The byte count and buffer pointer are set when the user does a send and are updated as the output process removes and sends data.

RTXWAKEUP - (16 bits) Count of the number of retransmissions sent without receiving any new ACKs. It is cleared when a valid ACK comes in. When the number of retransmissions exceeds a preset value, the user is notified of "TCP not responding."

RTXPOINTER - (16 bits) The pointer to the head of the retransmission ring buffer.

RTXCOUNT - (16 bits) The number of bytes of data in the retransmission buffer.

RTXCONTROL - (16 bits) It contains the control field of the control packet queued up to be retransmitted. If zero, then no control packet queued up. Only one control packet can be queued for retransmission.

RTXCNTRLSEQ - (32 bits) The send sequence number of the control packet queued up to be retransmitted.

RTXDATASEQ - (32 bits) The send sequence number of the data byte at the head of the retransmission ring buffer.

In addition, there are several assembly-time constants that set the size of various buffers.

MAXPACKETSIZE - The maximum number of data bytes that can be put in the text field of a internet packet. MAXPACKETSIZE, internetwork header length and length of local PSN control fields determine the size of the send packet buffer.

MAXRCVWS - The maximum receive window size is set by the size of the reassembly ring buffer.

MAXRTXCOUNT - The maximum amount of data queued up to be retransmitted. This determines the size of the retransmission ring buffer.

There are also several buffers associated with the connection. They are:

RCVPKTBUFFER - The buffer that incoming packets are written into by the network device driver.

REASSMBUFFER - The ring buffer where input data is reassembled and stored pending deliver to the user process.

REASSMFLAGS - A boolean vector that indicates which elements of the reassembly buffer contain a data octet.

SENDPKTBUFFER - The buffer that outgoing packets are constructed in and sent out by the network device driver.

RTXBUFFER - The buffer where data waiting to be ACKnowledged is enqueued.

4.5 INPUT PACKET HANDLER

After initializing the local network interface, the TCP INPUT process is awakened when a packet arrives from the network. The packet is checked for an internetwork message; malformed packets are simply discarded. The validity check involves verifying that the message is long enough to contain the TCP packet header and the packet header version number is correct. The BOS and EOS control bits must both be asserted; the current implementation can not handle fragmented segments. The checksum is finally calculated and detectably damaged packets are discarded; they will be re-transmitted by the sender.

After validation, the message is checked for special function or error information and processed appropriately. In this preliminary specification, their handling is not detailed; but the TCP must be sensitive to RESET ALL, RESET and QUERY special functions and all error conditions.

Packets without control dispatch refer to specific connections; the foreign and local sockets are checked against those of the single connection we service. If different, an error message (connection non-existent) is constructed and queued to be sent by the Net Output Process.

If in the SYNSENT state and we receive a SYN with INT, DSN or FIN then the SYN is malformed and an error is returned. If the packet acknowledges the SYN we sent, the connection is synchronized. We ACKnowledge the received SYN, initialize the Transmission Control Block and notify the user of connection establishment. If instead of an ACK, the packet contains only a SYN, then we have a simultaneous attempt by both sides to open the connection. (Cf. [3] section 4.3.2 for details of SYN collision.) The new connection state is SIMULINIT.

In the SIMULINIT state, if we receive a SYN it is first checked to see if it is a duplicate of the SYN that caused the state change to SIMULINIT. If so, then just ACKnowledge receipt. Otherwise we have two different SYNs and can not tell which is valid, so we send back an error and reinitialize. If instead of a SYN, we get an ACKnowledge of our SYN, the connection has been established by a four-way handshake. Notify the user and process any data that may accompany the ACK.

Once the connection is established, errors are sent for all SYNs received, except for duplicates of the original.

5 USER INTERFACE LOGIC DOCUMENTATION

```
OPEN (OPENBLOCKPOINTER):
        if CONNECTIONSTATE notequal CLOSED  then
            return (connection already open error)

        (move socket addresses into control block)
        (convert DNET,DTCP address into local PSN host/gateway address)
        RCVWS:=MAXRCVWS
        INITCONNECTION
        comment- return to caller. notify user process when connection
            becomes established or on error condition.
        return (ok)


SEND (BUFFERADDRESS, BUFFERLENGTH):
        comment- put buffer pointer and length into TCB for send process

        if BUFBYTECOUNT notequal 0  then
            return (too many SENDs error)
        else
            begin
            BUFFERPOINTER:=BUFFERADDRESS
            BUFBYTECOUNT:=BUFFERLENGTH
            (notify TCP OUTPUT PROCESS to send data)
            end
        return (ok)


INTERRUPT
        if CONNECTIONSTATE notequal ESTABLISHED  then
            return (connection not open error)

        (notify TCP OUTPUT PROCESS to flush send data)
        (notify TCP OUTPUT PROCESS to send INT)
        return (ok)


RECEIVE (BUFFERADDRESS, BUFFERLENGTH, result BYTECOUNT):
        if CONNECTIONSTATE notequal ESTABLISHED  then
            return (connection not opened error)

        PTR:=BUFFERADDRESS
        BYTECOUNT:=0

        while (BYTECOUNT < BUFFERLENGTH) and REASSMFLAGS(INPUTHEAD)  do
            begin
            user buffer (PTR) := REASSMBUFFER (INPUTHEAD)
```

```
            REASSMFLAGS (INPUTHEAD) := FALSE
            PTR:=PTR+1
            INPUTHEAD := (INPUTHEAD+1) MOD MAXRCVWS
            BYTECOUNT:=BYTECOUNT+1
            end

        RCVSEQ := RCVSEQ + BYTECOUNT

        (notify TCP OUTPUT PROCESS to send ACK)

        if REASSMFLAGS (INPUTHEAD)  then
            (notify user process of data remaining to be received)

        return (ok)


CLOSE:
        case CONNECTIONSTATE  of
            =SYNSENT:
            DELETECONNECTION

            =SIMULINIT:
            =ESTABLISHED:
            begin
            CONNECTIONSTATE:=FINWAIT
            (notify TCP OUTPUT PROCESS to flush send data)
            (notify TCP OUTPUT PROCESS to send FIN)
            end

            =CLOSED:
            =FINWAIT:
            =FINRECEIVED:

        return (ok)
```

6 TCP INPUT PROCESS LOGIC DOCUMENTATION

```
TCP INPUT PROCESS:
        (wait for network interface initialization)

  LOOP: NETINPUT (RCVPKTBUFFER)

        if (packet length greaterthan or equal minimum permitted) and
             (packet header version number equal 0)  then
            begin
            comment- packet verified as a TCP message.
            if (packet BOS bit =1 and packet EOS bit =1)  then
                begin
                comment- unfragmented message, process.
                if CHECKSUM (RCVPKTBUFFER) = 0  then
                    comment- checksum ok, packet not damaged.
                    if (packet Control Dispatch bits equal 0)  then
                        HANDLEREGULARPACKET
                    else
                        HANDLESPECIALPACKET
                end
            else
                comment- fragmented message.  code to do fragment
                    reassembly goes in here. but for now just...
                (log error)
            end
        else
            comment- garbage packet
            (log error)

        goto LOOP


CHECKSUM (PACKETPOINTER):
        comment- computes the 16 bit 1's com[illegible]ment sum of the header
            and text fields of the packet.  [illegible] the sum is 0, then the
            packet is not (hopefully!) damaged.


HANDLEREGULARPACKET:
        if ADDRESSCHECK  then
            begin
            comment- packet is for this connection. process according to
                connection state.

            case CONNECTIONSTATE of
                =SYNSENT:
                if (packet SYN bit =1)  then
                    begin
                    if (packet FIN, INT or DSN bits =1)  then
```

```
                comment- should not have these control bits set,
                    return unacceptable SYN error.
                XMITERROR (EFP+USYN)
            else
                if (packet ACK bit =1)  then
                    begin
                    if ACCEPTABLEACK  then
                        begin
                        SETTCB
                        (notify TCP OUTPUT PROCESS to send ACK)
                        CONNECTIONSTATE:=ESTABLISHED
                        (notify user of connection established)
                        HANDLEACK
                        end
                    else
                        XMITERROR (EFP+USYN)
                    end
                else
                    comment- simultaneous attempts to open the
                        connection.
                    begin
                    CONNECTIONSTATE:=SIMULINIT
                    SETTCB
                    (notify TCP OUTPUT PROCESS to send ACK)
                    end
            end

        =SIMULINIT:
        if (packet SYN bit =1)  then
            begin
            if (packet seq number equal INITSEQ)  then
                comment- duplicate of first SYN, don't send an
                    error, but force an ACKnowledgement.
                (notify TCP OUTPUT PROCESS to send ACK)
            else
                begin
                comment- we have received two different SYNs and
                    can't tell which to believe. so send error
                    message and reinitialize connection and try
                    again.
                XMITERROR (EFP+USYN)
                INITCONNECTION
                end
            end
        else
            if (packet ACK bit =1)  then
                if ACCEPTABLEACK and INRCVWINDOW  then
                    begin
                    comment- acknowledged our SYN, so connection
                        now synchronized.
                    CONNECTIONSTATE:=ESTABLISHED
                    (notify user of connection established)
```

```
                NORMALCASE
                end

        =ESTABLISHED:
        if (packet SYN bit =1)  then
            begin
            if (packet seq number equal INITSEQ)  then
                begin
                comment- duplicate of the original SYN that
                    established connection. force an ACK and
                    process any data.
                (notify TCP OUTPUT PROCESS to send ACK)
                if INRCVWINDOW  then
                    NORMALCASE
                end
            else
                comment- unacceptable SYN.
                XMITERROR (EFP+USYN)
            end
        else
            begin
            if INRCVWINDOW  then
                NORMALCASE
            else
                (notify TCP OUTPUT PROCESS to send ACK)
            end

        =INWAIT:
        if INRCVWINDOW and (packet FIN bit =1)  then
            begin
            comment- we have sent a FIN and now have received a
                FIN. ACKnowledge FIN and see if can delete the
                connection.
            RCVSEQ:= (packet seq number) + (packet text length)
            RCVSEQ:=RCVSEQ + CONTROLLENGTH (RCVPKTBUFFER)
            CONNECTIONSTATE:=FINRECEIVED
            (notify TCP OUTPUT PROCESS to send ACK)
            if (packet ACK bit = 1) and ACCEPTABLEACK  then
                HANDLEACK
            end

        =FINRECEIVED:
        if INRCVWINDOW and (packet ACK bit =1)  then
            if ACCEPTABLEACK  then
                HANDLEACK

    end
else
    XMITERROR (EFP+NONX)

return
```

```
INRCVWINDOW:
        comment- determines if any part of the packet that just came in
            lies inside the receive window.


ACCEPTABLEACK:
        comment- return TRUE if packet ACKs something we sent that has
            not yet been ACKed, i.e.  LSWEDGE <= ACKfield <= SNDSEQ


NORMALCASE:
        comment- this processes the normal case of putting new data into
            the right place in the circular reassembly buffer.  also
            processes other possible things in packet.

        if (packet ACK bit =1) and ACCEPTABLEACK  then
            HANDLEACK
        if (packet INT bit = 1)  then
            HANDLEINT
        if (packet text length greater than 0)  then
            HANDLEDATA
        if (packet DSN bit = 1)  then
            HANDLEDSN
        if (packet FIN bit = 1)  then
            HANDLEFIN

        return


HANDLEACK:
        comment- correlates the ACK that came in (and window, etc.) with
            what we have already put in the control block.  it is where
            confirming ACKs will remove data from the Retransmission
            ring buffer.

        if PRECEDE (LASTWSEQ, (packet sequence number) )  then
            begin
            comment- update the send window size if this is the latest
                packet we have seen.

            SNDWS := (packet window size field)
            LASTWSEQ := (packet sequence number)
            end

        comment- convert next sequence number expected to sequence
            number of last octet ACKnowledged.

        TMPSEQ:=(packet ACK field) - 1
```

```
        LSWEDGE := (packet ACK field)

        if RTXCONTROL notequal 0  then
            comment- see if control ACKed.

            if PRECEDE (RTXCNTRLSEQ, TMPSEQ)  then
                begin
                comment- if our FIN was ACKed, delete the connection.
                if RTXCONTROL = FIN packet  then
                    DELETECONNECTION
                RTXCONTROL:=0
                end

        if RTXCOUNT notequal 0  then
            comment- see if any data is ACKed and if so, remove them

            if PRECEDE (RTXDATASEQ, TMPSEQ)  then
                begin
                COUNT:=TMPSEQ - RTXDATASEQ + 1
                RTXCOUNT:=RTXCOUNT - COUNT
                RTXPOINTER:= (RTXPOINTER + COUNT) MOD MAXRTXCOUNT
                RTXDATASEQ:=RTXDATASEQ + COUNT
                end

        return



SETTCB:
        comment- fills received information into control block from
            arriving SYN packet
            RCVSEQ is the next sequence number expected,
            SNDWS is the send window size,
            INITSEQ is the initial receive sequence number used,
            LASTWSEQ is the last sequence number used to update send
                window,
            LSWEDGE is the left send window edge.

        SNDWS:= (packet window size)
        INITSEQ:= (packet seq number)
        LASTWSEQ:= (packet seq number)
        RCVSEQ:= (packet seq number) + 1

        return



HANDLEDATA:
        comment- this routine moves data from the input packet into the
            circular reassembly buffer.
```

```
        INPUTHEAD is a pointer to the head of the reassembly buffer,
        RCVSEQ is the left receive window edge sequence number,
        PINDEX is the index into the text field of the input packet
        REASSMBUFFER is the actual reassembly buffer and
        REASSMFLAGS is a vector of flags indicating which bytes in
            the REASSMBUFFER contain valid user data.
        MAXRCVWS is the length of the reassembly buffer

    PINDEX:=0
    START:= (packet seq number)

    if  PRECEDE (START, RCVSEQ)  then
        begin
        PINDEX:=RCVSEQ-START
        START:=RCVSEQ
        end

    PTR:= [START - RCVSEQ + INPUTHEAD] MOD MAXRCVWS

    AMOUNT:= MIN (RCVWS, (packet text length) )

    for I:= PINDEX  until AMOUNT + PINDEX - 1  do
        begin
        REASSMFLAGS (PTR):=TRUE
        REASSMBUFFER (PTR):= (packet text field indexed by I)
        PTR:= (PTR+1) MOD MAXRCVWS
        end

    if START = RCVSEQ  then
        (notify USER of new data received at left window edge)

    return


HANDLEINT:
    RCVSEQ:= (packet seq number) + 1
    (notify TCP OUTPUT PROCESS to send ACK)
    (flush receive data)
    (notify USER of INTERRUPT request)

    return


HANDLEFIN:
    comment- handle a valid FIN arriving when connection ESTABLISHED

    CONNECTIONSTATE:=FINRECEIVED
    (notify TCP OUTPUT PROCESS to flush send data)
```

```
        (notify USER of remote close)
        RCVSEQ:= (packet seq number) + (packet text length)
        RCVSEQ:=RCVSEQ + CONTROLLENGTH (RCVPKTBUFFER)
        (notify TCP OUTPUT PROCESS to send ACK)
        (notify TCP OUTPUT PROCESS to send FIN)

        return



HANDLEDSN:
        comment- method for handling DSN is yet unresolved

        return


ADDRESSCHECK:
        comment- returns TRUE if the packet is for the one valid
            connection.  the foreign NET, TCP & PORT address and the
            local PORT addresses must agree with those of the open
            connection.

        if CONNECTIONSTATE = CLOSED  then
            return FALSE

        if (packet source NET field) notequal DNET  then
            return FALSE

        if (packet source TCP field) notequal DTCP  then
            return FALSE

        if (packet source PORT field) notequal DPORT  then
            return FALSE

        if (packet destination NET field) notequal SNET  then
            return FALSE

        if (packet destination TCP field) notequal STCP  then
            return FALSE

        if (packet destination PORT field) notequal SPORT  then
            return FALSE
        else
            return TRUE


HANDLESPECIALPACKET:
        comment- handle special functions or error message packets
```

```
case (packet control dispatch field)  of
    =SPECIALFUNCTION:
    begin
    case (packet control data octet)  of
        =RESETALL:
        if (packet source TCP field) = DTCP  then
            RESETCONNECTION

        =RESET:
        if ADDRESSCHECK and ACCEPTABLEACK  then
            RESETCONNECTION

        =QUERY:
        if ADDRESSCHECK  then
            (send status message)

    end

    =ERROR:
    if ADDRESSCHECK  then
        begin
        comment- process error directed at us.

        case (packet control data octet)  of
            =USYN:
            if CONNECTIONSTATE = SYNSENT  then
                (send a reset)
            else
                if CONNECTIONSTATE = SIMULINIT  then
                    INITCONNECTION

            =NONX:
            =INACC:
            begin
            case CONNECTIONSTATE  of
                =SIMULINIT:
                INITCONNECTION

                =ESTABLISHED:
                if INRCVWINDOW  then
                    (notify user process of error)

                =FINWAIT:
                if INRCVWINDOW  then
                    DELETECONNECTION

            end
        end
return
```

```
INITCONNECTION:
        comment- initialize connection state

        CONNECTIONSTATE:=SYNSENT
        (notify TCP OUTPUT PROCESS to flush send data)
        (flush receive data)
        comment- pick initial sequence number by adding a constant to
            last sequence number used on previous connection.
        SNDSEQ:=SNDSEQ + 1
        (notify TCP OUTPUT PROCESS to send SYN)
        return


CONTROLLENGTH (packet buffer):
        comment- returns the number of octets used in the packet by
            control functions.

        COUNT:=0
        if (packet SYN bit =1)  then
            COUNT:=1
        if (packet INT bit =1)  then
            COUNT:=COUNT +1
        if (packet FIN bit =1)  then
            COUNT:=COUNT +1
        if (packet DSN bit =1)  then
            COUNT:=COUNT +1
        return  COUNT


PRECEDE (PARM1, PARM2):
        comment- returns true if PARM2 - 2**16 < PARM1. PARM1 and PARM2
            both being 32 bit numbers.  this is just a special inwindow
            test that returns true if PARM1 precedes or equals PARM2
            in the circular sequence number space.


XMITERROR (ERROR CODE):
        comment- send an error message to the remote TCP. the error code
            is passed as a parameter. see [3] section 2.4.3 about
            possible error codes.

        (swap source & destination socket ids)
        (put input packet sequence number in ACK field)
        (set Control Dispatch to indicate error present)
        (put ERROR CODE in control octet)
        (notify TCP OUTPUT PROCESS to send error packet)
        (wait for completion of error send)
        (set error send complete flag to false)
        return
```

DELETECONNECTION:

```
        comment- set the connection state to CLOSED, notify the user of
            close completion and flush all the queues and stuff.

        CONNECTIONSTATE:=CLOSED
        (notify user process of CLOSE completion)
        (notify TCP OUTPUT PROCESS to flush send data)
        return
```

7 TCP OUTPUT PROCESS LOGIC DOCUMENTATION

```
TCP OUTPUT PROCESS:
        comment- initialize Net interface.

        INITIALIZENETWORK
        (notify TCP INPUT PROCESS of initialization complete)

  LOOP: (wait for work to process)

        if (need to send error)  then
            begin
            (move packet from RCVPKTBUFFER into SENDPKTBUFFER)
            SENDPACKET
            (notify TCP INPUT PROCESS that error has been sent)
            (set need to send error flag to false)
            end

        if (need to flush send data)  then
            begin
            comment- flush the send data and retransmission queues.
                first remove any outstanding SENDs.

            if (need to send data)  then
                begin
                (set need to send data flag to false)
                (notify USER of send completion, ready for new send)
                end
            BUFBYTECOUNT:=0

            comment- then remove any data or control to be retransmitted
            RTXCOUNT:=0
            RTXCONTROL:=0
            RTXWAKEUP:=0
            (set need to retransmit data flag to false)
            (set need to retransmit control flag to false)
            (set need to flush send data flag to false)
            end

        if (need to send SYN, INT or FIN)  then
            begin
            comment- construct a packet and add the appropriate control
                bits.

            INITIALIZEPACKET
            if (need to send SYN)  then
                (set packet SYN bit =1)
            if (need to send INT)  then
                (set packet INT bit =1)
            if (need to send FIN)  then
                (set packet FIN bit =1)
            (set need to send SYN, INT and FIN flags to false)
```

```
        SENDPACKET
        RTXCNTRLSEQ:=SNDSEQ
        RTXCONTROL:= (packet control word field)
        SNDSEQ:=SNDSEQ + CONTROLLENGTH (SENDPKTBUFFER)
        end

    if (need to send data)  then
        begin
        comment- SPACELEFT is amount of space left in send window
            RTXCOUNT is the number of bytes in the retransmission
                queue,
            MAXRTXCOUNT is the length of the retransmission queue
                buffer,
            RTXSPACE is amount of space left in retransmission queue
            MAXPKTSIZE is maximum number of data bytes in packet
            BUFBYTECOUNT is number of bytes in user send buffer
            BUFFERPOINTER is address of start of user send buffer.

        SPACELEFT:=SNDWS - SNDSEQ + LSWEDGE
        RTXSPACE:=MAXRTXCOUNT - RTXCOUNT
        COUNT:= MIN (SPACELEFT, BUFBYTECOUNT, RTXSPACE, MAXPKTSIZE)
        if COUNT notequal 0  then
            begin
            INITIALIZEPACKET
            if RTXCOUNT = 0  then
                RTXDATASEQ:=SNDSEQ
            PKTPTR:= (index of start of packet text area)
            for I:=1 until COUNT  do
                begin
                (copy byte from user buffer into packet)
                (copy byte from buffer into retransmission queue)
                BUFFERPOINTER:=BUFFERPOINTER+1
                RTXPOINTER:= (RTXPOINTER + 1) MOD MAXRTXCOUNT
                PKTPTR:=PKTPTR+1
                BUFBYTECOUNT:=BUFBYTECOUNT-1
                RTXCOUNT:=RTXCOUNT+1
                end
            (set packet text length equal COUNT)
            SENDPACKET
            SNDSEQ:=SNDSEQ + COUNT
        if BUFBYTECOUNT = 0  then
            begin
            (set need to send data flag to false)
            (notify USER of send completion, ready for next send)
            end
        end

    if (need to retransmit data) and CONNECTIONSTATE = ESTABLISHED
        then
        begin
        comment- RTXWAKEUP counts the number of retransmissions
            sent without receiving any ACKs back. It is cleared
```

```
        in HANDLEACK when a valid one comes in and incremented
        by the clock interrupt routine before notifying the SEND
        process to retransmit data or control.  if the data
        remains on the retransmission queue too long, the user
        is notified.

    if RTXWAKEUP greaterthan maximum allowed  then
        comment- the foreign TCP has failed to ACK data that has
            been waiting.  it is assumed that the destination
            is not responding

            (notify USER that destination TCP not responding)
    else
        begin
        comment- now retransmit the data queued up.

        SPACELEFT:=SNDWS - RTXDATASEQ + LSWEDGE
        COUNT:= MIN (SPACELEFT, RTXCOUNT, MAXPKTSIZE)
        if COUNT notequal 0  then
            begin
            INITIALIZEPACKET
            RTXPTR:=RTXPOINTER
            PKTPTR:=(index of start of packet text area)
            for I:=1 until COUNT  do
                begin
                SENDPKTBUFFER(PKTPTR):=RTXBUFFER(RTXPTR)
                PKTPTR:=PKTPTR+1
                RTXPTR:= (RTXPTR+1) MOD MAXRTXCOUNT
                end
            (packet sequence number):= RTXDATASEQ
            (packet text length):= COUNT
            SENDPACKET
            end
        end
    (set need to retransmit data flag to false)
    end

if (need to retransmit control)  then
    begin
    if RTXCONTROL notequal 0  then
        begin
        if RTXWAKEUP greaterthan maximum allowed  then
            begin
            (notify USER that destination TCP not responding)
            if RTXCONTROL equals FIN packet  then
                comment- a FIN packet has timed-out. so close
                    the connection anyway.
                DELETECONNECTION
            end
        else
            begin
            SPACELEFT:=SNDWS - RTXCNTRLSEQ + LSWEDGE
```

```
                if SPACELEFT greaterthan 0  then
                    begin
                    INITIALIZEPACKET
                    (packet sequence number):= RTXCNTRLSEQ
                    (packet control word):= RTXCONTROL
                    SENDPACKET
                    end
                end
            end
        (set need to retransmit control flag to false)
        end

    if (need to send ACK)  then
        begin
        INITIALIZEPACKET
        SENDPACKET
        end

    goto LOOP


INITIALIZEPACKET:
    comment- initialize the internet header

    (move SNDSEQ into packet seq field)
    (move foreign socket id into packet)
    (move local socket id into packet)
    (move RCVWS into packet)
    (zero out rest of packet header)
    if (need to send ACK)  then
        begin
        comment- piggyback ACK onto data or control packet
        (set packet ACK bit =1)
        (put receive left window edge in ACK field)
        (set need to send ACK flag to false)
        end
    return


SENDPACKET:
    comment- calculate a checksum and put it in the header and then
        sent it.

    (put zero in packet checksum field)
    (packet checksum field):=CHECKSUM (SENDPKTBUFFER)
    NETOUTPUT (SENDPKTBUFFER)
    return
```

8 ARPANET INTERFACE LOGIC DOCUMENTATION

```
INITIALIZENETWORK:
        comment- initialize the device drivers and send NOPs to IMP

        (reset network device driver)

        (construct IMP NOP message)

        for I:=1 until 4  do
            begin
            comment- send four NOPs to the IMP

            (give NOP message to network driver to send)

            (wait until message sent)
            end

        return


NETINPUT (PACKETBUFFER):
        comment- start input from IMP

  LOOP: (start message input from IMP)

        (wait until finished)

        if (message not on experimental links, 155-158)  then
            goto LOOP

        if (message type notequal regular or minimum effort)  then
            goto LOOP

        comment- return message to TCP

        return


NETOUTPUT (PACKETBUFFER):
        (initialize IMP-HOST header)

        (give to network device driver to send)

        (wait until message sent)

        return
```

References

1. Bolt Beranek and Newman Inc., "Interface Message Processor," Report No. 1822, December 1975.

2. R. C. Sunlin, "Packet Radio Channel Access Protocol Program," Packet Radio Note No. 144, 29 September 1975.

3. V. Cerf, Y. Dalal, C. Sunshine, "Specification of Internet Transmission Control Program," INWG Note No. 72, December 1974 (Revised).

4. V. Cerf, R. Kahn, "A Protocol for Packet Network Interconnection," IEEE Transactions on Communication, Vol. COM-22, Number 5, May 1974.

DISTRIBUTION

ARPA

Director (2 copies)
ATTN: Program Management
Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

ARPA/IPT
1400 Wilson Boulevard
Arlington, VA 22209

Dr. Robert Kahn
Mr. Steven Walker

Bell Laboratories

Dr. Elliot N. Pinson, Head
Computer Systems Research Dept.
Bell Laboratories
600 Mountain Avenue
Murray Hill, New Jersey 07974

Dr. Samuel P. Morgan, Director
Computing Science Research
Bell Laboratories
610 Mountain Avenue
Murray Hill, New Jersey 07974

Dr. C. S. Roberts, Head
The Interactive Computer Systems
Research Department
Bell Laboratories
Holmdel, New Jersey 07733

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

Mr. Jerry D. Burchfiel
Mr. R. Clements
Mr. A. McKenzie
Mr. J. McQuillan
Mr. R. Tomlinson
Mr. D. Walden

Burroughs Corporation

Dr. Wayne T. Wilner, Manager
Burroughs Corporation
3978 Sorrento Valley Boulevard
San Diego, CA 92121

Mr. David H. Dahm
Burroughs Corporation
Burroughs Place
P. O. Box 418
Detroit, MI 48232

Mr. B. A. Creech, Manager
New Product Development
Burroughs Corporation
460 Sierra Madre Villa
Pasadena, CA 91109

Cabledata Associates

Mr. Paul Baran
Cabledata Associates, Inc.
701 Welch Road
Palo Alto, CA 94304

California, University - Irvine

Prof. David J. Farber
University of California
Irvine, CA 92664

California, University - Los Angeles

Professor Gerald Estrin
Computer Sciences Department
School of Engineering and Applied Science
Los Angeles, CA 90024

Professor Leonard Kleinrock
University of California
3732 Boelter Hall
Los Angeles, CA 90024

Mr. William E. Naylor
University of California
3804-D Boelter Hall
Los Angeles, CA 90024

Collins Radio Group
1200 N. Alma Road
Richardson, Texas 75080

Mr. Don Heaton
Mr. Frederic Weigl

Defense Communications Engineering Center

Dr. Harry Helm
DCEC, R-520
1860 Wiehle Avenue
Reston, VA 222090

General Electric

Dr. Richard L. Shuey
General Electric Research and Development Center
P. O. Box 8
Schenectady, New York 12301

Dr. A. Bell Isle
General Electric Company
Electronics Laboratory
Electronics Park
Syracuse, New York 13201

Mr. Ronald S. Taylor
General Electric Company
175 Curtner Avenue
San Jose, CA 95125

General Motors Corporation
Computer Science Department
General Motors Research Laboratories
General Motors Technical Center
Warren, MI 48090

Dr. George C. Dodd, Assistant Head
Mr. Fred Krull, Supervisory Research Engineer
Mr. John Boyse, Associate Senior Research Engineer

Hawaii, University of
The ALOHA System
2540 Dole Street, Holmes 486
Honolulu, Hawaii 96822

Professor Norman Abramson

Hughes Aircraft Company

Mr. Knut S. Kongelbeck, Staff Engr.
Hughes Aircraft Company
8430 Fallbrook Avenue
Canoga Park, CA 91304

Mr. Allan J. Stone
Hughes Aircraft Corporation
Bldg. 150 M.S. A 222
P. O. Box 90515
Los Angeles, CA 90009

Hughes Aircraft Company
Attn: B. W. Campbell 6/E110
Company Technical Document Center
Centinela and Teale Streets
Culver City, CA 90230

IBM

Dr. Patrick Mantey, Manager
User Oriented Systems
International Business Machines Corp.
K54-282, Monterey and Cottle Roads
San Jose, CA 95193

Dr. Leonard Y. Liu, Manager
Computer Science
International Business Machines Corp.
K51-282, Monterey and Cottle Roads
San Jose, CA 95193

Mr. Harry Reinstein
International Business Machines Corp.
1501 California Avenue
Palo Alto, Ca 94303

Illinois, University of

Mr. John D. Day
University of Illinois
Center for Advanced Computation
114 Advanced Computation Bldg.
Urbana, Illinois 61801

Institut de Recherches d'Informatique et d'Automatique (IRIA)
Reseau Cyclades
78150 Rocquencourt
France

Mr. Louis Pouzin
Mr. Hubert Zimmerman

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

Dr. Marty J. Cohen
Mr. Steven D. Crocker
Dr. Steve Kimbleton
Mr. Keith Uncapher

London, University College

Professor Peter Kirstein
UCL
Department of Statistics &
Computer Science
43 Gordon Square
London WC1H OPD, England

Massachusetts Institute of Technology

Dr. J. C. R. Licklider
MIT
Project MAC - PTD
545 Technology Square
Cambridge, Massachusetts 02139

MITRE Corporation

Mr. Michael A. Padlipsky
MITRE Corporation
1820 Dolly Madison Blvd.
Westgate Research Park
McLean, VA 22101

Network Analysis Corporation
Beechwood, Old Tappan Road
Glen Cove, New York 11542

Mr. Wushow Chou
Mr. Frank Howard

National Bureau of Standards

Mr. Robert P. Blanc
National Bureau of Standards
Institute for Computer Sciences
and Technology
Washington, D. C. 20234

Mr. Ira W. Cotton
National Bureau of Standards
Building 225, Room B216
Washington, D. C. 20234

National Physical Laboratory
Computer Science Division
Teddington, Middlesex, England

Mr. Derek Barber
Dr. Donald Davies
Mr. Roger Scantlebury
Mr. P. Wilkinson

National Security Agency
9800 Savage Road
Ft. Meade, MD 20755

Mr. Dan Edwards
Mr. Ray McFarland

Norwegian Defense Research Establishment
P. O. Box 25
2007 Kjeller, Norway

Mr. Yngvar G. Lundh
Mr. P. Spilling

Oslo, University of

Prof. Dag Belsnes
EDB-Sentret, University of Oslo
Postbox 1059
Blindern, Oslo 3, Norway

Rand Corporation
1700 Main Street
Santa Monica, CA 90406

Mr. S. Gaines
Mr. Carl Sunshine

Rennes, University of

M. Gerard LeLann
Reseau CYCLADES
U.E.R. d'Informatique
B. P. 25A
35031-Rennes-Cedex, France

Stanford Research Institute
333 Ravenswood Avenue
Menlo Park, CA 94025

Ms. E. J. Feinler
Augmentation Research Center

Dr. Jon Postel
Augmentation Research Center

Mr. D. Nielson Director
Telecommunication Sciences Center

Dr. David Retz
Telecommunication Sciences Center

System Development Corporation

Dr. G. D. Cole
System Development Corporation
2500 Colorado Avenue
Santa Monica, CA 90406

Telenet Communications, Inc.
1666 K Street, NW
Washington, D. C. 20006

Dr. Holger Opderbeck
Dr. Lawrence G. Roberts
Dr. Barry Wessler

Transaction Technology Inc.

Dr. Robert Metcalfe
Director of Technical Planning
Transaction Technology Inc.
10880 Wilshire Blvd.
Los Angeles, CA 90024

Defense Communication Agency

Dr. Franklin Kuo
4819 Reservoir Drive
Washington, D. C. 20007

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Mr. David Boggs
Dr. William R. Sutherland

STANFORD UNIVERSITY

Digital Systems Laboratory

Mr. Ronald Crane
Mr. Yogen Dalal
Ms. Judith Estrin
Professor Michael Flynn
Mr. Richard Karp
Mr. James Mathis
Mr. Darryl Rubin
Mr. Wayne Warren

Digital Systems Laboratory Distribution

Computer Science Department - 1 copy
Computer Science Library - 2 copies
Digital Systems Laboratory Library - 6 copies
Engineering Library - 2 copies
IEEE Computer Society Repository - 1 copy

Electrical Engineering

Dr. John Linvill