FG (12)

ADA024823

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) TN-33	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER (9)
4. TITLE (and Subtitle) (6) ARPA INTERNETWORK PROTOCOLS PROJECT STATUS REPORT		5. TYPE OF REPORT & PERIOD COVERED Technical Note, 15 Nov 75 - 15 Feb 76 ~~Nov. 15, 1975 - Feb. 15, 1976~~
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Vinton G. Cerf		8. CONTRACT OR GRANT NUMBER(s) (15) MDA903-76C-0093, ARPA Order ~~No.~~ 2494
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford Electronics Laboratories Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 6T10 (11) 15 Feb 76
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Information Processing Techniques Office 1400 Wilson Ave., Arlington, VA 22209		12. REPORT DATE 2-15-76 / 13. NO. OF PAGES 55
14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office) Mr. Philip Surra, Resident Representative Office of Naval Research Durand 165, Stanford University		15. SECURITY CLASS. (of this report) UNCLASSIFIED (12) 53p.
		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"

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)

Protocols, interprocess communication, networks, Gateways, ARPANET, Packet Radio Network, computer communication

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

This quarterly report summarizes changes in the TCP specification, measurement of VDH overhead, plans for the single connection TCP/TELNET, and mentions the development of a very small LSI-11 interface to the BBN IMP.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408 071

[illegible]

ARPA INTERNETWORK PROTOCOLS PROJECT

STATUS REPORT

FOR THE PERIOD NOVEMBER 15, 1975 - FEBRUARY 15, 1976

by

Vinton G. Cerf
Principal Investigator

February 25, 1975

Technical Note #83

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

1. TCP Implementation

As this quarter came to a close, a series of meetings were held at Stanford with representatives from University College London (Mr. A. Hinchley) the Norwegian Defense Research Establishment (Mr. P. Spilling), Bolt Beranek and Newman (Mr. R. Tomlinson) and the SU-DSL staff. Also attending a portion of these meetings were Dr. J. Postel, Mr. R. Rom, and Mr. L. Garlick, all of Stanford Research Institute, Mr. T. Chandler (Collins Radio), and Mr. A. D. Owen (BBN).

The purpose of the TCP-related meetings was to determine the appropriate changes and additions needed in the TCP specification [1] to reflect changes made during implementation and testing, and to incorporate complete specification of the new resynchronization and CLOSE processing design, as well as modifications to error and other control processing.

This rewrite of the TCP specification is important, not only to reflect reality, but also to provide a more up-to-date basis for inclusion in the AUTODIN II specification. Dr. J. Postel (SRI) has been tasked with the integration of priority and security capability into the TCP and SU-DSL has agreed to provide an updated text on which to base the DCA work.

1.1 Very Distant Host Measurements

During the months of December (1975) and January (1976), we undertook extensive and detailed timing measurements of the ELF VDH behavior to ascertain the degree to which VDH performance affected total TCP bandwidth and delay. The three most fundamental discoveries were

a) ELF VDH and IMP VDH do not handle the two VDH logical channels compatibly.

b) ELF VDH introduced unnecessary retransmissions into the system.

c) ELF VDH did not pipeline packets to the outgoing modem (i.e. ELF VDH started I/O for packet 1 and waited for output completion before starting output for packet 2).

The statistics gathered were voluminous (a typical case of "datarhea") and reflect changes made in the ELF VDH to deal with problem (b). We know how to solve problems (a) and (c), but have put the task on a backup queue, since Packet Radio software (see TCPO, TELNET in section 2) currently has priority for our programming resources.

Problem (a) has to do with the way in which the IMP VDH actually transmits packets. In the design specification [2], outgoing messages are split into a leader and zero or more (up to 8) data packets, each at most 63 x 16 bit words long. Packets are assigned alternately to logical channels 0 and 1. The IMP VDH code, however, randomly selects packets to be sent on each logical channel (i.e. the outgoing buffers are filled, in order, by the IMP code, but sent in random order). The ELF VDH code assumes packets will be sent (and therefore received) in order (channel 0, channel 1, channel 0...).

If a packet arrives on channel 2 when the ELF VDH is expecting new data on channel 1, the ELF code assumes an error has caused the channel 1 packet to have been dropped and discards the channel 2 packet. Thus a certain fraction of valid packets are discarded by the ELF VDH. This problem can be overcome by rewriting the ELF VDH so that packets arriving with proper "ODD/EVEN" bits set to indicate a new transmission on the associated logical channel are queued and accepted in the correct order.

Problem (c) requires that the VDH not block, waiting for completion of an output on one channel if a second buffer can be sent on the alternate channel. If both channels are in use (really, if the next correct channel to send on), then the VDH should wait until the channel is available (by blocking and giving up the CPU until the next channel is free).

Problem (b) was just a bug in the TCP ELF implementation which was easily repaired. Some packets were being retransmitted before they could have been acknowledged.

In a throughput experiment conducted January 14, 1976, by J. Mathis, the TCP was run using a traffic generator which sent traffic to itself via the IMP. Three TCP letters were allowed to enter the pipeline before requiring an acknowledgment. Transmission continued for 1 minute. Table I summarizes the results. Taking just the 70 character letter results, we can form a model of the line utilization and some of the delays. It is worth noting that 10 character and 70 character letters both fit in one RTP packet (including the TCP header), so the relative bandwidth of 6.6-6.7 TCP letters/second is probably less affected by actual letter length than by the number of RTP packets required to carry it.

The statistics shown in Table I are slightly inconsistent owing to the asynchronous way in which they were gathered (Table values were updated whenever an event occurred affecting the table), but are approximately consistent. Table 2 illustrates the approximate consistency checks. Each TCP letter sent causes two RTP packets to be sent to the IMP. These data packets are sent back, along with an RTP packet containing a RFNM. This accounts for 1215 of the RTP data packets received (in the 70 character letter case) and

	70 character letters	10 character letters
Total letters sent	405	393
RTP packets received w/o error	2290	2094
Non-duplicate data packets received	1269	1234
Duplicates received	0	0
Null packets received	789	604
Packet errors	0	0
Messages received on non TCP links	6	4
Messages received on TCP links	841	820
Packets received out-of-sequence	36	128 (!)
Original packet sends	844	822
Retransmission (RTP)	0	0
Null packets sent	943	1004
Special packets sent	197	129 (!)
Wait for IORB	133	58
Wait for busy channel	7	14
Calls to DESCHED in WAIT channel	13	25
Storage allocation failures	0	0
TCP letters/sec	6.7	6.6
TCP bit rate/sec	3730	528

TCP/VDH Bandwidth Experiment 1/14/76

Table I

	2290	total RTP packets received w/o error
	-1269	total non-duplicate data packets received
	1021	
	- 789	total null packets received
	232	
	- 197	total special packets sent*
	35	unaccounted for
	405	TCP letters sent
actual (841)	810	TCP messages received (TCP letters + RFNM's)
actual (1269)	1215	RTP packets (TCP HEADER + TEXT, ARPA LEADER, RFNM) received

VDH Statistics Consistency Check

Table II

* Special packets like "HELLO" get "I HEARD YOU" responses, so we can assume approximately equal numbers of sent and received special packets.

approximates the 1269 non-duplicate data packets actually received. By the same analysis, the 841 messages received are approximately 405 TCP letters and 405 RFNM's. Some of the variation might have been the result of TCP retransmissions. Furthermore, the 36 "out of sequence" packets would have caused the ELF VDH to force retransmission from the IMP owing to the ELF VDH assumption that packets will arrive in sequence for alternate channels. Nor would the ELF VDH have detected the retransmissions, since it would not have acknowledged receipt of the out-of-sequence arrivals. The 789 null packets received correspond roughly to the number of RTP data packets sent (2 x 405 = 810); it is possible that sometimes, both logical channels were acknowledged by a single null packet, or that a data packet carried the acknowledgment. The disparity between 405 TCP internet packets sent and $\frac{844}{2}$ = 422 TCP link messages sent may indicate some TCP retransmission. The 841 TCP link messages received (as opposed to 844 sent) merely reflects delays in acknowledgment at the TCP level.

Earlier TCP tests indicate that a self-loop test with three TCP packets in the pipeline will avoid any empty TCP acknowledgment packets, so we can estimate the actual channel bit rate in use based on the model given below.

VDH Channel Bit Rates

Just as a simple way to characterize the packets that flow on the VDH link, let us define several shorthand terms.

(BLO)	Basic Line Overhead = 88 bits (SYN,SYN,DLE,STX,RTP Control (16 bits), ..., DLE,ETX,CRC (24 bits)) This is the format of a standard VDH packet
(NRP)	NULL RTP Packet = 88 bits (see format above)
(SP)	SPecial Packets = 88 bits (HELLO, I-HEARD-YOU...)

(ALP)	ARPA LEADER Packet = BLO + 32 bits = 120 bits
(TAP)	TCP Acknowledgment Packet = BLO + 288 bits (TCP header) = 376 bits
(TDP/H)	TCP Data Packet with header = TAP + [data: 0-720 bits] = 376 bits (0 data) - 1096 bits (720 data) This packet is the first one, which includes the TCP header.
(TDP/NH)	TCP Data Packet without header = BLO + [data: 0-1008 bits] = 88 bits (0 data) - 1096 bits (1008 data)

The RTP periodically sends "HELLO" packets and responds to any received with "I-HEARD-YOU" Packets. The normal ARPANET message is split into a leader part and data portions divided into 63 16 bit words each. The leader part is sent as an ALP (see above) and the receiving RTP sends a null packet containing an acknowledgment. If there is a packet awaiting transmission on the other side, that packet will carry the acknowledgment bits.

The expected measured activity is characterized below.

TCP TRANSMISSION SIDE	IMP TRANSMISSION SIDE
ALP -> (send leader)	
	<- NRP (ack leader)
TDP/H-> (letter text)	
	<- NRP (ack text)
	<- ALP (send leader)
NRP-> (ack leader)	
	<- TDP/H (letter text)
NRP-> (ack text)	
	<- ALP (RFNM)
NRP-> (ack RFNM)	

The actual activity differs somewhat, as table III shows, indicating that fewer null RTP packets were sent or received than expected. This merely shows that the output queues at the ELF and IMP VDH were often non-empty, eliminating the necessity for null acknowledgment RTP packets. The actual line utilization is about 20% in each direction, assuming a nominal 50 kbits/second available full-duplex capacity between ELF and the IMP.

The VDH measurements described above do not illustrate the actual delays in the system. To observe these, we implemented two kinds of timestamping procedures, one associated with individual TCP internet packets, and one associated with events occurring inside the TCP/VDH code. The latter involves the maintenance of a resident timestamp table in the TCP code which is dumped on closing a TCP connection. The events which are timestamped are scattered throughout the VDH and TCP code (e.g. arrival of inbound packets, processing of outbound packets, etc.).

The TCP packet-based timestamping system will be described in more detail in a forthcoming technical report, but the preliminary results indicate that the TCP introduces about 20 ms of delay from the time a user program asks to SEND a letter to the time the letter (one packet letter) is handed to the VDH. On the input side, it takes about 42-43 ms from the time a packet is handed to the TCP to the time the user is signalled that a letter has been received. In the experiments performed so far, the "echoer" process takes about 8 ms to be awakened and to send the next packet out. Thus, we estimate that the TCP alone should be able to sustain a rate of 1000/70 - 14 packets/second [single packet letters]. The degradation to 6.6 packets/second must be accounted for by delays in the VDH processing. We would have to observe

60 second statistics

Sent from ELF VDH	Received by ELF VDH
422 (422) ALP [leader]	421 (421) ALP [leader]
422 (422) TDP/H [TCP packet]	421 (421) TDP/H [TCP packet]
943 (1269) NRP [RTP acks]	789 (842) NRP [RTP acks]
197 (96) NRP [RTP specials]**	421 (421) ALP [RFNM]
	6 (0) ALP + 88 bits [NCP RFC's]*
	197 (96) NRP [RTP specials]**

ELF Transmit side analysis

422 (ALP + TDP/H) + (943 + 197) NRP

= 422 (120 + 376 + 560) + 1140 (120)

= 422 (1056) + 1140 (120) = 582432 bits/60 sec = 9.7 kbits/sec

ELF Receive side analysis

421 (ALP + TPP/H + ALP) + 789 (NRP) + 6 (ALP + 88) + 197 (NRP)

= 421 (120 + 376 + 560 + 120) + 789 (120) + 6 (120 + 88) + 197 (120)

= 421 (1176) + (789 + 197) (120) + 6 (208)

= 421 (1176) + 986 (120) + 6 (208) = 614664/60 = 10.2 kbits/sec

Actual full-duplex delivered data bandwidth = 405 x 70 x 8/60 = 3.8 kbits/sec (i.e. 3.8 kbits/sec of data in each direction).

* These were RFC's from MIT-DMS and probes from other sites

** We haven't accounted for the nearly doubled number of Hello/I heard you's. They should be generated in pairs every 1.25 seconds.

Analysis of ELF Send/Receive Line Bandwidth

Table III

delays of 150 ms/packet to account for the bandwidths achieved thus far, and would therefore predict VDH processing delays (and IMP RTP delays, as well as transmission delays) on the order of 80 ms.

In a single experiment performed by J. Matnis on 1/10/76, the SU-DSL TCP opened and established a TCP connection which looped through the SUMEX IMP and back to SU-DSL. 10 characters of data were sent with each letter. le IV summarizes the results. In this particular experiment, the retrans-sion process awakened asynchronously every 4 seconds. TCP acknowledgments were sent by the retransmission routine, so this experiment was not an attempt at maximum throughput. It is evident from Table IV that on the outgoing side, about 20 ms are needed to prepare a letter for VDH transmission, and another 10 ms elapse while these packets are sent to the IMP. 54 ms more are needed to echo the data packets through the IMP and ELF VDH to the TCP. The TCP takes another 33 ms and notifies user of internet packet arrival. Since arriving internet packets are printed on a CRT, increased delays in user processing show up in Table IV in the form of 50 ms delay before user does a new RECEIVE. TCP ack is sent only if the retransmission process spots an unacknowledged "new receive." In our case, this delays user send processing by about 1.1 seconds. Table V summarizes the important delays, accounting for timestamp overhead as well (see next section).

	interval	cumulative
TCP Send/Start RTP*	18.5 ms	18.5 ms
RTP completes unacked transmission	9.9 ms	28.4 ms
RTP receives looped TCP packet, signals TCP	38.9 ms	67.3 ms
RTP RFNM processing delay/TCP wakes up	14.7 ms	82.0 ms
TCP signals user, letter received	28.5 ms	110.5 ms
TCP receives scheduler winds up	4.8 ms	115.3 ms
User executes RECEIVE**	49.9 ms	165.0 ms
TCP RECEIVE processing	12.8 ms	177.8 ms
IDLE/TCP Retrans. awakens***	866.4 ms	1044.2 ms
TCP retrans routine sends ACK	6.5 ms	1050.7 ms
RTP packet processing/TCP awakens	201 ms	1251.6 ms
TCP signals user, letter sent and acked	7.4 ms	1259.0 ms

* RTP, reliable transmission package (VDH software)

** includes delay to display letter on CRT

*** RETRANS awakens asynchronously every 4 seconds

TABLE IV

TCP loop/delay measurement

Round-trip TCP SEND to TCP RECEIVE

TCP	51.8 ms	-	5.7 ms	(Timestamp overhead) =	46.1 ms
RTP	63.5 ms	-	12.4 ms	(Timestamp overhead) =	50.1 ms
User	49.9 ms	-	0.2 ms	(Timestamp overhead) =	49.7 ms

Rough allocation of delay:

TCP = 46.1/145.9 = 31.6%

RTP = 50.1/145.9 = 34.3%

User = 49.7/145.9 = 34.1%

Excluding the user:

TCP = 46.1/96.2 = 47.9%

RTP = 50.1/96.2 = 52.1%

Delays in TCP/RTP Corrected for Error

Table V

Other Overhead Measurements

To further understand the impact on CPU utilization of the VDH, we set up a remote loop at the IMP side of the VDH line and traced the activity of the ELF RTP in the absence of any TCP traffic. Since the ELF CPU was isolated from the ARPANET, only locally generated RTP activity would be observed. As we expected, at regular 1.25 sec. intervals, the RTP sent "HELLO" packets which it answered with "I HEARD YOU" packets. In the absence of any other traffic, we found that the RTP continued to transmit null packets every 200 ms. This has since been changed so that the retransmission routine times out every 100 ms, but is not scheduled at all if no output channels are in use. The November 17, 1975, measurements of overhead are therefore an upper bound, and amount to about 3% (25 ms every 1250 ms for Hello and I Heard You processing: 2%; and 2 ms every 200 ms for null packet: 1%).

The overhead for actually gathering delay statistics is about 350 μsec per timestamp when called from the RTP and about 200 μsec when called from the TCP (the difference has to do with additional register saving in the RTP). This includes the time to store away the 32 bit time and identity of the caller in the monitor table. The clock times are accurate to 40 μsec. The graininess in clock times are expected to cancel each other after enough measurements are made.

Recent VDH changes/PDP-11 changes

In mid-February, BBN installed an additional 8K of memory in the SUMEX IMP, in part for the purpose of running the IMP VDH in extended memory. We have observed an increase in delay (and decrease in throughput) of about 5% as a result of this change. A more serious influence on throughput is a change in our PDP-11 clock cycle time from 245 ns to 280 ns. This slows everything down by about 15%. Consequently, measurements made after 7 March 1976 will appear to show poorer performance than before. As of the time of this writing (10 March) we had achieved about 7 packets/second with the exerciser looped to itself, but going through the IMP. This may degrade to about 6 packets/second again, as a result of the basic clock rate change (which was made to improve the reliability of the PDP-11/20).

1.2 Resynchronization

In the original TCP specification, we did not describe the method by which connections would resynchronize themselves. In a recent DSL Technical Note [3], we described the problem of knowing when to resynchronize. R. Tomlinson [BBN], Y. Dalal [SU-DSL], Dr. Belsnes [OSLO], and more recently, J. Mathis [SU-DSL] have contributed greatly to our understanding of this complex issue [4-6].

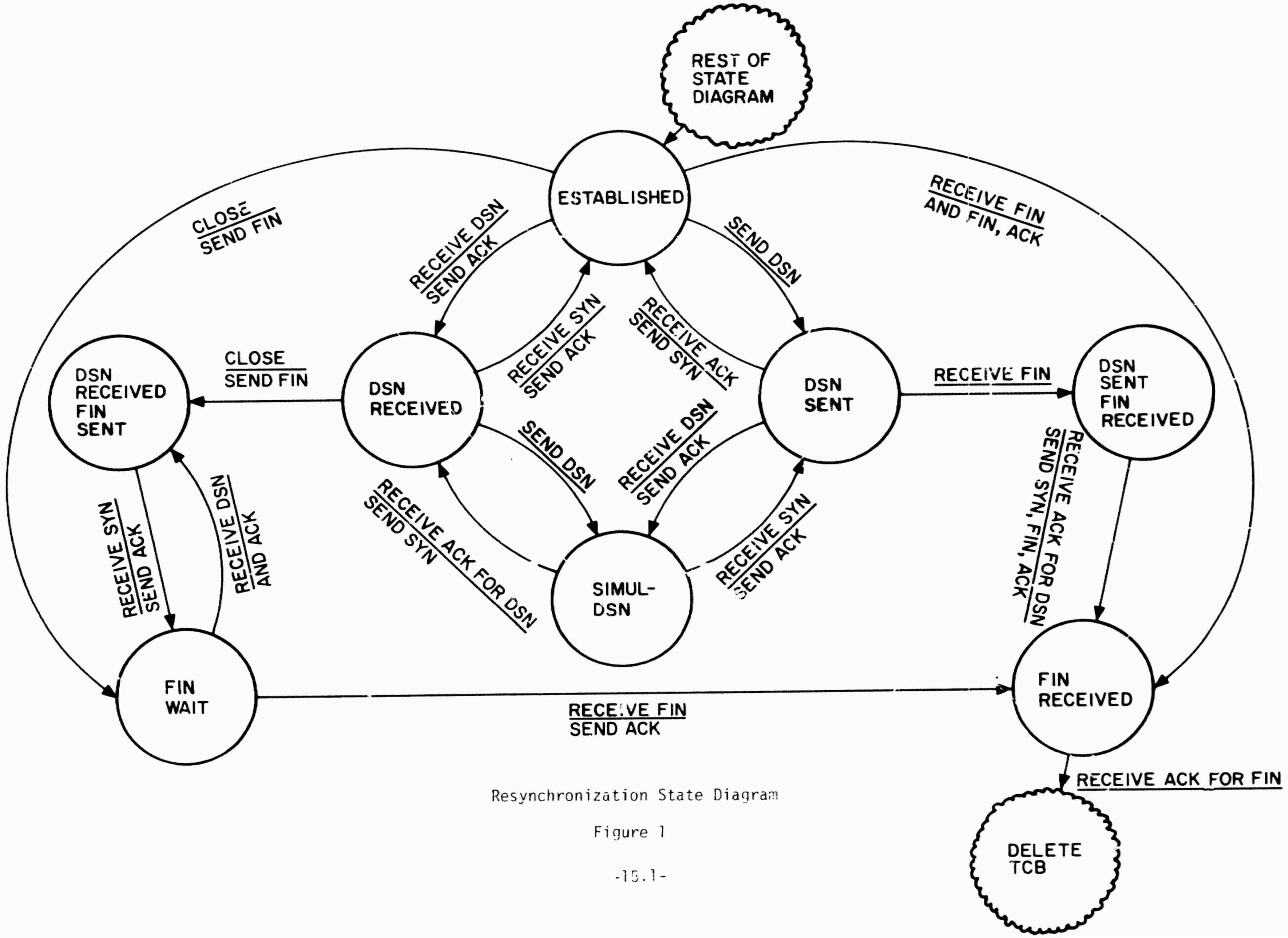
In meetings during the week of 16-20 February, attended variously by representatives from NDRE (Paal Spilling), UCL (Andrew Hinchley), SRI (Jon Postel, Larry Garlick, Raphi Rom), SU-DSL (Yogen Dalal, Richard Karp, Jim Mathis, Judy Estrin, Vint Cerf) and BBN (Ray Tomlinson), we discussed a variety of resynchronization methods ranging from forced resynchronization of sequence numbers on both sides of a full-duplex connection to strategies only requiring that one set of sequence numbers be resynchronized (remember, each transmitter selects his own sequence numbers).

The surprising result was that resynchronization was generally more complex than initial synchronization. We started with the assumption that either or both sides could resynchronize their sequence numbers at any time (even just after establishing a connection). The earliest model was to send a DSN packet (DSN is a control bit in the TCP packet header meaning "desynchronize" sequence numbers in this direction). The sender of the DSN would await an ACKnowledgment (the DSN takes up 1 byte of sequence space so it can be unambiguously placed in the transmission stream) and then send a SYNchronize packet.

The fundamental problem with this simple-sounding strategy is that the receiver of a DSN, after acknowledging it, could not distinguish between the receipt of an old SYN packet from earlier resynchronizations on the initial connection establishment. In an attempt to rescue matters, it was proposed that a SYN,ACK packet be sent, rather than a naked SYN, so as to validate it, but if the receiver of the DSN is also in the process of desynchronizing, there can be a basic ambiguity in the value of the ACK field (old or new sequence numbers). J. Mathis uncovered this ambiguity, much to the astonishment of the rest of us who thought everything was fine.

We finally compromised on a strategy which permits unilateral or simultaneous resynchronization, but which relies on the notion that no previous SYN packets are around any longer from the initial connection exchange or later resynchronizations. In principle this is not unreasonable if the packet lifetime is much shorter than the cycle time of the sequence number space. In our case, the packet lifetime is probably on the order of a few tens of seconds while the cycle time is 4.5 hours. The resulting state diagram is not very compact. The original TCP specification has 8 states and the new one has 13. The size of the state diagram need not, however, imply a larger implementation, since suitable encoding of the state information will allow the state machine to be factored into several smaller ones (as in BBN's implementation of the TCP). Figure 1 illustrates just the resynchronization portion of the TCP connection state diagram. The "FIN received" and FIN wait" states are a normal part of the TCP state diagram, but are shown since FIN processing must also be managed during resynchronization.

The basic cycle for unilateral resynchronization is for the sender to go from "ESTABLISHED" to "DSN sent" and back, while the receiver goes from "ESTABLISHED" to "DSN received" and back. Once a sender has transmitted DSN, he must continue to retransmit all unacknowledged packets preceding the DSN as well since the receiver of the DSN cannot acknowledge it until he has received all preceding packets as well. Once resynchronization is started, the sender of the DSN must not send any other packet (except those mentioned above), even if the user has said "CLOSE." Thus, the exchange of FIN's must await completion of the resynchronization. However, a DSN and a FIN can "cross in the mail" and this leads to the mixed DSN/FIN states shown in figure 1. Note that even if a DSN has been received, a FIN can be sent if the receiver of the DSN has not also begun resynchronization of his own sequence numbers.



Resynchronization State Diagram

Figure 1

1.3 Window Control

In the TCP specification, no mertion is made of the specific way in which the receiver's window size might interact with his duplicate detection mechanism. In the SU-DSL implementation, the initial decision was to use the receive window to filter acceptable packets out of all those arriving on a given connection. This idea leads to some difficulties if the receiver ever shrinks the window in such a way that permission to send a particular range of sequence numbers is revoked, after the sender has transmitted packets in that range. For example, the receiver might set a window size of 100 octets, specifying that the sender is free to transmit sequence numbers 200 to 299 (remember, it is the combination of the ACK field and window field in a TCP packet which uniquely grants permission to send a certain range of sequence numbers). Suppose that the sender promptly emits a packet whose sequence number is 200 and whose length is 100, using all the sequence numbers it has been allowed. Suppose, further, that the receiver discovers that the user he is serving has inexplicably slowed down packet processing (or even stopped) so that too many packet buffers are now filled with packets awaiting service. The receiver might send an ACK back indicating that the new window size is now 50.

There are several problems here. If the sender stops all packetizing and all retransmission, the system may deadlock. One alternative is to continue to retransmit any packet at the head of the retransmission queue, even if it includes sequence numbers disallowed by the receiver. The receiver could just accept those it has offered to accept and discard the others. In this way, a packet might arrive piece-meal after several retransmission.

Alternatively, the packet might be repacketized into two separate packets so that the first part fits the current allowance. SU-DSL has implemented the former approach.

A second problem concerns the closing or interrupting of a connection which has had its window shrunk. If we absolutely forbid the transmission of packets outside the permitted range, or if the receiver ignores all packets outside of the permitted range, we may find it impossible to close or interrupt the connection, since the FIN and INT control bits are allocated 1 octet each in the sequence number space (to reliably distinguish any duplicate control packets arriving at the receiver). If we take the position that window sizes are suggestions only, as in [7], then it is important to decouple the duplicate detection mechanism from the flow control to some extent. A. McKenzie advocated this decoupling some time ago in technical protocol discussions and we now understand the merit of his position. We have chosen the maximum allowed window size (2^{16} octets) as the duplicate detection filter, independent of the receiver's current window size. A slightly more accurate choice would be the window between the next sequence number expected and the largest sequence number for which permission has been granted (i.e., if a window is sent from receiver to sender which grants permission to send packet sequence numbers larger than those currently allowed, the duplication detection filter would be updated). If the window shrinks, no change would be made to the filter. Even this idea is not without a possible glitch. If the sender uses up the maximum he is allowed and <u>then</u> needs to send INT or FIN, he may need to violate the flow control to do it and the receiver must be able to process these packets. Either the sender must avoid using up the last octet he is allowed or the receiver must berd a little to allow a few more octets through the sequence number filter. We have taken the latter approach: the receiver can accept packets beyond the maximum range he has granted, just for this purpose.

A different problem related to window control cropped up during the design of an SPP (Station to Packet Radio Protocol) proposal [8]. D. Retz adapted a simplified version of TCP for reliable packet transmission for packet radio use. In the Packet Radio Net, it is considered important for the Station to be able to unilateraly silence a repeater acting as a source for a host or a terminal, particularly if there appears to be congestion building up in the neighborhood of the repeater in question. Consequently a window size of zero was taken to mean "absolutely no more transmissions until otherwise notified." Notification was to come from the receiver through emission of an ACK packet with a new window when the station (as receiver) opened the window again. Under certain circumstances, however, it might be impossible to distinguish between the ACK which shut the window down and the one which opened it up (.e. same sequence number and ACK field, but different window size). Since packets can arrive out of order, the repeater might think the window was zero when, in fact, it was not. Several schemes were proposed to fix this problem. What is relevant to the TCP design is that a similar position (i.e. no retransmissions or new transmissions if the send window is zero) could lead to similar problems. One possible solution is for the receiver, in the absence of other traffic, to send ACK packets repeatedly with the new window size. The question is: "when should the receiver stop transmitting ACKs?" One answer is "when the sender finally sends something," which might be a long time away.

One alternative is for the TCP which changes its receive window from zero to non-zero to send a NOP packet containing the new window. The NOP takes up a sequence number and can therefore be ACKED, allowing the receiver to reliably tell the sender when the window was open, and also allowing the receiver to unambiguously identify the latest window information.

Let us analyze the idea of not sending anything (except an INT, FIN, DSN, or NOP) when the window size is zero. There are basically three cases to deal with in the case of simultaneously shut windows and two cases for the unilaterally shut case (the others are taken care of by symmetry).

<u>Currently shut windows</u>

Case A: Both sides have empty retransmission queues

When either or both sides re-open their windows, a NOP with the new window size is sent to the other side. Eventually the NOP(s) will be acknowledged, showing that the other side now knows about the non-zero window, and data can flow again.

Case B: One side has a non-empty retransmission queue and the other is empty.

It is immaterial what window size information is contained in the packets still in the retransmission queue.

If the destination has been told through the acknowledgment of one of its packets that the window is now zero, the arrival of any packet already on the retransmission queue before the window went to zero will not cause the receiver to believe that the send window is non-zero. Only the latest window information (based on sequence number and ack field of the arriving packet) should ever be believed. If the TCP with an empty transmission queue re-opens the window, it can send a NOP with the latest window information. Data will then flow in the other direction. If the TCP with the non-empty transmission queue wants to re-open the window, there may be a problem. If packets on the retransmission queue have their window fields changed to reflect the latest window size, their checksums must be recomputed. Furthermore, if the packet has arrived before, the new window data may not even be accepted by the receiver.

If a NOP packet is sent out, but placed at the end of the retransmission queue, it may not arrive and won't be retransmitted. If it is placed at the head of the queue and <u>is</u> retransmitted, data may eventually flow in the opposite direction, but the NOP can't be acknowledged until all the other packets on the queue have been sent (due to the cumulative acknowledgment strategy of the TCP). If <u>only</u> the head of the retransmission queue is retransmitted, a deadlock will result. If all packets are retransmitted, then the situation will eventually resolve itself. One rule might be to transmit all packets (when they timeout) if the window $\neq 0$, otherwise only the head of the queue (the NOP). Such a strategy may lead to a lot of unnecessary retransmission and some messy work to match acknowledgments with retransmission queue entries.

Case C: Both sides have non-empty retransmission queues

The analysis for this case is the same as for B, leading to the same conclusions.

Unilaterally shut window

Case E: Receive window is zero with empty retransmission queue

The receiver can send a NOP which unambiguously reopens the window.

Case F: Receive window is zero with non-empty retransmission queue

The receiver must somehow reliably inform the sender that the window has re-opened. So long as the outgoing window is not zero, sending and queueing a NOP with the new window size as in E will eventually work. If the other side closes the window, problems akin to cases (B) and (C) may arise.

Matters are simplified if the sender becomes responsible for finding out if the window is non-zero again. In this case, the sender always retransmits the head of the retransmission queue (albeit at a lower rate, perhaps, when the window is zero). If a sender wants to send a letter, but the window is zero and the retransmission queue is empty, a NOP packet can be sent and retransmitted until acknowledged. The acknowledgment will contain the current window size. By convention, NOP's should not be acknowledged if the window is still zero (since this would defeat their purpose).

Although it is not strictly necessary, a receiver who reopens the window can also send and queue a NOP with the new window size if he wishes to advise the sender, regardless of the sender's interest in transmitting anything just now.

For the present, we take the position that

a) Senders should always be retransmitting the head of the retransmission queue, even if the window is zero.

b) Senders should queue and send a NOP if they want to packetize new letters but the window is zero. Receivers should never ACK a NOP with a zero window.

c) Receivers re-opening the window from zero to non-zero may send and queue a NOP with the new window size.

Step (b) above is important. Even if the retransmission queue is non-empty, a NOP should be queued if the window is zero when a letter is waiting to be sent. If this is not done, it's possible for the retransmission queue to drain out leaving a non-empty letter queue and nothing to transmit to remind the receiver that the sender still wants to transmit. As described. the protocol may produce extra NOP's. Since the receiver will not ACK a NOP with a zero window, eventually the sender will be rewarded for his perserverance with a non-zero window.

1.4 Flush/Fin

The current TCP specification implies that the receipt of a packet containing a FIN should cause all undelivered data to be flushed and a FIN returned. We have already described (in our previous quarterly report) how the FIN's must be acknowledged so that the sender of the FIN can tell whether the connection closed normally or not [9] Further discussion among NDRE, UCL, BBN, and SU-DSL participants have resulted in a version 2 specification in which FIN does not cause flushing. A new control bit, called FLUSH, has been added which takes up one octet of sequence space. Octets whose sequence numbers precede the FLUSH bit are flushed if undelivered. The FLUSH is exactly like INT, but causes no signal to the user at the receiving end. In fact, we can separate the signalling function from the FLUSH function of INT. Each can be delivered reliably and separately. New user interfaces can be defined to permit each to be sent individually.

Such a redefinition will not rule out the implementation of the existing user interfaces by having the INTERRUPT user call generate INT and FLUSH; similarly, CLOSE could generate INT and FIN, if desired. TCP's should be prepared to receive these independently, however, so that a "graceful" close could be implemented (i.e. all packets preceding the FIN are delivered and acknowledged before sending an acknowledgment for the FIN). We plan to incorporate these changes in version 2 of the TCP specification. As an interim policy, we recommend that FIN/FLUSH and INT/FLUSH be sent together until all TCP implementations have been updated to version 2.

1.5 Special and Error Packet Handling

The original TCP specification was not very explicit about the way in which error packets were to be generated. We have not found any use for the special REJECT packet and will delete this from the specification. Y. Dalal wrote the draft below describing error handling procedures which we believe to be correct but which will be tested during the next quarter.

We recapitulate in this section the salient features of the mechanisms that exist in the TCP for recovering from loss of synchronization between two communicating processes. Detailed descriptions can be found in the TCP [1] and in INWG Protocol Note #14 [10]. We do not deal explicitly with the processing of error in the context of desynchronization, but believe that nothing contained herein conflicts with the proposed desynchronization/resynchronization method of section 1.2. We also believe that the mechanisms will work with either the flush type FIN (Version 1 TCP) or graceful FIN (Version 2 TCP).

Error handling and recovery consists in transmitting and receiving either ERROR or RESET packets. The ERROR packets that can be transmitted are

(I) EFP 6 - unacceptable SYN [SYN,ACK]

(II) EFP 7 - connection non-existent

(III) EFT 8 - foreign TCP inaccessible

All these packets have a sequence number that corresponds to the left send window edge of the associated connection, if there is one; otherwise 0 or any random number. None of the error packets occupies a sequence number.

In the ACK field of the packet the sequence number of the received packet that caused the error packet is present. In the case of the RESET packet, the sequence number of the error packet that caused its generation is in the ACK field. This is in order that these packets can be believed when they are received, and so duplicates can not foul up the connection. The ACK bit is always set in ERROR or RESET packets.

Errors 7 and 8 are very simple to understand and implement. First consider the arrival of either packet. If the state of the connection is in any of the synchronizing states then the state is changed to OPEN and the synchronization efforts resumed from scratch (should the need be, i.e. we may be a listening port). If this repeatedly happens, the user timeout will go off, and the user will be advised of this and can do whatever it wants to. If the state of the connection is in any of the FIN processing states then the TCP will simulate (to the user) that the connection was closed with a Flush FIN from the foreign TCP. If the connection is established then the user is given the error message. The message implies that something is amiss, and the user can carry on (if the message was EFT 8) or can CLOSE the connection.

These errors are generated as follows.

Error EFP 7 is generated at the foreign TCP when it is performing the address check on the packet received.

Error EFT 8 is generated by the Operating System at the Host (say).

Error EFP 6 and RESET are for the purpose of establishing connections reliably, and for being able to recover from crashes or loss in synchronization. Loss in synchronization may occur owing to long delays and lost packets. In short this mechanism recovers from the existence of half open connections.

A RESET is ONLY generated if the TCP is in SYNsent state, and gets Error EFP 6. The receipt of this error (when trying to open a connection) implies at the worst that the other end is in a half open, i.e. established, state. Hence it must be reset.

A RESET is only believable if it references an error packet that was just sent out. When a RESET that is believable is received and the connection is ESTD, it is once again in the OPEN state and the user is informed about it. All outstanding SEND and RECEIVE buffers are returned, and the input packet queue and retransmission queue flushed. The user may then issue a CLOSE if he wishes or do some more SENDs or whatever. This is the usual case. If the connection was in the synchronizing states then it is put into OPEN and the user is not told anything. If it was in the FIN states then the connection is closed and the user told that it was a Flush type close. If the state was in one of the desynchronization states, then the connection is put into the OPEN state and the user informed in a similar way as the case where the connection was in the ESTD state.

The generator of the RESET is put into the OPEN state and synchronization efforts will continue.

The following is a detailed analysis of how EFP 6 and RESET interact in the SU-DSL TCP.

An error EFP 6 can be generated in the following situations:

OPEN: A SYN with some other control bits like FIN,ACK,INT or DSN arrives. We haven't considered doing fancy things like SYN,INT in one packet yet.

SYNsent: A SYN with some other control like FIN,INT or DSN arrives, or there is a bad ACK (if at all) with the SYN.

SYNrcvd: A SYN that is different from the one which got us so far arrives. It may have random bits set too. Note that we do not go back to OPEN since that may be an old duplicate and all may be well now.

SIMULINIT: A SYN that is different from the one that got us so far arrives. Not only do we send an error EFP 6, but we ALSO set the state back to OPEN and try again.

ESTD: If any SYN different from the one that synchronized the connection arrives, then an error is sent. Note the implications of this on doing desynchronizations at random times has not been evaluated. It should not be harmful at all since the receiver of it can discard it as in many of the cases above.

FINwait: Error EFP 6 is never generated in this state since under all circumstances the connection will close (even by timeout).

FINrcvd: This state never generates an EFP 6 either for the same reasons as above.

The arrival of EFP 6 will have the following actions, if it is BELIEVABLE.

OPEN: No action, as it can't be believable!

SYNsent: Send a RESET packet and set the state back to OPEN.

SYNrcvd: Set the state back to OPEN.

SIMULINIT: Set the state back to OPEN.

ESTD: It can not be acceptable since we couldn't have sent a SYN!

FINwait: It can not be acceptable.

FINrcvd: It can not be acceptable.

The BELIEVABLE test: a few comments are necessary here. When testing to see if an ERROR packet is believable it may be sufficient to check if the ACK field refers to something we sent that has not been ACKed. To believe a RESET packet we also have to check that not only does the ACK refer to something we sent, but we also did in fact send an EFP 6. This additional precaution is necessary since the consequences of getting a RESET are grave. A malicious TCP may fabricate a RESET which ACKs something that the destination TCP has sent. If that TCP believes the RESET without making sure it really sent an EFP 6 then we may just have lost a connection. It is true that malicious TCP's could cause havoc, but it seems worth making redundant checks where appropriate. It does not seem that giving ERROR or RESET control a sequence number of their own will solve any of the problems we have.

With the addition of Desynchronization it should not be very difficult to add the extra processing for error or reset packet generation and handling.

In short the mechanism of dealing with half open connections is that they are reset (an attempt is made). Note that a RESET is never retransmitted (the sender of it goes back into OPEN too). If this RESET is lost then we still have an half open connection which will at some time get reset by another RESET type scenario or by an EFP 6 type message. We believe that all kinds of strange situations can be coped with, using this simple exchange, as at the most they will degenerate to half open connections.

1.6 Acknowledgment Generation

In general, TCP acknowledgments are innocuous and their liberal generation does not cause any particular malfunction. Nevertheless, too liberal generation of acknowledgments can cause a general reduction in overall bandwidth.

The cumulative, piggy-back nature of the TCP acknowledgments makes it feasible, and thus desirable to limit the transmission of useless acknowledgments. On the other hand, only delay in acknowledging a packet may lead to its unnecessary retransmission and this could be more costly in lost bandwidth than sending an empty acknowledgment packet.

In our experiments we have tried several strategies. In one version, we only sent an acknowledgment when the retransmission process woke up and found a "NEWRECEIVE" flag set, indicating that one or more unacknowledged packets had arrived.

The result of this strategy was that for connections having traffic flowing only in one direction, the bandwidth was directly proportional to the frequency of running the retransmission routine, and unnecessary retransmissions were high in number.

A second strategy was to force an acknowledgment to be sent immediately upon receipt of a packet (assuming it was in sequence). This reduced retransmissions, but had a high cost in wasted bandwidth for unnecessary acknowledgments.

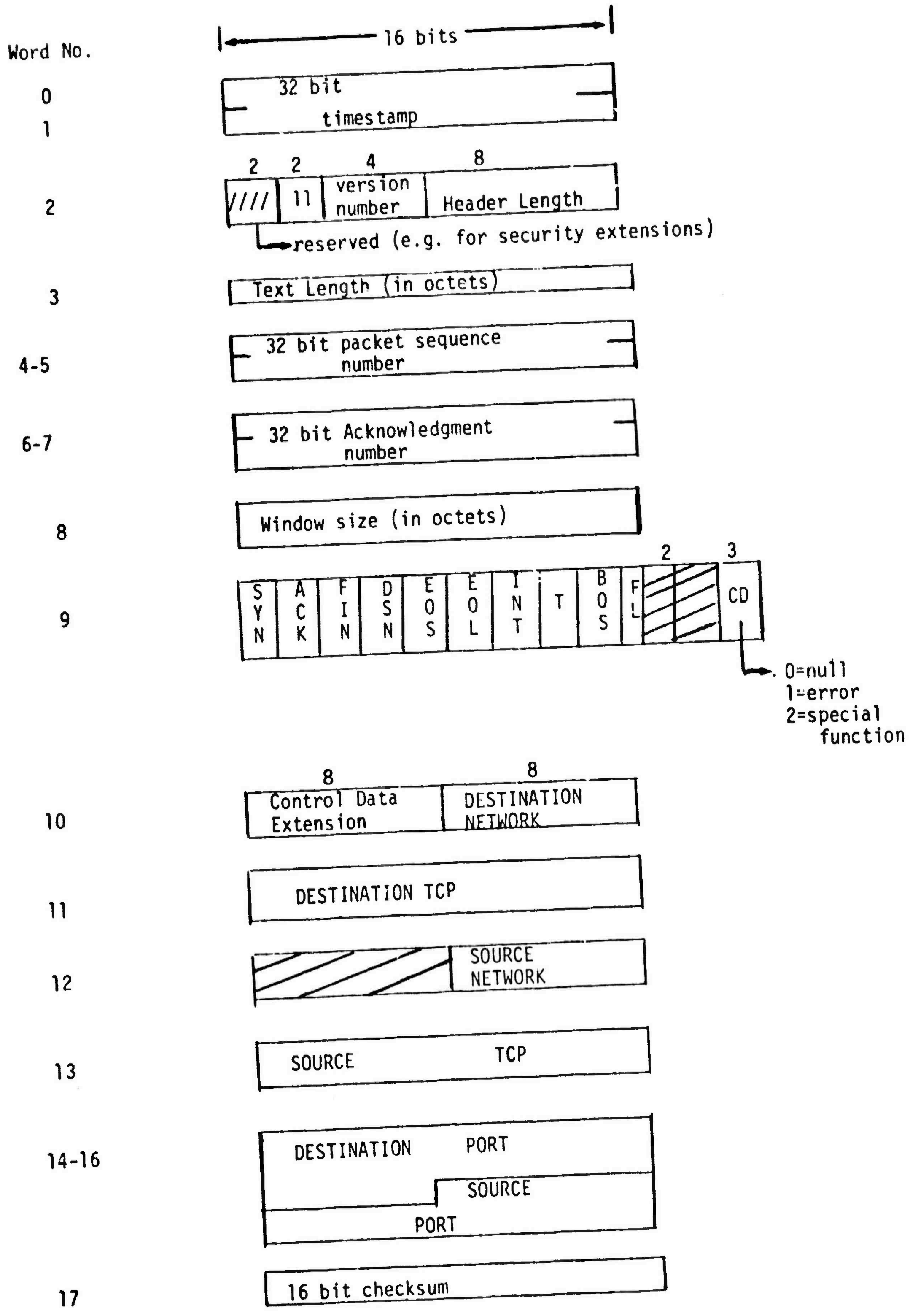
A third scheme sent an acknowledgment packet only if the outgoing letter queue for traffic in the reverse direction was empty. If there is a lot of back traffic in the reverse direction, we expected this strategy would minimize empty acknowledgment traffic since acknowledgments would be piggy-backed on the return traffic.

This last case seemed optimal, but still seems to have some bugs, since running in this mode still appears to generate a substantial number of unnecessary acknowledgments. One likely explanation for this is that, if queues form in the experiments we have conducted, they form other than in the outgoing letter queue and consequently a large number of unnecessary acknowledgments are sent because the letter queue is usually empty. In the case of the "echoer" program, its letter queue is almost always empty. When an incoming packet passes through the reassembler, a check is made to see if an ACK should be sent. Since the letter queue is empty, an ACK is sent and shortly thereafter, a packet is sent on the reverse channel, carrying exactly the same acknowledgment information. We consider this still an open problem.

1.7 New TCP Header Format

The TCP header format remains almost the same as specified originally, but with a few additions to the control word and an extra 32 bit timestamp field. The latest format is shown in figure 2 and explanations for the changes follow.

The 32 bit timestamp has been added primarily for debugging and performance measurement. It is currently used mostly by BBN, but some co-operative experiments are planned which could make use of this facility. The timestamp is _not_ checksummed.



TCP HEADER
Figure 2

A "T" (Timestamp) control bit has been added to indicate the presence in the text of the packet of an index word [16 bits, points to last timestamp] and a series of timestamps which have been accumulated as the packet winds its way through TCP's, Gateways, Echoers, and so on. A description of the timestamping conventions is to be published in a separate technical note and will also be incorporated in the rewrite of the TCP specification.

A "BOS" [beginning of Segment] bit has been added to simplify the reassembly and checksumming of several fragmented packets concurrently.

The "FL" [flush] bit has been added to allow for a graceful FIN and a non-interrupting flush.

The "NOP" bit has been added to allow for reliable re-opening of windows set to zero.

For fragmentation and general processing purposes, sequence numbers are associated with some of the bits as shown in Figure 3. Thus, the SYN, if present

Sequence No's occupied by packet

SYN	NOP	INT	FL	text	DSN	FIN

Logical Placement of Control Bits
Figure 3

is associated with the packet's sequence number. If not present, but NOP is present, then the NOP takes the sequence number. Similarly, if present DSN is associated with the last or next to last packet octet (FIN if present, is last).

As before, the checksum word is only meaningful if the "EOS" (end of segment) flag is set.

The source and destination networks are now just 8 bit fields (before they were pairs of 4 bit fields).

1.8 Miscellaneous Updates

When the TCP was first specified, it was thought useful to include a set of special functions that could be performed independent of any connection (e.g. RESET ALL connections, deliver packet to TRASHCAN, request ECHO, send ECHOREPLY, QUERY connection status, send STATUS, and RESET specific connection). The only ones which still appear useful are RESET specific and possibly the QUERY STATUS.

Consequently, the notion of the "well-known TCP socket" can be eliminated, since no packets need ever refer to it. As it happens, another well-known socket has been proposed for remotely controlling the internal parameters of the TCP. This is the parameter-change socket. The parameter-change package will be described in more detail in a forthcoming DSL technical note. It is rather like the FAKE HOSTs inside IMPs which allow the TCP state to be tweaked by a remote experimenter.

2.0 Packet Radio Network

2.1 TCPØ Implementation [J. E. Mathis]

During this quarter, the single channel TCP (TCPØ) was designed and a preliminary design specification released [11]. While the TCPØ is designed to run as a user and/or server, and is compatible with standard full-size TCP's, some simplifications have been possible which greatly reduce its size and complexity. Some changes have been made in the design during implementation, partly to accommodate the updated TCP specifications described in section 1.

TCPØ uses a single ring buffer implementation as suggested in [7] which substantially simplifies memory allocation and window control. We plan to test TCPØ on our PDP-11/20 during the month of March and then move it as soon as possible to SRI's LSI-11 for test in the PRNET. We plan to interface TCPØ to a TELNET on one side and both CAP and SPP on the other so as to test the delay, throughput, and reliability properties of TCP running with and without SPP. We are hoping that SRI will provide an implementation of SPP [13] for our use on the LSI-11.

As for size, the TCPØ by itself appears to require about 1.5K (16 bit) words. CAP and SPP may occupy another 1K and TELNET another 1.5-2K, so we are still expecting to have the entire LSI-11 terminal software package resident in less than 4K. However, as was mentioned in our last quarterly report, we've ordered 8K of RAM just in case.

2.2 Mini-TELNET [D. Rubin]

Implementation began on our TELNET to interface to TCP. We are hoping to test TCPØ and TELNET together on the PDP-11/20 and later the LSI-11 at SRI during the month of March.

A preliminary implementation specification was issued by D. Rubin on 2 February [1] and will be updated as needed.

One major issue still to be faced is what the TELNET will connect to. So far as we know, BBN does not have a TELNET/TCP combination in operation (only a fake TELNET). We can do a certain amount of local testing, but a good demonstration of useful internetting will need a TENEX-based TELNET/TCP.

We do not anticipate any problem interfacing TELNET with either TCPØ or TCP.

2.3 BBN 1822 Spec Interface for LSI-11 [R. Crane]

A double-height interface card was designed and implemented during this quarter. It has entered testing during this writing (early March) and is compatible with both the DR-11 on our PDP-11/20 and with the DRV-11 on the LSI-11/03 at SRI.

Our own LSI-11 is still on order and we are still unsure of its delivery date. Once testing is complete, we will publish the design of this device for the benefit of other groups at SRI and Collins Radio Corporation, which may have an interest in such a device.

2.4 PR Unit

All the cabling required to connect the IMP-11A interface on our PDP-11/20 to a PR unit to be installed on the Durand Building roof has been pulled, terminated and tested. There should be no problem driving the PDP-11 to PRU cable at 50-100 Kilobits/sec.

As of this writing, no radio tests had been performed to ascertain whether the PRU could be successfully used from the Durand roof. We are expecting these tests momentarily, but this is up to SRI.

3. Gateway Plans

We have made very little progress in our attempts to specify experiments using internet gateways. Neither do we have in hand a definitive document describing all the tasks of a gateway and its functions. The stabilization of the various TCP implementations should introduce some new pressures for these specifications, as will the planned PRNET/Internet demonstrations next quarter.

Aside from the obvious functions that permit a gateway to both <u>connect</u> nets together and at the same time <u>isolate</u> the internal functions of the networks from each other, there are a morass of fuzzy ideas still to be worked out.

We continue to believe that gateways should be simple and minimal, but must confess that some may, perforce, turn out to be complicated.

The model presented by J. Burchfiel [14] at the 14 February 1975 Gateway meeting at ARPA/IPTO is very appealing. He divides the Gateway into 3 major parts:

(a) Network Independent Functions

(b) Local Network Dependent Functions

(c) Fake Host functions (reached via raw internet, non-fragmented packets)

<u>Network Independent Functions</u>

1. Routing
2. Access Control
3. Accounting and Statistics Collection
4. Fragmentation
5. Congestion Control
6. Retransmission (?)
7. Checks and Balances

<u>Local Network Dependent Functions</u>

1. local header generation/removal
2. Local network reliability enhancement and flow control

<u>Fake Host Functions</u>

1. Debugging
2. Packet echoing
3. Traffic generation
4. ....

We have been of the past opinion that the Gateway should remember nothing about the packets it sees. Except for local network retransmission, flow control, and duplicate filtering (e.g. for PRNET, since this isn't done in the subnet), the Gateway might never remember anything about a packet. On the other hand, efficiency and lower delay might be achieved through retransmission and positive acknowledgment between Gateways. The tricky business here is that transmission between the Gateway and the destination TCP or between source TCP and Gateway might require another (albeit thin) layer of protocol between the TCP and Gateway to make things look uniform to the Gateways. We're not sure it's worth it, but will be thinking about it.

Another issue has to do with error message propagation between networks. This issue is also relevant for the TCP design, first because local network error messages must propagate in internet packets and second because the TCP must accept local network error messages and give them to the user in some standard vanilla form. A basic problem is whether a subnet error message will contain enough information so that, even if delivered to the correct source Gateway, can be propagated to the correct source TCP! Without remembering something about the packet, the Gateway may be unable to associate an error message with a particular packet and TCP source. This, in itself, isn't a disaster, but from the standpoint of isolating failures, we may find it useful to retain internet packet status information until

a) local net says it has delivered it

or b) the next Gateway has acked it

or c) the destination TCP (in the attached local net) has acked it.

We have yet to identify a common set of local net errors which should be propagated across the Gateway.

4. TCP Testing

4.1 Local Throughput Tests

A number of different variables can have an effect on the throughput performance of the TCP. The priority of the TCP modules in relation to user code, ELF system software and the RTP turned out to have only a minor effect on throughput. In early self-loop tests, we discovered that the generation of TCP acknowledgment packets had a profound effect on throughput (not surprisingly) as did the coupling of the retransmission routine to ACK generation. So did the number of letters contained in the "pipeline."

In an early (November 17, 1975) throughput experiment, we used our primitive exerciser, slightly modified, to self loop a TCP connection (i.e. its output went to its input). Table VI illustrates the main results. Note that these results do not reflect later improvements both in RTP programming and TCP parameter setting. There are five groups of results, the first group shows what happens if TCP acknowledgments are sent only when the retransmission process runs (analogous to a full duplex channel on which only one side is transmitting). The second group shows forced acknowledgments on receipt. In both cases, it's clear that multiple TCP packets are being acknowledged at one time. The biggest improvements, however, show up when tne retransmission timeout is 4 seconds, allowing the pipeline to carry acknowledgments in next packets. In the last 3 groups, acks were forced on receipt of a new packet, but retransmissions were reduced to zero since the retransmission routine never found any unacknowledged 4-second-old packets. Running the user at priorities 3 and 4 had little effect on throughput. The best bandwidth was roughly 3 letters/second.

17 November 1975 Throughput Tests

All Measurements of 1 Minute Duration

	No. letters on Send Queue	Letter Length In Chars.	Retrans Timeout in Sec.	Wakeup Timeout in Sec.	No. Letters Sent	No. Acks Sent by Retrans	No. Retrans Sent by Retrans
No Forced Acks	2	10	.5	.5	176	74	72
	2	10	.5	.25	171	78	72
	3	10	.5	.5	119	80	172
	3	10	.5	.25	139	81	148
Forced Acks	2	10	.5	.5	108	37	113
	2	10	.5	.25	107	34	114
	3	10	.5	.5	81	55	187
	3	10	.5	.25	76	52	182
Priority 3	3	10	4	4	178	0	0
	3	70	4	4	170	0	0
Priority 4	3	10	4	4	173	0	0
	3	70	4	4	164	0	0
Priority 4	5	10	4	4	170	0	1
	5	70	4	4	27*		

* Blew up after running out of message storage

Early Bandwidth Tests

TABLE VI

A similar experiment was run December 29, 1975, and found that forced acknowledgment with a 4 second retransmission timeout yielded 2.9 letter/sec for 10-50 character letters. The results are summarized in Table VII.

It became evident that forcing an acknowledgment when not really necessary would use up valuable bandwidth. To test this hypothesis, we ran an experiment on December 30, 1975, to eliminate forced acknowledgments, but to send 3 packets into the pipeline. This way, each succeeding packet would tend to acknowledge the previously received one (again self looping a TCP connection through the IMP). Table VIII shows the results.

Earlier timestamp experiments indicated that the TCP was spending a substantial amount of time in its "pick send connection" and "pick receive connection" procedures. R. Karp re-wrote these and ran a throughput experiment on December 31, 1975, the results of which are shown in Table IX.

letter length retrans.timeout sends in pipe Duration	10 char 0.5 sec 1	10 char 4 sec 1	10 char 4 sec 3	50 char 0.5 sec 1	50 char 4 sec 1	50 char 4 sec 3
30 sec.	77(2.6)	79(2.6)	93(3.1)	74(2.5)	--	87(2.9)
60 sec.	151(2.5)	155(2.6)	188(3.1)	148(2.5)	152(2.5)	180(3.0)
90 sec.	226(2.5)	234(2.6)	273(3.0)	227(2.5)	231(2.6)	265(2.9)
120 sec.	301(2.5)	314(2.6)	359(3.0)	297(2.5)	307(2.6)	349(2.9)
150 sec.	371(2.5)	397(2.6)	447(3.0)	372(2.5)	381(2.5)	431(2.9)
180 sec.	449(2.5)	474(2.6)	533(3.0)	443(2.5)	460(2.5)	519(2.9)

Numbers in () are letters/sec.

12/29/75 Throughput Experiment
(forced acks)

TABLE VII

Letter length	Retrans. Timeout	Duration	Sends in Pipe	Letters sent	Letters/sec
10 chars	4 sec.	150 sec.	3	720	4.8
50 chars	4 sec.	150 sec.	3	640	4.7

12/30/76 No Force Ack Throughput Experiment

TABLE VIII

Letter Length / Sends in Pipe / Retrans. TIMEOUT — duration	10 chars / 3 / 4	50 chars / 3 / 4
30 sec	197 (6.6)	-----
60 sec	386 (6.4)	340 (5.7)
90 sec	575 (6.4)	513 (5.7)
120 sec	766 (6.4)	688 (5.7)
150 sec	961 (6.4)	859 (5.7)
180 sec	1150 (6.4)	1038 (5.8)

Note: numbers in () are letters/sec

12/31/75 Throughput Tests

TABLE IX

4.2 Local Test Recommendations

We believe that substantial and uniform local testing should be performed at each TCP implementation and that the results be collected together in a single report for the benefit of all experimenters. We have produced a tentative list of these local tests below. Some implementations such as TCPØ may only be able to perform a subset.

a) Bandwidth tests

1. Internal cross-patch (to avoid network entirely) of a single TCP connection (to itself). Run 1 byte, 90 byte, and 972 byte letters.
2. Same as above, but loop the network interface.
3. Same as above, but loop through the local packet switch (IMP,PRU,SIMP).

4 - 6. Same as 1-3, but run two connections simultaneously (e.g. echoer to exerciser). (This can't be done by TCPØ).

These test reports should be well documented as to window size, retransmission timeouts, number of letters in the pipeline, buffer utilization, and so on.

b) Reliability

1. Transmit for 2 hours, through network, to a sink (or echo package).
2. Cycl through resynchronization logic several times (might be done by setting resynchronization detection logic to relatively short time like 5 minutes).
3. Verify OPEN/CLOSE will work repeatedly (e.g. a few hundred times). Might also collect average delay statistics for this function.

c) Sizes

The sizes of various parts of the TCP should be reported; buffer pool size; other relevant program sizes (e.g. echoer, exerciser, parameter change program,...).

4.3 UCL/SU-DSL Experiments

By mid-February, UCL's implementation was sufficiently complete to begin some tests with SU-DSL. As of this writing, experiments were scheduled twice a week on Tuesday and Thursday and last about four hours each (0800 - 1200 SU-DSL time and 1600-2000 UCL time). Many of the experiments have been frustrating, owing to a bug of some kind in UCL's buffer allocation scheme causing them to crash irrevocably when attempting to achieve high bandwidth.

Initially, however, we can say that the simple terminal to terminal kind of communication is stable and reliable. It also appears that a connection running up to 1 letters/second can be sustained indefinitely by UCL. The round-trip delays are horrendous (2-8 seconds!) and we are planning some timestamp experiments to track them down.

4.4 Timestamping and Parameter Change

During the quarter, the design of timestamping and parameter change mechanisms were frozen and implemented at UCL and SU-DSL. These mechanisms should aid in the experiments we plan for future months, especially to allow remote control of TCP parameters via parameter change. We will produce, as technical notes, the specification of both mechanisms, during the next quarter.

5. Security Systems

We are continuing our work in this area and plan a meeting 19-20 February with R. Tomlinson and A. D. Owen of BBN and T. Chandler of CRC to specify in more detail the software functions of the red and black side crypto controllers.

REFERENCES

1. V. Cerf, Y. Dalal, C. Sunshine, "Specification of Internet Transmission Control Program," INWG General Note #72, December 1974 (revised).
2. "Specification for the Interconnection of a Host and an IMP," BBN Report No. 1822, Appendix F (VDH).
3. V. Cerf, "TCP Resynchronization," DSL Technical Note #79, 11 January 1976.
4. R. Tomlinson, "Selecting Sequence Numbers," INWG Protocol Note #2, September 1974.
5. Y. Dalal, "More on Selecting Sequence Numbers," INWG Protocol Note #4, October 1974.
6. D. Belsnes, "On Single Message Communication," INWG Protocol Note #3, September 1974.
7. V. Cerf and R. Kahn, "A Protocol for Packet Network Intercommunication," IEEE Trans. Comm., Vol C-20, No. 5, May 1974, p. 637-648.
8. D. Retz and V. Cerf, "Station-Packet Radio Protocol," PRTN #160, December 19, 1975.
9. V. Cerf, "ARPA Internet Protocols - Project Status Report," Digital Systems Laboratory Technical Note #68, Stanford University, November 15, 1975.
10. Y. Dalal, "Establishing a Connection," INWG Protocol Note #14, March 1975.
11. J. Mathis, "Single-Connection TCP Specification", Digital Systems Laboratory Technical Note #75, Stanford University, January 25, 1976.

12. D. Rubin, "TELNET under Single-Connection TCP Specification," Digital Systems Laboratory Technical Note #76, Stanford University February 2, 1976.

13. M. Beeler, "Will the Real SPP Please Stand UP?," PRTN #165, February, 1976.

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. Mark Rochkind
Bell Laboratories
600 Mountain Avenue
Murray Hill, New Jersey 07974

Mr. Kenneth L. Thompson
1103 Hig Court
Berkeley, CA 94708

BOLT BERANEK AND NEWMAN INC.
50 Moulton Street
Cambridge, MA 02138

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

B N R, Inc.
3174 Porter Drive
Palo Alto, CA 94304

Mr. Alex Curran, President
Mr. Alan Chapman
Mr. Barry Gordon

BURROUGHS CORPORATION

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

Mr. John Mazola
Burroughs Corporation
25725 Jeronimo Road
Mission Viejo, CA 92675

Mr. Louis de Bartelo
Burroughs Corporation
1671 Reynolds
Irvine, CA 92714

CABLEDATA ASSOCIATES

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

CALIFORNIA, UNIVERSITY - IRVINE

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

CALIFORNIA, UNIVERSITY - LOS ANGELES

Professor Gerald Estrin
Computer Science Department
3732 Boelter Hall
Los Angeles, CA 90024

Professor Leonard Kleinrock
Computer Science Department
3732 Boelter Hall
Los Angeles, CA 90024

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

A

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 22090

GENERAL ELECTRIC COMPANY

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

Mr. J. T. Duane, Mgr.
Special Purpose Computing Center
1285 Boston Avenue
Bridgeport, Connecticut 06602

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

GENERAL MOTORS CORPORATION
Computer Science Department
General Motors Technical Center
Warren, Michigan 48090

Dr. George C. Dodd, Assistant Head
Dr. Joseph T. Olsztyn
Dr. James Thomas

HAWAII, UNIVERSITY OF

Professor Norman Abramson
The ALOHA System
2540 Dole Street, Holmes 486
Honolulu, Hawaii 96822

HUGHES AIRCRAFT COMPANY

Mr. R. Eugene Allen
Bldg. 604, M.S. D-222
P. O. Box 3310
Fullerton, CA 92634

Mr. Thomas J. Burns
Bldg. 390, M.S. 2007
P. O. Box 92919
Los Angeles, CA 90009

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

HEWLETT-PACKARD

Mr. Don Senzig
H-P Laboratories
Building 18
1501 Page Mill Road
Palo Alto, CA 94304

Dr. J. R. Duley
HPL/ERL
3500 Deer Creek Road
Palo Alto, CA 94304

Mr. Stephen Walther
HPL/ERL
3500 Deer Creek Road
Palo Alto, CA 94304

IBM

Dr. Leonard Y. Liu, Manager
Computer Science
K51-282, 5600 Cottle Road
San Jose, CA 95193

Mr. Harry Reinstein
1501 California Avenue
Palo Alto, CA 94303

Dr. Donald Frazer
IBM Watson Research Center
P. O. Box 218
Yorktown Heights, New York 10598

ILLINOIS, UNIVERSITY OF

Mr. John D. Day
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 SOURTHERN CALIFORNIA
4676 Admiralty Way
Marina Del Rey, CA 90291

Mr. Steven D. Crocker
Mr. Keith Uncapher

LONDON, UNIVERSITY COLLEGE

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

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

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

MICROTECHNOLOGY CORPORATION
224 N. Wolfe Road
Sunnyvale, CA 94086

Mr. Fred Buelow
Mr. Naoya Ukai
Mr. John J. Zasio

MITRE CORPORATION

Mr. Michael A. Padlipsky
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. Howard Frank

NATIONAL BUREAU OF STANDARDS

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

Mr. Ira W. Cotton
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. C. 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 (4 copies)
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

3180 Porter Drive
Palo Alto, CA 94303

Dr. Jerome Elkind, Manager
Computer Science Laboratory

Mr. Robert Taylor, Principal Scientist
Computer Science Laboratory

Dr. Butler Lampson

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 - 5 copies
Engineering Library - 2 copies
IEEE Computer Society Repository - 2 copies

Electrical Engineering

Dr. John Linvill