Bolt Beranek and Newman Inc.

LEVEL II

AD A093164

Report No. 4526

Combined Quarterly Technical Report No. 19

SATNET Development and Operation
Pluribus Satellite IMP Development
Remote Site Maintenance
Internet Development
Mobile Access Terminal Network
TCP for the HP3000
TCP-TAC
TCP for VAX-UNIX

November 1980

Prepared for:
Defense Advanced Research Projects Agency

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 4526	2. GOVT ACCESSION NO. AD-A093	3. RECIPIENT'S CATALOG NUMBER 764
4. TITLE (and Subtitle) COMBINED QUARTERLY TECHNICAL REPORT No. 19		5. TYPE OF REPORT & PERIOD COVERED 8/1/80 to 10/31/80
		6. PERFORMING ORG. REPORT NUMBER 4526
7. AUTHOR(s) R. D. Bressler		8. CONTRACT OR GRANT NUMBER(s) MDA903-76-C-0252 MDA903-80-C-0353 & 0214 N00039-78-C-0405 N00039-79-C-0386 N00039-80-C-0664
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, MA 02238		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order Nos. 3214 and 3175.17
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE November 1980 13. NUMBER OF PAGES 130
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) DSSW Rm. 1D, The Pentagon Washington, DC 20310 NAVALEX Washington, DC 20360		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE/DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer networks, packets, packet broadcast, satellite communication gateways, Transmission Control Program, UNIX, Pluribus Satellite IMP, Remote Site Module, Remote Site Maintenance, shipboard communications, Terminal Access Controller, VAX.

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

This Quarterly Technical Report describes work on the development of and experimentation with packet broadcast by satellite; on development of Pluribus Satellite IMPs; on a study of the technology of Remote Site Maintenance; on the development of Inter-network monitoring; on shipboard satellite communications; and on the development of Transmission control protocols for the HP3000, TAC, and VAX-UNIX.

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(9) COMBINED QUARTERLY TECHNICAL REPORT NO. 19, 1 Aug-31 Oct 80

(6) SATNET DEVELOPMENT AND OPERATION
PLURIBUS SATELLITE IMP DEVELOPMENT
REMOTE SITE MAINTENANCE
INTERNET DEVELOPMENT
MOBILE ACCESS TERMINAL NETWORK
TCP FOR THE HP3000
TCP-TAC
TCP FOR VAX-UNIX

(12) 151

(10) F.D./Bressler

(11) November 1980

(14) BBN-4526

This research was supported by the Defense Advanced Research Projects Agency under the following contracts:

(15) MDA903-76-C-0252, ARPA Order No. 3214
N00039-78-C-0405, ARPA Order No. 3175.17
~~N00039-79-C-0386~~
MDA903-80-C-0353, ARPA Order No. 3214
MDA903-80-C-0214, ARPA Order No. 3214
N00039-80-C-0664

Submitted to:

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

Attention: Program Management

060100

Table of Contents

FIGURES

1 INTRODUCTION

This Quarterly Technical Report is the current edition in a series of reports which describe the work being performed at BBN in fulfillment of several ARPA work statements. This QTR covers work on several ARPA-sponsored projects including (1) development and operation of the SATNET satellite network; (2) development of the Pluribus Satellite IMP; (3) Remote Site Maintenance activities; (4) inter-network monitoring; (5) development of the Mobile Access Terminal Network; (6) TCP for the HP3000; (7) TCP-TAC; and (8) TCP for the VAX-UNIX. This work is described in this single Quarterly Technical Report with the permission of the Defense Advanced Research Projects Agency. Some of this work is a continuation of efforts previously reported on under contracts DAHC15-69-C-0179, F08606-73-C-0027, F08606-75-C-0032, and MDA903-76-C-0213.

2 SATNET DEVELOPMENT AND OPERATION

2.1 Minimal Host-SATNET Protocol for Loader/Dumpers

The search for a mechanism for loading the UCL gateway, once the ARPANET trunking line via SATNET to the London TIP has been removed from service, has led to renewed interest in a gateway loading access path via SATNET directly. This requires a loader/dumper be written for the gateway which implements XNET4, Internet Protocol, Host-SATNET Protocol, and ARPANET VDH Protocol. Clearly, given the number of functions involved, it is essential that the implementation contain only minimal subsets of all protocols involved. Towards that goal we have extracted the minimal subset of the Host-SATNET Protocol necessary for communicating with a device which is assumed to have implemented the full the Host-SATNET Protocol. This subset, which has been implemented in the stand-alone Honeywell 316 VDH loader/dumper currently in use for loading the Etam and Tanum SATNET Satellite IMPs, is detailed below.

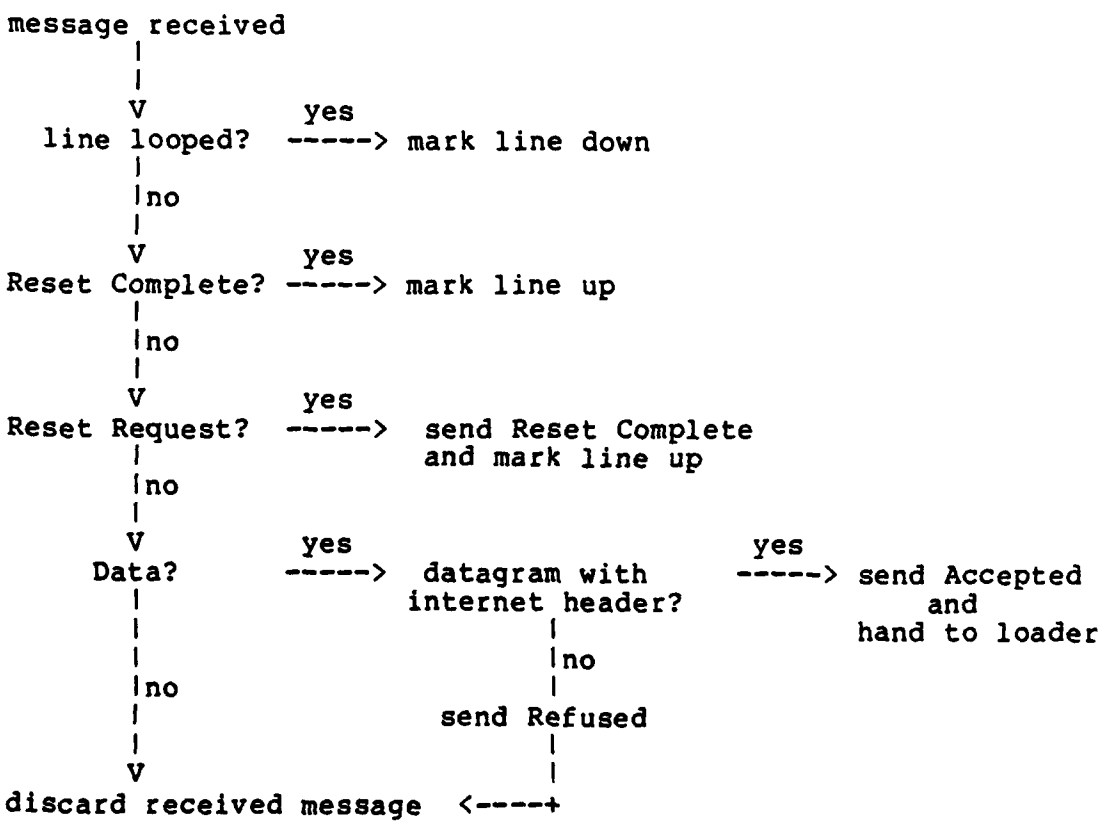
A loader/dumper implementing a minimal subset of the Host-SATNET Protocol need not respond to the entire list of message types defined in the protocol. In particular, of the 10 message types available in the protocol (shown in the following table)

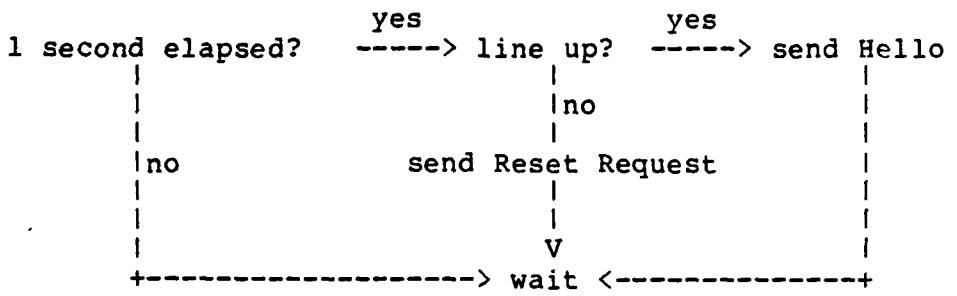
MESSAGE TYPE	FUNCTION
1	Data
2	Accepted
3	Refused
4	Status Request
5	Status
6	Hello
7	Data with Errors
13	Format Error
14	Restart Request
15	Restart Complete

only message types 1, 14, and 15 need be recognized among the received messages; all other received message types are discarded, including Hellos, status reports, and error reports. Furthermore, of all the packets encompassed by the Data message type (type 1), only datagrams with internet headers are accepted; stream data packets and packets without internet headers are discarded by the program, although a Refused message response is sent to indicate that a packet is discarded. In response to a Reset Complete, the line is marked up. In response to a Reset Request, a Reset Complete is sent, and the line is marked up. If the line is detected to be looped, the line is marked down. In background, the line is checked once every second; if up, then a Hello is sent; if down, then a Reset Request is sent.

The above is essentially a complete description of the minimal Host-SATNET Protocol. Clearly this subset violates basic elements inserted into the protocol for flexibility and efficiency, but this is irrelevant. All totaled, the SATNET VDH

loader/dumper requires fewer than 200 lines of code in Honeywell 316 assembly language to implement the Host-SATNET Protocol, an indication of its simplicity. The following two flow charts depict this implementation, where the first indicates action done upon receipt of every message and the second indicates action done every second to ensure that the line-up procedures are maintained.





2.2 Software Problems Fixed

A change was made to the software such that whenever the Satellite IMP is out of frame synchronization, the indicators of Hello packets received are reset for interpretation as no Hello packets whatsoever being received. Thus, the composite Satellite IMP Hello reception quality indicator, which is displayed in the B-register when sense switch 1 is reset, assumes the value zero. Furthermore, Hello packet MONITOR reports display a row of all 16's, meaning a failure to hear all Hello packets. This change reduces confusion between particular kinds of problematic Satellite IMP operation and correct operation.

The transition late last spring of the NORSAR-SDAC ARPANET circuit from a commercial satellite circuit to a military controlled satellite circuit has been accompanied by a severe deterioration in service and availability to users of the NORSAR TIP. In addition, one particularly vexing problem is that whenever the circuit is down, load splitting among the gateways

results in half the internet packets originating from the far side of the UCL gateway being discarded in the NDRE gateway. Although the preferable solution to this problem entails that the gateway routing algorithm make intelligent decisions based on information available regarding fractured networks, the choice was made during the October 1980 internet meeting for the expedient solution of having the communication path between the NDRE gateway and the UCL gateway permanently broken.

Originally, we undefined the NDRE gateway access path in the Tanum Satellite IMP, inasmuch as this action could be implemented very quickly by changing a parameter. The disadvantage of this approach, however, is that the NDRE gateway access path is removed completely from service. Any failure in the 50 Kb/s circuit between the NDRE gateway and the Tanum Satellite IMP during the time the circuit is undefined remains unknown to the Monitoring and Control Center until the circuit is needed due to a malfunction in the BBN Gateway or the Etam Satellite IMP.

Subsequently, we created a patch in the Host Access Protocol at Tanum which recognizes and discards any traffic specifically destined for Host 1 at Goonhilly, the UCL gateway. This effectively breaks the pathway from the NDRE gateway to the UCL gateway; the reverse path, however, is still operational. Nevertheless, this is sufficient to prevent any internet data traffic between the two gateways, since the gateway routing

algorithm requires two-way communication. A monitoring and control path from and into SATNET through the NDRE gateway continues to function with this scheme, allowing us both to monitor the circuit and to make use of the circuit for SATNET specific operations. Upon request, special accommodations will be made for people who require that the gateway-to-gateway operation be temporarily restored.

We modified the Satellite IMP so that upon initialization of the satellite channel interface, a sequence of table-driven commands will be issued to the CMM module of the PSP terminal. Inserted commands at this time include one to reset the interface, three to enable the transfer of T&M data from the PSP terminal to the Satellite IMP, and two to eliminate the carrier-sync bits in the modem preamble. (Carrier-sync bits are no longer required following the removal of SPADE modems from SATNET usage.) Thus, whenever the primary modem in the PSP terminal is removed from service, we can conveniently program the Satellite IMP to switch to the backup modem simply by adding the appropriate command to the table.

We changed the TENEX programs RECORDER and EXPAK to use only one special TENEX queue each for both sending and receiving traffic; originally, a sending queue and a receiving queue were allocated by each program. Since TENEX assigns up to a total of eight queue numbers, program operation is subject to queue

availability. Only recently with the operation of several versions of RECORDER and EXPAK for the Wideband Satellite Network, for MATNET, and for SATNET had the necessity for this change been made apparent, as queues became unavailable. The Satellite IMP program was modified as well to recognize single queues for transmit and receive.

2.3 Hardware Problems Fixed

Below are summarized several hardware problems which were manifested in the operation of SATNET during the last quarter. In cases involving the Honeywell 316, we not only diagnosed the problems, but also corrected them. In the other cases, we were involved primarily in the detection of the problems and helped with diagnosis.

A severe power glitch at Tanum crippled the Honeywell 316 Satellite IMP, such that the machine would not remain in a run state. Site personnel under direction from BBN service personnel swapped voltage regulator boards among the drawers in the Honeywell 316 and were able to neutralize the problem. Further examination revealed that the power-down circuitry on the voltage regulator card in the CPU drawer failed. Inasmuch as this circuitry is not used in the other drawers, board swapping succeeded in fixing the problem.

Substantial performance problems with SATNET during the last five months emanated from a 1 dB reduction in transmit power levels on June 1. Over the lifespan of SATNET, Intelsat has reduced transmit power levels in the net by 4 dB, to the extent that SATNET is without margin. Consequently, net performance has severely deteriorated, as is very much evident in the MONITOR statistics; even after power adjustments are made at all sites, Hello packets are routinely missed at the rate of a few percent. Comsat is currently examining procedures to have the power increased.

Transmit power adjustments, which normally should be routine and require under an hour, have on occasions consumed considerable time, causing lengthy disruptions to SATNET activities and London TIP connectivity. In one such example, a spurious, unidentified interference source corrupted the measurement of the transmit power levels and thereby reduced the accuracy of the adjustments.

On many occasions, the PSP terminals at Etam and at Goonhilly entered into undefined states, with the result that the attached Satellite IMPs were unable to receive satellite channel information. Correction required site personnel to press manual resets on the PSP terminals. On one occasion at Goonhilly, manual resets failed, and we reconnected the Satellite IMP to the SPADE modem in order to restore site operation and ARPANET

connectivity to the London TIP. Comsat discovered that a power-off/power-on sequence was needed to reset the PSP terminal. On another occasion, a steady deterioration in the PSP terminal operation at Goonhilly eventually developed into a total isolation of Goonhilly from SATNET, and we again reconnected the Satellite IMP to the SPADE terminal. Subsequent maintenance on all the PSP terminals by Comsat personnel seems to have alleviated the problem.

We supported Comsat in their field modification of all PSP terminals for performing a 1's complement transformation on the first three T&M words. Effectively, a mismatch between DLE and the first six bytes in the T&M data is created by converting the inserted 0's to inserted 1's in the least significant bit of each byte. This is a temporary measure to allow T&M data to be processed in the Satellite IMPs; the permanent solution requires new software and ROMs in the Linkabit interface.

Comsat restored the SPADE modem equipment to its original state and thereby removed its capability to serve as a backup to the PSP terminal for SATNET operation. Hence, we must now rely on the redundancy in the PSP terminal to overcome equipment failure. Comsat also calibrated the T&M data functions in the modems at all sites.

A cable fault developed in the 9.6 Kb/s circuit between London and Goonhilly, causing packets to arrive at Goonhilly from

London with checksum errors at a sufficiently high packet error rate to prevent the ARPANET protocols from bringing the line up reliably. After an outage of several days, the British Post Office succeeded in repairing the circuit. A fault developed in the 50 Kb/s circuit between the NDRE gateway and the Tanum Satellite IMP. After an outage of almost a week, the Norwegian Telecommunications Office succeeded in repairing this circuit.

The 4.8 Kb/s circuit between ARPANET and the Clarksburg Satellite IMP failed several times. We believe the Bell 208 modem at BBN is the culprit; placing the modem into a looped state and then removing the loop seems to clear the problem. We are keeping the modem under observation.

A squirrel, which entered a Clarksburg power substation, caused an explosion, disrupting all power to Comsat Labs including the Satellite IMP. After five hours of outage, power was restored.

3 PLURIBUS SATELLITE IMP DEVELOPMENT

The major areas of activity during the quarter were support of the continuing integration of the PSAT with other elements of the Wideband Packet Satellite Network and documentation of the PSAT algorithms and implementation which have evolved over the last several years.

During August and September, integration of the PSAT and the ESI was accomplished in San Diego. This work was carried out both by Linkabit personnel interacting with BBN via telephone and by BBN personnel at the Linkabit facility. During the PSAT/ESI integration, several minor bugs in the PSAT hardware and software design were identified and corrected: (1) an error in the PSAT round-trip time calculation resulting from improper processing of the ESI burst delay word, (2) an improper terminating resistor on the Satellite Modem Interface (SMI), and (3) an extra byte being emitted by the SMI hardware. At the end of September, the PSAT was shipped from Linkabit and installed at the Information Sciences Institute in Marina Del Rey.

The integration of the PSAT and miniconcentrator at Lincoln Laboratory was aimed primarily at permitting checkout of the UMC-Z80 interface hardware. By the end of October, software running on the Lincoln equipment was able to successfully bring up the PSAT host access link and exchange datagrams as well as status messages. Two minor PSAT bugs related to the timing of

status messages and the handling of queue overflow conditions were corrected as a result of this integration activity.

Documentation of the PSAT has focused on the development of two major documents. A draft PSAT Technical Report was produced during the quarter. This report describes the basic algorithms incorporated in the PSAT to manage the use of the satellite channel as well as specifying the details of the interfaces presented by the PSAT to subscriber hosts and the ESI. The Host Access Protocol (HAP) has been undergoing continual design and development over the past several years. The section of the PSAT Technical Report which describes HAP, however, is the first formal specification of the protocol to date. This section was also prepared as a separate document for distribution at the Wideband Network meeting at ISI in November. An introductory section of the PSAT Technical Report summarizes basic concepts and definitions relevant to understanding the operation of the PSAT.

The second major document prepared in draft form during the quarter is the PSAT Software Report. Although the heavily commented PSAT code structured by RATMAC control constructs remains the ultimate source of documentation, the PSAT Software Report was developed to provide a higher level of software documentation. Information on the Pluribus hardware management function implemented by STAGE and the application support

functions of SATOPS are included, in addition to information on the PSAT application processes, strips and routines themselves. Basic process interactions and data flow through the system as well as detailed descriptions of individual routines and data structures constitutes the major portion of the PSAT Software Report. A final section of the report discusses the TENEX-based network management and control support software which operates in conjunction with the Pluribus-based routines.

One of the facilities provided as part of the PSAT is the basic instrumentation required to carry out experiments related to speech transmission. The internal speech generation host is a key element of this instrumentation. During the quarter the speech host was modified to be compatible with the internal PSAT service host which supports the management of host streams. Initial experiments were run using the speech host aimed at verifying the speech predictor model developed by Lincoln Labs.

As the Wideband Network expands from its initial configuration to support growing user requirements, it will become increasingly important to have a reduced-cost replacement for the current Pluribus-based PSAT. During the quarter, a study of such reduced-cost PSAT implementations was completed. The recommendation resulting from the study is that a PSAT based on the Butterfly Multiprocessor system currently being developed for DARPA by BBN is probably the best choice for supporting the low-

cost PSAT application while at the same time offering expandability to support next-generation packet satellite throughput requirements. The five candidate approaches we considered are discussed in section 3.2 below. Several of these candidate approaches analyzed involved the implementation of new microprogrammed instructions optimized for the PSAT application. To understand the impact of these new instructions quantitatively, analysis of the PSAT application processing requirements based on instruction counts was carried out. The results of this work are summarized in the following section.

3.1 Evaluation of Microcoding Enhancements to the PSAT

This section presents a numerical basis for the estimate of throughput enhancement achievable by microcoding key PSAT functions. To simplify the analysis, we have focused on only the speedup potentially achievable by microcoding the ENQUEUE and DEQUEUE operations. These two operations are known to consume a large fraction of PSAT CPU cycles. Other operations that are candidates for microcode implementation are expected to affect throughput to a lesser degree. ENQUEUE/DEQUEUE alone, therefore, should give a minimum estimate of the speedup achievable.

We proceed by counting the total number of uplink and downlink instruction cycles executed per PODA frame (approximately 20 ms) by the Host Protocol Module (HPM), and

Channel Protocol Module (CPM) for an assumed steady-state traffic scenario, both before and after microcoding. The difference between those two counts can be applied, for a fixed computational power, to support greater PSAT throughput.

Our analysis is based on steady state uniform flows in a four-PSAT network. The following assumptions are made:

- All stream and datagram messages are 1 buffer or less in length (a buffer holds about 3000 bits of user data).
- Each PSAT originates 1 datagram message per frame to a remote host.
- Each PSAT supports 2 host streams aggregated into a single channel stream.
- One stream message exists per host stream per frame.
- No datagram aggregation is carried out.
- No group addressing is used.
- Steady-state flows are in progress; no setups are being processed.
- Acceptance/Refusal mechanism has been disabled.
- PSAT considered is not the leader PSAT.
- Poller task is ignored (this is and will be handled by a dedicated processor).
- ESI filters the PSAT's own transmissions on the downlink.

The current implementation for queuing operations involves the following macrocode:

ENQUEUE -- 50 instructions

DEQUEUE -- 30 instructions

The execution time of the microcode to implement these same instructions (assuming the SUE semiconductor memory fetch and store times dominate the time expended in logic operations) is:

ENQUEUE -- 27 microseconds,
14 equivalent SuperSUE instructions

DEQUEUE -- 20 microseconds,
10 equivalent SuperSUE instructions

We define the instruction count Nwxyz as follows:

w = c for CPM
h for HPM

x = u for uplink
d for downlink

y = f for fixed (traffic independent)
t for traffic dependent

z = e for ENQUEUE instruction cycles
d for DEQUEUE instruction cycles
o for other instruction cycles.

Thus, for example, Ncdte is the number of instructions associated with ENQUEUE that are traffic dependent on the CPM downlink. For the assumed scenario the following instruction counts apply:

HPM

Nhute	=	900
Nhutd	=	360
Nhuto	=	2370
Nhut	=	Nhute + Nhutd + Nhuto = 3630
Nhdte	=	900
Nhdtd	=	270
Nhdto	=	1599
Nhdt	=	Nhdte + Nhdtd + Nhdto = 2769
Nhu	=	Nhut, Nhd = Nhdt
Nh	=	Nhu + Nhd = 6399

CPM

Ncufe	=	0
Ncufd	=	150
Ncufo	=	987
Ncuf	=	Ncufe + Ncufd + Ncufo = 1137
Ncute	=	1750
Ncutd	=	1020
Ncuto	=	2935
Ncut	=	Ncute + Ncutd + Ncuto = 5905
Ncdfe	=	100
Ncdfd	=	60
Ncdfo	=	302
Ncdf	=	Ncdfe + Ncdfd + Ncdfo = 462
Ncdte	=	2100
Ncdtd	=	1410
Ncdto	=	7045
Ncdt	=	Ncdte + Ncdtd + Ncdto = 10,555
Ncu	=	Ncuf + Ncut = 6842
Ncd	=	Ncdf + Ncdt = 11,017
Nc	=	Ncu + Ncd = 17,859

Total

N	=	Nh + Nc = 24,258
Nf	=	Ncuf + Ncdf = 1599
Nt	=	N - Nf = 22,659

Now with new microcoded instructions, the following will hold:

N'wxyd = (5/30)Nwxyd

N'wxye = (7/50)Nwxye

Applying this we compute:

N'hut	=	2742
N'hdt	=	1941
N'h	=	4683
N'cuf	=	1037
N'cut	=	3765
N'cdf	=	350
N'cdt	=	8103
N'c	=	13,255
N'	=	17,938
dN	=	N-N' =6320
N't	=	16,551
dN/N't	=	.38

This implies that one should expect 38% throughput increase due to microcoding of ENQUEUE and DEQUEUE for the assumed traffic scenario. The total microprogramming improvement that is achievable depends not only on the accuracy of the above analysis but also on the actual traffic conditions which exist and the impact of microprogrammin- other key PSAT operations in addition to ENQUEUE and DEQUEUE. Because of the difficulty in accounting for these factors, we shall use 50% as the nominal throughput improvement achievable using enhanced instruction sets implemented in firmware.

3.2 Low-Cost PSAT Study

A ground rule for this evaluation was that the low-cost design must be at least equivalent in terms of functionality and

performance to the current PSAT. In general, this implies that the low-cost PSAT must be capable of providing integrated PODA control on a 3 Mbps satellite channel and must support multiple interfaces to hosts at megabit rates. In particular, we assume that the low-cost PSAT should be sized to support a sustained user throughput of at least 1.5 Mbps simplex. That is, the PSAT should be able to support at least 1.5 Mb/sec of datagram/stream traffic sent from a local host to a remote host in addition to dealing with up to 1.5 Mbps of additional channel traffic flowing among other network stations. This requirement eliminates the class of reduced capability solutions that might otherwise be considered (e.g., a PSAT implementing only centralized control for datagrams and streams; or a PSAT capable of receiving only at a rate below 3 Mbps). Although we assume that the low-cost PSAT should provide high system availability, we do not require a redundant multiprocessor configuration such as provided by the Pluribus.

We identified five implementation approaches for the development of a low-cost PSAT:

1. MBB Emulation of a Pluribus
2. MBB C-Machine Based Node
3. Butterfly Multiprocessor Based Node
4. Reconfigured SuperSUE PSAT
5. Commercial Minicomputer Based Node

Each of these approaches is described briefly below.

3.2.1 MBB Emulation of a Pluribus

The Microprogrammable Building Block (MBB) is an economical, general-purpose computer system developed by BBN that is optimized for the emulation of other machines. One approach to developing a low-cost PSAT is MBB emulation of a single processor Pluribus running existing PSAT software. It is hoped that this approach will best preserve the investment in existing software. Although direct emulation of the current PSAT (including emulation of all I/O performed by the satellite and host interfaces) is possible, a mixture of emulation plus hardware enhancement is probably more attractive. Of the several approaches which fall into this class, the following specific implementation was selected:

1. Basic MBB processor with 160K words of memory
2. MAR daughter board to implement Pluribus memory mapping
3. MIR daughter board to support op code dispatching and I/O device handling
4. Word-oriented hardware interface (ESI) to the Earth Station Interface
5. Word-oriented hardware interface for a high speed host

6. Byte-oriented 1822 interface for a medium speed host

7. Firmware implementation of selected PSAT operations (e.g. QUEUE/DEQUEUE, PACK/UNPACK, Multiple Word Arithmetic)

8. Modifications of the existing PSAT software for operation in the MBB hardware environment (primarily recoding of particular STAGE functions).

This configuration is expected to fit on 3 or 4 large MBB PC boards (the number depending on the availability of 64K RAM chips). The standard MBP and MBM boards are piggybacked with tailored MIR and MAR daughterboards respectively. Two host interfaces and the satellite interface should fit on a single I/O interface board which would be specially designed for this purpose.

The Pluribus has a 20-bit physical address space and deals with 20-bit pointers by packing and unpacking them into 16-bit words. This activity is implemented via macros called PACK and UNPACK. Similarly, the MBB supports 20-bit addressing directly through its 20-bit word width. However, our approach to dealing with the Pluribus's need for 20-bit addresses on the MBB is to microcode the existing PACK and UNPACK operations. This is viewed as a compromise between simply leaving these operations in macrocode and modifying the existing PSAT software to take

advantage of the 20-bit addressing supported by the MBB.

The maximum throughput of this design is constrained largely by the processing capability of an MBB uniprocessor relative to a 6 SUE PSAT. To compare the two we note that (1) microcoding SUE instructions on the MBB will lead to about a factor of 2 increase in raw execution speed, and (2) firmware execution of ENQUEUE/DEQUEUE and other common PSAT operations will give another 50% speedup (see Section 3.1). Therefore, ignoring I/O, the MBB is equal to approximately a 3 SUE processor PSAT. We now assume that the I/O of the Satellite Modem Interface (SMI) and megabit host requires 4 microcycles per 16-bit word. Assuming 1.5 Mbit/sec host-to-host traffic and 3 Mbit/sec on the channel downlink, the high-speed I/O load on the PSAT is 6 Mbit/sec or a 16-bit word every 2.67 microseconds. Each of these words consumes 540 ns (135 ns MBB cycle time X 4), therefore the I/O load on the MBB processor is .54/2.67 or about 20%. The equivalent number of SUE processors available for non-I/O tasks is (.8 X 3) or 2.4. This is inadequate to provide the same performance as the existing PSAT.

3.2.2 MBB C-Machine Based Node

The MBB hardware described above has been the basis for the development by BBN of a machine, the C/70, optimized for execution of the C programming language. Programs written in C

are compiled into an intermediate language which is directly executed by the C/70. A C/70 processor can potentially provide the foundation for the development of a low-cost Wideband Packet Satellite Network Node. The underlying attraction of this approach is its use of existing familiar hardware to support a higher level language implementation.

As in the Pluribus emulation approach described above, a MBB-C based PSAT should be configured with word-oriented I/O interfaces to support high speed hosts and the ESI in order to conserve MBB processing cycles. Of course, this approach involves a complete rewrite of the PSAT software and associated Pluribus support functions from SUE assembly language and RATMAC to C. We expect that after establishing operation of the PSAT written in C, we could identify selected portions of the code to commit to microcode in order to enhance the system throughput. This procedure has proved quite successful in the implementation of UNIX on the C/70.

Our original thinking was that this approach would result in a greater throughput than the MBB emulation of the Pluribus due to direct execution of intermediate C-language instructions by the hardware. However, this may not be the case. Although a reduction in the I/O load on the processor will result from the use of 20-bit rather than 16-bit words, this effect will be relatively small. We expect that the nodal throughput of the

MBB-C PSAT would not be considerably greater than the MBB/Pluribus PSAT, which was shown to be inadequate in section 3.2.1 above.

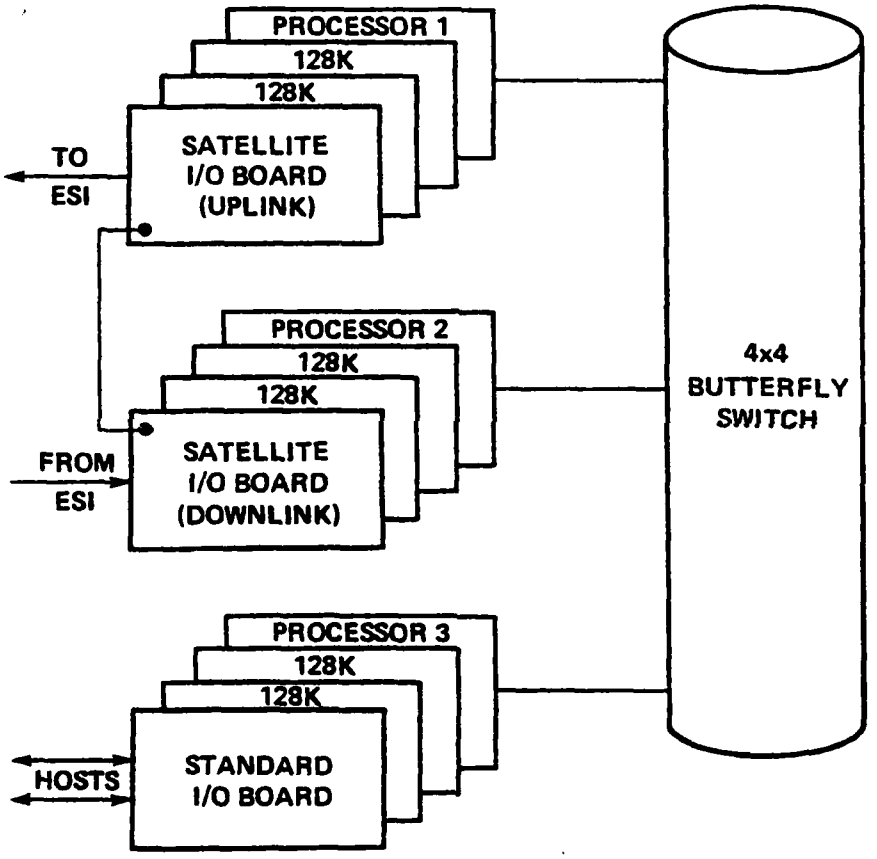
3.2.3 Butterfly Multiprocessor Based Node

The Butterfly Multiprocessor is a next generation general-purpose multiprocessor computer system currently under development at BBN for support of the Voice Funnel and other high bandwidth applications. A unique hardware switch provides for high speed, general purpose, low-cost communication between microprocessor-based processor nodes. The heart of each processor node is a Motorola MC68000 microprocessor with 128K to 1M bytes of memory. Additional bit-slice microprocessors and logic on each processor board support internode communication, memory mapping, protection and other special functions. Since a Butterfly Multiprocessor is designed to be configured with many processor nodes, tremendous computational power can be made available. The ability to configure Butterfly systems with only few processor nodes as well suggests that the Butterfly is a reasonable candidate for supporting the low-cost PSAT application.

A limited function operating system is currently being developed to facilitate system development in the Butterfly hardware environment. Software written for the Butterfly will be

written in the C programming language.

A three processor node Butterfly configured for the low-cost PSAT is illustrated in Figure 1. One processor node is used to handle hosts while the other two nodes are assigned the tasks of satellite channel uplink processing and satellite channel downlink processing, respectively. The instruction counts in Section 3.1 indicate that this assignment of functions divides the host and uplink load among two of the processing nodes approximately evenly. The more CPU intensive downlink processing should still be well within the power of the third processor node. The host processing node uses the standard Butterfly I/O board for interfacing with users. This board supports up to 4 synchronous bit-oriented interfaces (in addition to 4 asynchronous interfaces) at speeds up to 2 Mbit/sec each (4 Mbit/sec aggregate maximum). These interfaces can be used either individually or in pairs (as in the current PSAT) to provide multi-megabit host access. The lack of mechanisms to support precise timing on the standard I/O boards necessitates the development of a special satellite I/O board for interfacing to the ESI. As illustrated in the figure, some form of control line will be required between the two halves of the ESI interface in order to guarantee synchronization of uplink and downlink local time clock registers.



Butterfly Multiprocessor PSAT
Figure 1

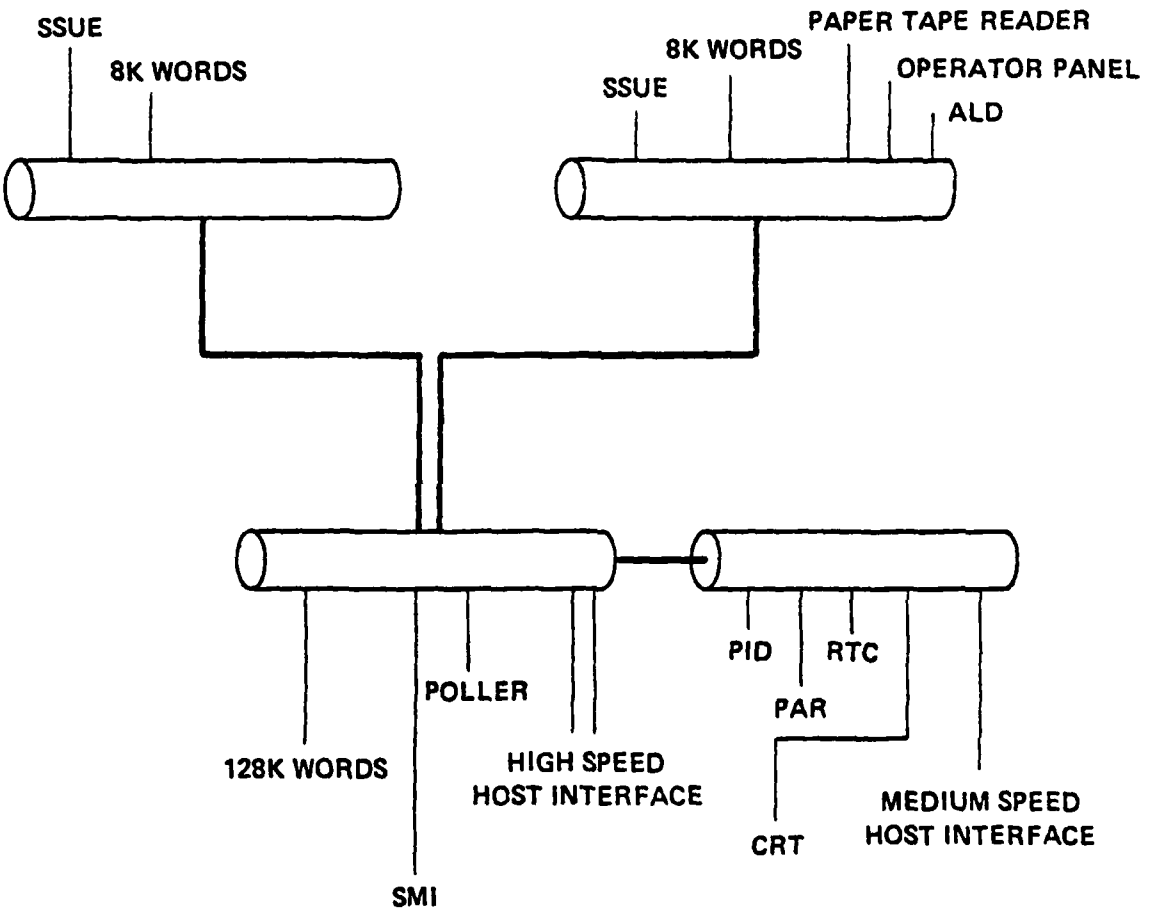
The processor nodes (PNs) of the Butterfly multiprocessor each contains an MC68000. Macroinstruction execution times for common operations on the MC68000 range from about 1 to 2.5 microseconds, depending on the particular operation. To be conservative, we assume that the average PN instruction execution time is 2 microseconds, twice the speed of the SUE processors; thus 3 PNs should be nominally equivalent to the current PSAT configuration. Since the Processor Node Controller provides Enqueue and Dequeue functions as standard firmware, another factor of 1.38 should be included, bringing the processing capability up to the equivalent of about 8.3 SUE processors. To account for operating system overhead, this number should be reduced by about 10%. A further reduction should be included to account for degradation due to I/O operations. As a worst-case analysis, consider the satellite downlink processor which must handle the full 3 Mbit/sec on the satellite channel. At the destination, this PN will also have to pass 1.5 Mbit/sec on to the Host PN for an aggregate I/O load of 4.5 Mbit/sec on the memory. Since the memory bandwidth is 32 Mbit/sec, I/O consumes about 15%. Being conservative, this could degrade the speed of the processor by 15% as well. Therefore, the total degradation in processing due to the operating system and I/O could be as much as 25%. The number of equivalent SUE processors available for PSAT processing is therefore (.75) (8.3) or 6.2. Taking into account the inefficiencies of the C-compiler and the quality of

the MC68000 instruction set relative to the SUE instruction set, one expects the current Pluribus configuration and the proposed Butterfly configuration to provide roughly the same computational power for the PSAT application.

3.2.4 Reconfigured SuperSUE PSAT

The SuperSUE is a next generation replacement for the standard SUE processor used in the Pluribus. Based on a microprogrammed hardware design, the SuperSUE executes the basic Pluribus instruction set approximately twice as fast as the standard SUE processor and fits entirely on one PC card. The off-the-shelf SuperSUE microcode can be augmented to implement key PSAT operations (QUEUE/DEQUEUE, PACK/UNPACK, etc.) to effect a significant increase in raw computational power when applied to the PSAT function (see Section 3.1).

The Reconfigured SuperSUE PSAT approach consists of two processor busses with a single common M/I bus to support all common memory and I/O interfaces as shown in Figure 2. To be consistent with the previous low-cost PSAT designs, we do not configure this Pluribus node with redundant hardware. Redundancy could, of course, be added for improved reliability at an additional cost. All of the existing PSAT software would continue to run without modification on this low cost configuration except where macro calls were replaced by op codes



M/I Bus--SuperSUE PSAT
Figure 2

for the special microcoded operations mentioned above. @begin(Multiple) We expect that two SuperSUE processors with extended instruction sets based on firmware enhancement should be nominally equivalent in processing power to a 6 SUE PSAT system. Each SuperSUE is approximately twice the speed of a standard SUE and another 50% can be obtained via special microcode. The primary question related to the M/I Bus PSAT configuration is the ability of the single common bus to support the combination of processor references and I/O references imposed. We consider each of these loads separately below and sum them to determine total bus loading.

To determine the processor loading on the M/I bus we first make several observations:

1. The nature of the PSAT program is such that about 50% of the instructions are fetched out of local memory and 50% are fetched out of common memory*. On the average, each of these instructions (whether fetched out of local or common memory) will require one operand fetch out of common memory.

2. Each instruction fetch requires 2 one-word memory references.

3. Instruction times are essentially dependent only on memory speed, since the SuperSUE overlaps instruction execution with instruction and operand memory references.

---

* One would like to execute a larger fraction of instructions out of local memory to speed up the processor and reduce common bus loading. Unfortunately, the PSAT currently uses all the local memory available [8K words per processor]. In addition, module positions have already been optimized for maximum local memory references.)

4. Time to do a local memory reference is 700 ns, which is the memory cycle time.

5. Time to do a common memory reference is 1.84 microseconds due to delays through the bus coupler added to the memory cycle time.

For instructions executed out of local memory, the nominal instruction time is 3.24 microseconds. Since this involves one common memory reference which ties up the M/I bus for 700 ns, the M/I bus utilization due to a processor executing only local instructions is .7/3.24 or 22%. For instructions executed out of common memory, the nominal instruction time is 5.52 microseconds. Since this involves 3 common memory references, the M/I bus utilization due to a processor executing only common memory instructions is 3 X .7/5.52 = 38%. Furthermore, since the proposed configuration has 2 SuperSUE processors, each referencing common and local memory half of the time via a single M/I bus, we can sum these two utilizations to get 60% M/I bus utilizations as the total due to processor references.

To compute the I/O device loading on the M/I bus, we proceed in a similar manner. The SMI is the highest speed device on the bus so we consider it first. The SMI must be able to send and receive simultaneously at a rate of 3 Mbps, giving an aggregate burst load of 6 Mbit/sec or .375 words/microsecond. Since each of these references also consumes 700 ns of M/I bus time, the SMI alone loads the bus (.7)(.375) or 26%. The high speed host interface running at 1.5 Mbps imposes one fourth of the SMI load

or 6.5%. To this we need to add the M/I bus load due to a microprogrammed poller which supports the SMI and high speed host. This device is expected to use about 6 memory references at 600 ns every 50 microseconds or 7.2% of the bus bandwidth. The total bus loading due to support the high speed I/O devices is therefore 39.7%. Without even considering the lower speed devices on the M/I bus, we are already up to 99.7% total bus utilization and this is only the average, not a worst-case analysis (i.e., both processors can be executing for brief intervals out of common memory). Therefore, the Reconfigured Pluribus M/I Bus Approach does not appear adequate to support a low-cost PSAT implementation.

3.2.5 Commercial Minicomputer Based Node

A final approach considered is the possibility of reimplementing the PSAT from scratch on a commercially available minicomputer. This approach would involve not only development of new hardware interfaces for connections to the ESI and hosts but would also require a complete rewrite of the PSAT software, potentially in assembly language to achieve the required efficiency. Both 16-bit and 32-bit processors are possible candidates. In the 16-bit category, the Digital Equipment Corporation PDP-11/70 or PDP-11/44 are typical high performance machines. There is an ever-increasing number of possibilities in

the 32-bit category including the VAX-11/780, the Perkin Elmer 3240, and the System Engineering Labs 32/77. From a program development standpoint, the 32-bit machines are preferred since the address map switching required to support the PSAT application on 16-bit machines will almost certainly introduce inconvenience or inefficiency.

Maximum throughput depends upon the particular minicomputer selected. Although it is difficult to evaluate throughput accurately for any particular selection based on the data currently available, several general conclusions can be drawn. High performance 16-bit processors like the PDP-11/70 and 11/44 with cache memory probably cannot support an instruction execution rate greater than about one million instructions per second. Even if these instructions are individually more powerful than those of the current SUE processors, the difference is probably not sufficient to make up for the fact that the PSAT with six 4 microsecond per instruction processors is nominally a 1.5 MIPS machine. Some of the high performance 32-bit processors such as the Perkin-Elmer 3240 may achieve the required instruction execution rate. These systems are quite expensive, however, and achieve baseline performance with top of the line processors suggesting limitations on expandability.

4 REMOTE SITE MAINTENANCE

4.1 Further Explorations into a Definition of Remote Maintenance

The Remote Maintenance Experiment (RME) is part of the Remote Site Module system for the Advanced Command and Control Architectural Testbed (ACCAT). The RME represents research in the general area of maintenance of distributed systems. The first requirement of this research is to understand the basic problems of system maintenance, then to extend this understanding to the multiple processor case, where the processors may be a heterogeneous collection of host systems connected by some network. In some measure, the overall project has two components, one the research into the remote maintenance of systems, the other the actual doing of system maintenance at the various ACCAT sites. These two components are intended to interact so that the ACCAT RSMs can serve as a testbed for the Remote Maintenance Experiment.

In the following sections various aspects of the remote maintenance problem are discussed. These include:

a description of the general problem of system maintenance;

a brief analysis of things which make system maintenance easy;

a discussion of multi-system and multi-site maintenance in a research or development environment;

a description of facilities which would simplify remote maintenance; and

an outline of research topics in maintenance of distributed systems.

4.2 General Maintenance Services

What is remote maintenance? What is the purpose of the Remote Maintenance Experiment of the ACCAT project? What other tasks must be carried out at the same time as the experiment?

It is first necessary to establish what a successful remote maintenance system (RMS) would accomplish. The initial model of an RMS is the ARPANET model. Almost all normal software maintenance functions can be performed from the ARPANET Network Control Center (NCC); it is virtually never necessary to go to a site to deal with a software fault. The NCC can examine the contents of any IMP or TIP, can cause it to be reloaded from a master copy or from an adjacent IMP or TIP, and so on. The NCC also monitors the activity of each element of the network, and has a reasonably complete model of normal activity.

As attractive as this model is, it overlooks some important aspects of the remote maintenance problem. In the first place, the sub-net consists entirely of systems containing no moving parts. In the second place, the system is entirely under control of the NCC; no one else introduces software, and any malfunction which exists is entirely due to the hardware and software which they control, or to the traffic which affects only tables

controlling the network. Finally, the sub-net has no direct human users.

The Remote Maintenance Experiment is concerned with the general operational maintenance problem for a distributed set of (relatively) homogeneous hosts on the network. This means that all of the the following tasks, which are included in the actual operational maintenance of a system, must be considered:

(1) correction of program errors in controlled software, especially the operating system kernel;

(2) on-going monitoring of software behavior and data system integrity;

(3) analysis of hardware and software error reports;

(4) immediate patching of serious software errors;

(5) correction of disk directories and similar data structures after crashes or other software malfunctions;

(6) coordination of hardware maintenance services, as required;

(7) on-going distribution of documentation;

(8) on-going operator and user training;

(9) site visits (in the case of remote maintenance) for general hand-holding and special support for demonstrations and major system changes;

(10) check-out and distribution of software from other sites;

(11) analysis of controlled software for weak spots, inconsistencies, and other problems needing correction; and

(12) installation of new and improved software, and its subsequent debugging and maintenance.

These tasks form the core of any maintenance system; a few of them (as noted) are slightly modified when the maintainer is not co-located with the system. Many of these tasks can be handled as easily from remote locations, but the importance of documentation and training (among other things) is increased by the physical separation.

It appears that, under present circumstances, site visits cannot be eliminated entirely if the software or hardware system changes significantly. Such visits are also probably necessary at other psychologically important times, like major demonstrations, important changes in overall project staff, and the like.

It would seem that the core of the Remote Maintenance Experiment is to determine which of these tasks can be effectively carried out remotely using techniques which are extensions of the current state of the art, and to examine the prospects for significant improvement in such systems by using novel methods. At the same time, the individual sites must be maintained using available technology, or the new tools as they become available.

4.3 Things That Make Remote Maintenance Easy

Before considering the real world situation, where a specific system is being maintained in a multi-site research program, it is worth while to step back and think about the things which would make the process simpler. Some of these, it will be seen, can be adopted in most maintenance situations, while others are ruled out by hard constraints.

In general, the greater control that the service organization has over the system the easier its job is. The following would contribute to this increased control:

(1) if the system has a relatively narrowly defined set of functions;

(2) if the system was constructed initially by the same organization;

(3) if the system can be changed at will by the maintaining organization (subject only to internal review, for example);

(4) if the system internals are completely hidden from the end-user; and

(5) if the system users are not competent or allowed to change the system.

It is quite clear that the ARPANET maintenance organization has all of these advantages. The current RSM system is based on a general time-sharing system, the UNIX operating system. Perhaps the greatest advantage that this system has for most of its users is that the organization that created it does not maintain it. Its appeal lies partly in the fact that the

operating system interface itself is fairly uniform from site to site, and partly in the fact that any reasonably clever user can bend it his own way. Therefore, none of these advantages is available to the maintainer. In fact, the ease of system modification makes it quite difficult to track updates very closely; this problem will be discussed at greater length in the next section.

An operational system based on the current RSM would not suffer as much from these problems as the current system does, even if it were made by simply copying the code which is running at the ACCAT sites. The code copies would most likely be binary images, absolutely controlled from the central location.

On the other hand, the better prepared the service organization is, the more effective it is. The following would contribute to this:

(1) really good documentation;

(2) trouble-shooting staff members available more or less any time of day that the system user wants help;

(3) specially trained trouble-shooters; and

(4) site representatives within the service organization.

One would think that these are all things which could be obtained simply by spending more money. Actually, the use of the word "really" to describe the good documentation glosses over the fact that not too much is known about how to prepare it, and some

research effort is needed here.

A user who is in some sort of trouble wants to get more or less instant response, so the trouble-shooters should be available whenever the normal system users are around. In the current scheme, this is handled by providing a phone number which can be called 24 hours a day; unfortunately the user will not receive an immediate response from the person who answers that phone after normal business hours. As a result, the user is often frustrated. An alternate communication path, through an ARPANET mailbox, is available, but this may seem to provide an even more disembodied response to the user. It is true that this sort of buffering does protect the service organization from those users who abuse the system, but the perceived low level of service for others hardly compensates for this advantage.

Good remote service trouble-shooters are quite special people. They must enjoy the complete confidence of the person who is in trouble, or they will not be able to do their job well. The user may be quite rattled, because the system is doing something terrible (for example, it just crashed). Or the user may simply be exercising selective memory about what just happened, not remembering or reporting all of the relevant details; in fact, the user cannot be expected to know what is relevant. Good trouble-shooters never get ruffled, never admit that they were on a fishing expedition.

Although it is tempting to save staff by using ordinary system programmers to do this kind of work, one quickly learns that few of them have the right temperament to do it successfully. For example, most programmers cannot keep the secret that they do not know what is wrong. It might seem that most of the time the answer would be known if the person were really clever. Unfortunately that is not the case. Bugs tend to get fixed as time goes on, and the ones that are left are increasingly obscure. Problems which are induced by hardware failures are truly random, and although these can be categorized, the ones that are easily understood are often corrected automatically by the software as time goes along. Other programmers simply act and sound too nervous, and may actually induce an already upset user to do the wrong thing. Finally one must be especially sensitive to the person who takes a perverse sort of pleasure out of abusing people who are not on the inside.

The site representative is a different sort of person. The effective trouble-shooter enjoys solving puzzles, unraveling the various threads, and fixing the system. The site representative is concerned about eliminating problems before they arise. This person is a sympathetic advocate who helps oversee the initial installation, does training, visits the site on occasion, and leans hard on the documentation people to see that everything is clear. Again, programmers do not often make good site representatives because they simply do not have the right kind of

sympathies. The site representative must understand the user and the user's approach to the system more clearly than most programmers do. On the other hand, this person has to resist the desire of the user to have the system endlessly tailored; programmers are all too eager to accept this challenge.

4.4 Multi-system and Multi-site Maintenance in Research Systems

The situation in the BBN-UNIX and Remote Site Module world is far from the ideal described in the last section for several reasons. The most obvious difference arises from the fact that this is a research system and, therefore, subject to continuous modification for a variety of reasons, some of them conflicting. This does cause significant complications in the maintenance of the system, as discussed below.

The other clear departure from the ideal situation is that the remote maintenance functions are being carried out by the research staff members who are attempting to develop novel approaches to the solution of these problems. These staff members must, of course, be fully familiar with the current methods, but it should be recognized that they are distracted from the research efforts by the day-to-day operational tasks, and, furthermore, they are not likely to be temperamentally the best suited people to be doing this work.

At one time, there was a plan to make the configuration of each Remote Site Module (except for the one at BBN, which serves other purposes) conform to some standard. While this can help, it is by no means essential. The differences in usage at the different sites also cause some complications, but do not present insuperable difficulties.

A few statistics about the BBN-UNIX system will give some perspective to the complexity of the system and the magnitude of the variations which it has undergone.

> The system sources (including the kernel portion of the Network Control Program (NCP)) are stored in 5 directories; there are 153 files, containing approximately 40,000 lines of text. There are relatively few comments in the text.
>
> There are 528 conditional assembly switches in the code (#ifdef and #ifndef statements); these are are controlled by 44 distinct variables.
>
> The installation document for the kernel and network code is 50 pages long.
>
> There have been 33 "frozen" versions of the operating system since BBN installed the 11/70 in June 1978 (approximately 1 per month).
>
> The history files describing the changes to the system contain 1900 lines of very short comments, averaging about 5 lines per revision; this means that there have been about 400 revisions made to the system in the same period, or one about every three days.

Modification to the system is often quite difficult because of the kernel space crowding which is endemic in PDP-11/70 operating systems. The operating system kernel (including the portion of the network code that is in the kernel) has 64K of

instruction space and 64K of data space available. The current BBN-UNIX systems are completely full.

The user community expects all of the standard Bell utilities (nearly 200 of them) to work on any UNIX system. In addition there are dozens of other commands which have been developed for the system. Certain weaknesses in the original system have been corrected or compensated for by introducing new system calls. These include the RAND port() and empty() calls, and the await() and capac() calls, which have been included to improve interprocess communication. The ordinary file and terminal handling code has been extended to handle the network, and there is a raw message interface as well as the Network Control Program (NCP) facility.

The control of the system and utility software is based on simple manual disciplines, which work relatively well because only a few individuals are involved in the system development. The situation becomes more complex when development is going on at more than one site; some of the graphics code is being developed at Naval Ocean Systems Center, and BBN recently received a shipment from RAND. Fortunately most of this is user level code, and therefore is easier to check out.

The following questions must be asked about the kind of service which is provided to the remote sites:

(1) Is there a central development site which has configuration control over all other sites, including other development sites?

(2) How closely should the remote site software track the software in use at the central site?

(3) How do changes developed at one site get incorporated into the systems at the other sites?

(4) How great an effort should be made to get all sites to run the same software?

It might be noted that the same questions really apply to the software running in any multi-machine facility, and even to the software running on any single machine which is used for development. In the latter case, the questions apply to the "normal" software which is run daily. For example, should it usually be some basic system, or should experimental software with all sorts of specialized features be used in an operation like the one at BBN?

The first question has been answered with a qualified "yes" in this project. The central site (BBN) nominally has configuration control over the software, and there is a long list of the names of controlled software modules. The operating system itself leads the list, and is the most fully controlled element.

The optimum frequency of distribution appears to be quite situation dependent. In this project, the time between installations is between 6 and 12 months. It would seem that low

frequency changes like this are particularly desirable in non-development sites. Perhaps more frequent changes would be better for NOSC, but this would make it quite difficult to keep the same software at all the sites, since it is not likely that they would all like to be uprooted every three months or so. Virtually all of the approximately 400 variants to the system have run at BBN; most of the time these changes are invisible to the user, since they are normally upwardly compatible, and generally do not cause significant disruption. Some of the changes have been serious enough that they have caused discomfort through frequent crashes or other bizarre behavior.

Currently, new things get distributed primarily from BBN. They are first incorporated in the main stream of development, then sent out to the sites. At first glance, there might be an argument for more mutual distribution, especially of utility programs. There are several drawbacks to this approach. In the first place, unless there is a highly sophisticated system control system in place, the maintainer never knows what must be fixed when a new version comes out. Second, the maintainer who is called in to work on a broken program would need the same sort of sophisticated control system to determine what was in fact being repaired. Third, it is hard to guarantee the compatibility of a particular modification with the operating system kernel in use at the site. Finally, there are difficulties in assuring that there is adequate testing and documentation with central

distribution; these become worse as the number of actors increases.

Because UNIX system code is easy to modify, there are very few duplicate systems in the world. In the RSM project, there is an attempt to have the same programs running at all the sites (other than BBN). This has not been completely successful, since it takes a great deal of time to perform an installation over the network. In fact, until better tools are available, as described in the next sections, it would seem wiser to do major installations during routine site visits. This would achieve greater synchronization of sources. Between major installations, smaller changes, in the order of patches to running programs, or the provision of new utilities, could be done over the network.

4.5 Improving Remote Maintenance Incrementally

4.5.1 General

If one assumes that part of the Remote Maintenance Experiment is providing service to the current users, and that the effort which can be made on research depends, at least in part, on the efficiency of the service, then it is plausible to spend some time investigating available incremental improvements to the system. In doing so, the following criteria seem to help in deciding what to do first:

Whatever is done should be consistent with the current system requirements. That is, if a short cut is chosen, it should not decrease needed system capabilities, but may add or subtract features which are not fundamental.

Whatever is done should be consistent with the best guess as to the requirements of potential *remote* maintenance research activities.

Since the systems which are part of this experiment are used for research, the improvements should not noticeably decrease their value as research tools.

When practical, the changes should be aimed at simplifying tasks which must be performed repeatedly.

Among the most attractive short-term improvements are tools which simplify debugging and repair of programs which have failed and tools to repair damaged file systems. The latter improvement has been described (QTR 18) and is under development. There will be many specialized program debugging tools; one of these, a core dump analysis tool called 'udb', is described below.

The discussion in Section 4 demonstrated the complexity of controlling the software in a multi-system, multi-site research facility. Several of the tasks in system maintenance (fixing programs, monitoring performance, analyzing software, distributing programs) require some sort of control if they are to be carried out efficiently. The fundamental importance of the control function is further described in "Toward a Theory of Remote Maintenance."

System control consists of two major elements: a data base describing the system and a set of programs which give the user access to that data base. In an initial implementation, the data base would probably be quite limited in the variety of information which it contains; for example, it would contain lists of dependencies, pointers to documents, and installation procedures. In the future, however, the data base would contain a wide range of information, such as performance data, for the various parts of the system.

Another potentially interesting area is the development of programs which make the user community more self-sufficient. For example, a bug reporting system which depends on the aforementioned system data base could contain a two or more level referral procedure. This would allow the community at the RSM to have some help when the central maintenance site is unavailable.

Over the past two years, various monitoring probes have been inserted in the kernel of the operating system; additional probes have been suggested. At this time, it would not seem that further development is warranted, pending the development of a performance model which would depend on the monitored information.

4.5.2 A System Dump Analysis Tool

Often a system problem must be diagnosed from no more than a system dump and a vague description of the behavior of the system just before the crash. The UNIX debugger, 'adb', is sufficient for many user-level programs, but is inadequate for crash dump analysis, because:

(1) the format of crash dumps differs from that of ordinary UNIX core dumps,

(2) the format of the executable file from which UNIX is booted differs from that of ordinary UNIX executable files, and

(3) 'adb' is unable to handle C structures (records) symbolically.

Structures are used very heavily in the kernel, and calculating the offset of each member of every structure is a tedious, error-prone process, as is entering such offsets to explore the contents of a structure.

Several modifications and extensions were made to 'adb' to solve these problems. First, options were added to adb which would automatically set up the internal storage map for kernel files and system dumps. Second, predefined formats were devised for all kernel structures. Each member of the structure is displayed in the appropriate format (octal, decimal, etc.) and labeled. The formats are defined as macros suitable for use by the general UNIX macro preprocessor, "m6." The new command, 'udb', invokes 'adb', but pipes the user's input through this

preprocessor. This combination gives the user the full power of 'adb' (since the preprocessor transparently copies its input to its output when it does not find a macro), plus the ability to call on predefined macros in order to print out the contents of a kernel structure.

It is necessary to recalculate structure member offsets by hand any time the definition of the structure changes, and change the predefined format string appropriately. A future change to the compiler could make this information available directly to the debugger.

4.5.3 A Preliminary System Data Base and Control System

A preliminary data base for the remote maintenance system would include the following elements:

source elements, object versions, and documents for the system;

locations and version information for all source elements;

locations and version information for documents;

locations of corresponding object files;

descriptions of sites; and

current and planned releases to each site.

The organization of the file system in RSM has been used to contain this information. In most cases, the person using the

file system needs to know where to look in a general sense; only the main portion of the manual has sufficiently patterned names to allow a program to find the correct file most of the time.

The current system does not support the presence of multiple versions in a single hierarchy. The following solutions have been proposed:

> multiple copies of the file, each complete, within a single directory which represents the file in the current system;
>
> a single file with imbedded "deltas" or changes to the text (this is the approach taken by the Source Code Control System); and
>
> a baseline file with a parallel "delta" file, either in the same directory or in another directory (this is similar to the approach taken in the current manual page management system).

The first of these solutions is the simplest extension of the present method, but is unsatisfactory because the storage space demands quickly become unreasonable. The SCCS approach is quite compact but has the disadvantage that retrieving the most recent copy of the file can require significant processing.

The correct identification of a source or object element must be based on some non-ephemeral information. In the UNIX file system, the creation date of a file is not a fixed quantity, but may be changed by touching the file, moving it from one file system to another, or touching some of the elements of its inode. In the Version 6 system, there is no creation date. The creation

date of a copy is the current date, not the date of the original. Clearly, this is not a satisfactory record keeping quantity. The system control system (SCS) must enter some information into the file; ideally this information should be available (in the case of a source file) for transmission to the object representation. The easiest way to accomplish this is to have the SCS maintain certain lines containing symbol definitions at the beginning of each module. In order to guarantee uniqueness of these symbols, the compiler should be extended to allow an additional character (for example, the commercial at sign '@') to be used in symbols. Whenever an element is entered into the system, the symbol definitions would be added or updated automatically. The SCS would, of course, check first to see that some change has been made whenever the file is updated. These conventional names then can be used within the file to identify the source.

At the same time, whenever any element is entered into the data base, the overall directory of locations will be updated as necessary. The actual locations of 'current' versions (those which are accessible, including the most recent version entered) should be more or less invisible to the user; there should be a tool available for extraction of elements, and a corresponding tool for the return of elements.

A pre-prototype of the data base directory has been constructed within the framework of a Version 7 system. This system, rather than BBN-UNIX, was selected because certain text processing programs, notably the stream editor 'sed' and the report generator 'awk', were available, along with the more advanced Bourne version of the shell (command processor). This pre-prototype system now consists of four main commands (all implemented as shell files):

mkcont, which analyzes a complete directory tree and constructs a sorted list of lines consisting of element names and complete pathnames, the list being augmented with the names of corresponding Makefiles and Build.info files;

fileloc, which accesses the directory and produces a list of elements matching the input line specification;

getfile, which extracts copies of the elements which correspond to the input line specification, generates a control file for the return operations, and places a journal entry in the central control file; and

storefile, which uses the local control file to return the elements, making backup copies if they have been modified.

The original pre-prototype data base directory has been mechanically constructed. In some cases, this will be inadequate, and individual modifications will be needed. The required information will be captured as part of the storefile operation.

The construction (and installation) of any particular command is controlled, in most cases, by either a Build.info or a

Makefile. In many cases, these files, which have been constructed by hand, are not complete. For example, system header files are often omitted from the Makefiles which accompany the Bell distribution. The current version of 'make' does not support the construction of archives, while the current version of 'build' requires the production of long, repetitious Build.info files even for the simplest directory. A merging of the facilities of these two programs is underway, using the syntax of 'make' as the starting point, on the assumption that this will be more widely understood. Additional tools are required to assure that the header file dependencies are properly recorded, and that the versions of the headers corresponding to the entered version of any source element are recorded as well.

A more complete list of the SCS functions which allow programs, subroutines, and documents to be entered, extracted, and identified follows:

> display - retrieves the code and associated documentation for examination. This command is not intended for use when the code is to be modified and returned to the system under the same name, but may be used when a new element based on a old one is being prepared.
>
> extract - retrieves the code and checks the program out to the user. In order to enforce this, the real user id must not be on the short list of anonymous users (root, bin, daemon, and any others which a parameter file contains). Only one user at a time may have an element checked out.
>
> install - receives the new code segment and the associated documentation files, transfers them to the appropriate files, generates object code when

appropriate, saves old copies, and installs new ones. These operations may be performed only by the person who has checked out the old version.

nevermind - a null case of install, used when, for one reason or another, the user decides to leave the current version intact.

fallback - replaces the current version with a previous one.

delete - equivalent to installing a null program in the sense that no historical versions or documentation are removed, but only the current version (in /bin or in a library).

rename - equivalent to extract - install - delete, but neater, or at least less cumbersome.

annotate - allows the addition of notes to the logs, indicating current problems, future modifications, or any other reasonable information.

document - generate various forms of documentation.

chdocs - modifies documentation files only. This command is particularly important in the maintenance of definition, design, and reference materials.

purge - transfer old versions to the archive, or to a backup file system.

In addition, there are command functions which allow the manipulation of site information. Sites may be added and their software described. These commands also generate distribution tapes/disks, or cause the transmission of updated information over the network.

site - defines a site or processor which receives information, and records essential information which describes the software at the site and the software which will be maintained.

transmit - gathers the information together and transmits it to the site.

4.5.4 An Active Programming Environment

Interactive programming environments include such facilities as smart editors, advanced testing tools, spelling correctors, and the like. A system with a comprehensive version of the data base which was described in the last section could extend the programming environment in other ways.

The ability to locate useful information could be extended to include the identification of standard building blocks which could, in turn, be used in the development of new programs. The UNIX system and its hundreds of utilities contain a wealth of potential building blocks; more effective methods of identifying these elements, extracting them from their current contexts, generalizing them, and making them available are needed if the full value of the previous work is to be realized.

It is often difficult for an individual programmer to avoid errors simply because of blind spots in the approach to design and debugging; users have similar difficulties in the way input data is constructed. The system data base can become the repository for other types of information about program performance, patterns of errors, and frequency of use. The error pattern data could be used, in some cases, to help the programmer identify some of the blind spots; in other cases, the programmer could build in references to this data to help the user in the prompt identification of errors.

4.5.5 Documentation: On-line and Paper

One of the watchwords in the computer community is that documentation is never prepared, or at least that useful documentation is very rare. The situation is, in some ways, better, and in other ways worse. The UNIX system, for example, has fairly good user documentation. Every command has some sort of entry in the user manual, and the more complicated ones have reference manuals and some sort of tutorials.

On the other hand, experience shows that many people do not agree with the statement that the documentation is good. Most users learn how to do things by word of mouth. The first question one might ask is whether the user has enough access to the documentation. At BBN, most regular users have at hand a paper copy of the manual. Further, the entire user manual is on-line, and can be referenced by typing a simple command, usually 'man commname', where 'commname' is the name of the command of interest. The few exceptions are annoying to the users, and should be rectified (by using the index, which is a part of the system data base directory, as the primary access path, rather than the file name).

Early in the project, it was thought that the lack of an index was a factor, and a rather good hand-constructed index was made available for the commands. There is no evidence that anyone actually uses this index, except for the people who

prepared it. At about the same time, it was thought that some of the manual pages, especially those which describe system calls, should be reworked on more logical grounds. It appears that these more logical groupings have not been more successful, especially if the manual entry is larger than a couple of pages.

There are some things which seem to be true about good documentation. One of them is that English should be written in a fluent style. Another is that technical writers with no personal technical competence seem to be able to produce better user documentation than people who really understand the programs, although there are some notable exceptions to this. It would be interesting to examine this problem in order to find other characteristics of good documentation.

Ever since the beginning of time sharing, there has been an interest in on-line documentation, yet there seem to be few, if any, examples of successes in this area. Many of the more successful examples prove to be successful only for small user communities. Among the large number of attempts, the follow general sorts of on-line documentation have been tried:

Entire manuals are made available directly in some way.

Manual searching procedures, using either hand-generated or machine-generated indexes, are available. The hand-generated index has been discussed briefly above. The Version 7 UNIX system has a machine-generated index which needs to be evaluated in this context.

More extensive descriptions written especially for on-

line use have been prepared. These are sometimes retrieved by keyword, sometimes by moving around on a tree.

User crutches, such as command completion, are included to support the occasional user.

One can speculate about things which might improve the usefulness of on-line documents; some ideas include:

The documentation must be written especially for on-line use, with care taken to see that the text appears in certain ways on the face of the display.

The user would like to have tools to reach into larger, more comprehensive documents as needed. The index should include references to all levels of documentation in a uniform way.

The documentation should not make unnecessary distinctions. For example, the documentation for a formatter operation should probably be available in the same way that the information about invoking the formatter is made available.

The availability of a documentation editor, to allow the user to cut and paste the original document to suit an individual view of the world, without requiring the user to take overt action to make a copy, and without destroying the system copy.

Some method to slip easily from the document to try something, and to return to the document without difficulty. Even in a system like UNIX which allows one to create processes quite cheaply, one does not have a sense of moving easily from active process to documentation and back.

Some method of appealing to another human being in the context of an on-line interaction with both the manual and the command.

Some of these things could be tested by using a modified version of available screen editors. At the same time, it is probably worth asking if the perception of having greater

bandwidth when reading paper copy is simply a psychological block, or if it has some other basis.

5 INTERNET DEVELOPMENT

5.1 Summary of Past Quarter's Work

During this quarter, we have been conducting frame-level tests of the RSRE X.25 block-interface hardware with the Telenet Network Engineering support personnel. The number of retries has been higher than anticipated, due to the occurrence of unexpected results in several cases; there is still a discrepancy between the Telenet specification and the RSRE implementation having to do with a FRAME REJECT state. After finishing these tests, we began writing the special purpose software necessary for packet-level tests with Telenet, soon to begin. The tests mentioned here are all at 1200 baud with a dial-up line to Vienna, Virginia.

We finished the replication and checkout of an enclosure with power supply for the ACC DEC-11 VDH interface. Subsequently, this enclosure was delivered to UCL for installation of the VDH interface currently part of the PDP-11/40 serving as the UCL gateway between the SATNET Goonhilly Satellite IMP and the ARPANET London TIP. The VDH interface eventually will be used with an LSI-11/02 gateway replacement machine.

Below are summarized several hardware problems which were manifested in the operation of the ARPA-sponsored gateways during the last quarter. Normally, we are involved only in the

detection of problems with the gateways, inasmuch as maintenance contracts with Digital Equipment Corporation (DEC) are purchased for each PDP-11. However, diagnosing the difficulty with the NDRE gateway fell entirely into our hands.

A failure in the NDRE gateway circuit between the ARPANET NORSAR TIP and the SATNET Tanum Satellite IMP was caused by several problems. First, to correct a clearly evident malfunction, BBN field service personnel changed cards in the Host interface on the NORSAR Honeywell 316 TIP. Afterwards, in spite of DEC field service personnel giving the PDP-11/40 a maintenance check, the gateway continued to be nonfunctional. Eventually, we traced the problem to a faulty EIS (Extended Instruction Set) card in the PDP-11/40; the multiply operation was demonstrably inaccurate. Apparently DEC service personnel do not check this card during PDP-11/40 maintenance and therefore were unable to diagnose the problem.

A failure in the BBN gateway between the ARPANET BBN40 TIP and the SATNET Etam Satellite IMP was traced to a burned out voltage regulator card in the PDP-11/40. DEC field service personnel replaced the card and three fans whose bearings had seized.

5.2 CMCC Development of Catenet Performance Measures

We implemented the software in the CMCC program for processing the new performance-measures message formats to be used in evaluating catenet performance dynamically. These message formats include:

- CPU idle time (a measure of how heavily the gateway is loaded);
- Packet delay across a gateway;
- Gateway-to-gateway delay (actually, the round trip time of a special packet echoed from a specified gateway);
- Throughput (bits);
- Queue occupancy traps (a signal for when the occupancy of a queue goes above or below a certain threshold value).

The initial performance-measures specification was revised to include the minimum and maximum delay values in addition to the cumulative delay values in the message types. We therefore revised the code in the CMCC program to reflect the new definitions and to display incremental rather than cumulative delay values by default. Full testing of the code awaits a gateway in which the new message types are implemented. The following subsections detail these performance measures.

CPU Idle Time

CPU idle time, report type 8, gives an idea of the amount of time the gateway machine is not doing useful processing. The

purpose of this is to find out when the CPU becomes saturated, which will be the case if the proportion of idle time becomes very small. The report consists of two 32-bit counts following the monitoring header containing:

1. The amount of CPU idle time since the gateway started, in milliseconds.
2. The time since the gateway started, in seconds.

Packet Delay

Packet delay, report type 9, refers to the length of time a packet stays in the gateway. The measurements of this delay and of gateway to gateway delay are related; measurement of one begins where the other ends. The model used here assumes that gateway processing takes place in three parts: network I/O, queuing, and routing. Implementation considerations will affect just where the packets can be time-stamped on their way through the gateway; for some gateways it may be possible to stamp a packet at the network I/O level, while for others it may not be possible until the packet enters the routing processing. Thus, this specification does not define where the boundary should lie. It is important, however, that together the measures account for all the delay that a packet will experience as far as the gateway is concerned. It is recommended that the packet delay be made to refer to as large a fraction as possible of the time the packet spends in the gateway. The report, consisting of two 32-bit

counts and two 16-bit counts with all delays in milliseconds, is as follows:

1. The total number of packets processed since the gateway started (32 bits).
2. The total delay for all packets processed (32 bits).
3. The minimum delay experienced by a single packet (16 bits).
4. The maximum delay experienced by a single packet (16 bits).

Gateway-to-gateway Delay

Gateway-to-gateway delay, report type 10, can represent the measured one-way delay directly, provided the gateways obtain equipment for time synchronization. Currently, however, only the round trip delay can be determined, assuming that the gateways will use echo packets to find the round trip delay to each of their neighbors.

The report format is a table ordered by internet addresses, considered as 32-bit unsigned integers. Each table entry consists of an internet address followed by two 32-bit counts and two 16-bit counts. The internet address is the neighbor address for which this delay applies. Of the 32-bit counts, the first is the cumulative total of the echo packets returned by the neighbor since this gateway started, and the second is the total delay experienced by those returned packets, in milliseconds. The two 16-bit counts are the minimum and maximum delays, in

milliseconds, for a single packet sent to the neighbor. There will be one table entry for each neighbor address, so that if a gateway is a neighbor on two networks then it will have two table entries. There will be an entry for each such address for each neighbor that replies to the echoes, whether or not that neighbor is a routing gateway. The table size may grow as new neighbors come up while a gateway is running, but it may not shrink; the entry for a gateway that stops replying will simply remain unchanged. This format is depicted below.

```
Internet address of first neighbor,
        (lowest network number)
Total of echo packets returned by this neighbor
        (32 bits)
Total delay experienced (32 bits)
Minimum delay to this neighbor (16 bits)
Maximum delay to this neighbor (16 bits)
                .
                .
                .
Internet address of last neighbor,
        (highest network number)
Total echo packets returned (32 bits)
Total delay (32 bits)
Minimum delay for this neighbor (16 bits)
Maximum delay for this neighbor (16 bits)
```

Bit Throughput

In contrast with the packet throughput report, which has its emphasis on the number of packets a gateway can process, the bit throughput report, report type 11, focuses on how fast a gateway's network connections can accept or deliver data. The report is a table of pairs of 32-bit counts ordered by interface; the first count in each pair is the cumulative total of bits

processed coming in at that interface, and the second is the output count. Interfaces are ordered as in the gateway description message, report type 0. There are two extra 32-bit counts at the end of the message: the first is the total of bits dropped, and the second is the time since the gateway started, in seconds. The counts for the interfaces include all traffic at that interface, including control traffic and messages originating at the gateway. This format is depicted below.

```
Input count for first interface
Output count for first interface
                .
                .
                .
Input count for nth interface
Output count for nth interface
Dropped count
Time since gateway up
```

Queue Occupancy

Queue occupancy, trap type 3, is a trap message which is sent by the gateway whenever a queue length exceeds a threshold percentage specified in the trap request message, or when the occupancy falls below that threshold after having been above it for some time. If a queue is loaded such that the threshold occupancy is continually being passed in each direction, a large number of these traps would be generated in a short time. To avoid this, there should be some minimum time interval between successive trap messages. It is left up to the individual gateway implementors to decide what this time interval should be;

experience with using this trap type will probably suggest a reasonable value. Note that this replaces the earlier queue full trap described in IEN 131. The percentage occupancy trap, however, is more useful than a queue full trap; if a queue becomes full, the gateway is probably already dropping packets, and the latter fails to provide an early warning. In any case, a queue full trap is just a 100% percentage occupancy trap.

The DO TRAP message for this trap type requires an extra piece of information: the percentage occupancy of the queue which is to trigger the trap. This is expressed as an integer in a single byte following the report ID field in the DO TRAP message. A gateway should only use one value of this threshold at a time, so that a second DO TRAP message will supersede the previous one if the threshold value is different. The DO TRAP message for this trap type has the format:

Bits	Contents
0	1
1	1
2-3	0
4-7	0
8-15	3
16-31	report identifier
32-39	occupancy threshold

The trap message has the following format:

Bits	Contents
0-7	Interface number of Queue.
8-11	Input(0) or output(1) queue.
12-15	Above(0) or below(1) the specified occupancy.
16-23	The occupancy percentage used as a trigger.

6 MOBILE ACCESS TERMINAL NETWORK

6.1 Summary of Past Quarter's Work

Summarized immediately below are the tasks worked on during the last quarter in the development of the Mobile Access Terminal (MAT) Red subsystem. Section 6.2 presents a detailed description of an important element implemented in the microcode design of the C/30 packet switch processor serving as the Red processor, namely, satellite channel event triggers. Section 6.3 illustrates the satellite channel packet formats for the various control and data packets.

We installed in the MAT test-facility rack a pre-delivery C/30 packet switch processor with 64K words of memory, temporarily made available to the MATNET project by the BBNCC (BBN Computer Corporation). This unit is currently being used for MATNET software testing and development. In particular, we began the testing of the new macrocode and microcode software for the I/O drivers, including the 1822 Host-to-IMP interfaces and the Red/Black interface. Initial tests concentrated on the integrity of the microcode software for handling the real-time clock and the satellite channel event registers (see section 6.2).

We finished the design and implementation of the Satellite IMP macrocode software for the 1822 Host-to-IMP interface. This

software is necessary for interfacing the LSI-11/03 Terminal Interface Unit and the LSI-11/03 Gateway to the Satellite IMPs. Insertion of the interface into the Host Protocol Module (HPM) was non-trivial, due to the inadequacy of the software hooks built into the HPM. Testing of this software has already begun.

As a preliminary step to building a MATNET Satellite IMP, we wrote the microcode software which emulates the hardware satellite channel interface sans the T&M data interface, which currently resides in the Honeywell 316 SATNET Satellite IMP. This is a logical step, since it allows us to test the microcode software with existing macrocode software known to be working correctly in SATNET. To date, we have operated two C/30s with 64K words of memory through the satellite channel simulator, each running the SATNET Satellite IMP program. The distributive decision algorithm in each unit successfully processed control packets from both units, while internal message generators were used to load the satellite channel. Next, we began writing the microcode software specific to MATNET, including the COMSEC drivers and the scheduling of packet transfers from the Red processor to the Black processor.

One of the problems that plagued us in the beginning was that the Satellite IMP would not send MONITOR reports to TENEX whenever CPODA (Contention Priority-Oriented Demand Assigned) satellite channel access protocol was running. Other channel

protocols presented no problem. For our earliest tests of the microcode software, we essentially ignored the problem and simply ran FPODA (Fixed Priority-Oriented Demand Assigned) satellite channel access protocol. Eventually, when the problem could no longer be ignored, we were able to exonerate the software implementing the Channel Protocol Module and the MONITOR fake host. This left as the most likely candidate the software for the Reliable Transmission Protocol (VDH) which provides the foundation for the Host-MATNET access protocol. Further examination revealed that the microcode software was mishandling DLE doubling. (DLE doubling is the procedure whereby extra DLE characters are inserted into the data stream to circumvent the hardware removal of single DLE characters naturally occurring. Coincidentally, the bits sent to TENEX signifying CPODA operation were interpretable as a DLE character.) Once the DLE doubling microcode software was corrected, MONITOR reports appeared normally when running CPODA satellite channel access protocol.

6.2 Microcode Design of Event Triggers in the C/30

In SATNET, special-purpose satellite channel interface hardware contains a 16-bit real-time clock, incremented every 10 microseconds, which forms the reference time for initiating events and which is read by the Satellite IMP software for time-stamping packets upon arrival. Also implemented in the hardware

are clocked event registers controlling the precise time when satellite channel events occur (for example, when the transmitter turns on). In operation, trigger pulses are generated by comparator circuits between the real-time clock and the event registers.

In MATNET, the C/30 packet switch processor contains hardware clock triggering at 100 microseconds and special-purpose microcode software implementing the real-time clock, event registers, and comparator circuits. The direct implication of the longer fundamental clock interval is that: (a) larger guard bands between packet transmissions are necessary, and (b) microcode realization of comparator circuits must rely on comparison results including not only equality but also the relational function **lesser than**. The latter requirement reflects that the macrocode software specifies events to a precision of a 10-microsecond clock, whereas the C/30 can implement a clock no more precise than 100 microseconds.

In order to keep the capability of specifying events as long as 0.65 seconds into the future, we originally wrote the microcode software to maintain an internal clock to 17-bits precision. The disadvantage of this approach is that the C/30 instruction cycle time must be increased from 125 nanoseconds to 135 nanoseconds, concomitant with processor operation involving 20-bit words. Subsequently we devised a scheme which allowed us

to specify an internal clock to 16-bits precision with no detrimental effect on MATNET system performance. Below is the detailed specification of this algorithm.

Let {E} with least-significant-bit (LSB) in units of 10 microseconds represent an event time in the future handed to the microcode software by the macrocode software. It is necessary to be able to specify events as long as 0.65 seconds in the future. The goal is to recognize when the current time matches or just exceeds the event time.

Let the current time {t} be kept as a 16-bit number with LSB in units of 20 microseconds. Each clock tick requires {t} to be incremented by 5, equivalent to 100 microseconds. Note, {t} wraps around in 1.3 seconds, which is a period sufficiently large that the range of positive numbers (and negative numbers as well) is able to describe intervals as large as 0.65 seconds.

When {E} is handed to the microcode software, the following operations are done:

Let {e} = [{E} + 1] / 2

The division operation is a non-signed integer division by two (logical shift right one). The incrementing of {E} before the division is not a rounding operation (which makes no sense, inasmuch as truncation and rounding have the same RMS error when dividing by two), but is a bounding operation to prevent a decrease in value of the representation of the event time through

the loss of the LSB when dividing by two. In operational terms, to prevent the wakeup timer from triggering before the event can happen, we are using the integer upper bound for {e}. The definition of {e} guarantees that it is always a positive number with an integer range of 0-32767 in units of 20 microseconds. Next,

```
If  {e} - {t}  >  0  then continue
               <  0  then let {e} = {e} + sign bit on
```

This step maps future events into positive values of {e}-{t}. Note, {e} cannot equal {t}, because of the assumption that {e} is in the future. Finally,

{ee} = {e} - 1

is stored in a table of events to be checked during interrupt processing. The reason for decrementing {e} is that in the determination of when the event has just happened, this operation maps equality of event time and current time into the range of minus numbers to be checked during interrupt processing with a simple branch-on-minus instruction. Computer processing power is saved if this operation is not done during interrupt processing.

During interrupt processing, the following operation is done:

```
If  {ee} - {t}  >  0   then continue
                =  0   then continue
                <  0   then do event
```

Although the procedure detailed above seems straightforward, implementation and checkout consumed a significant amount of our time due to the complex interrelationship of all the microcode

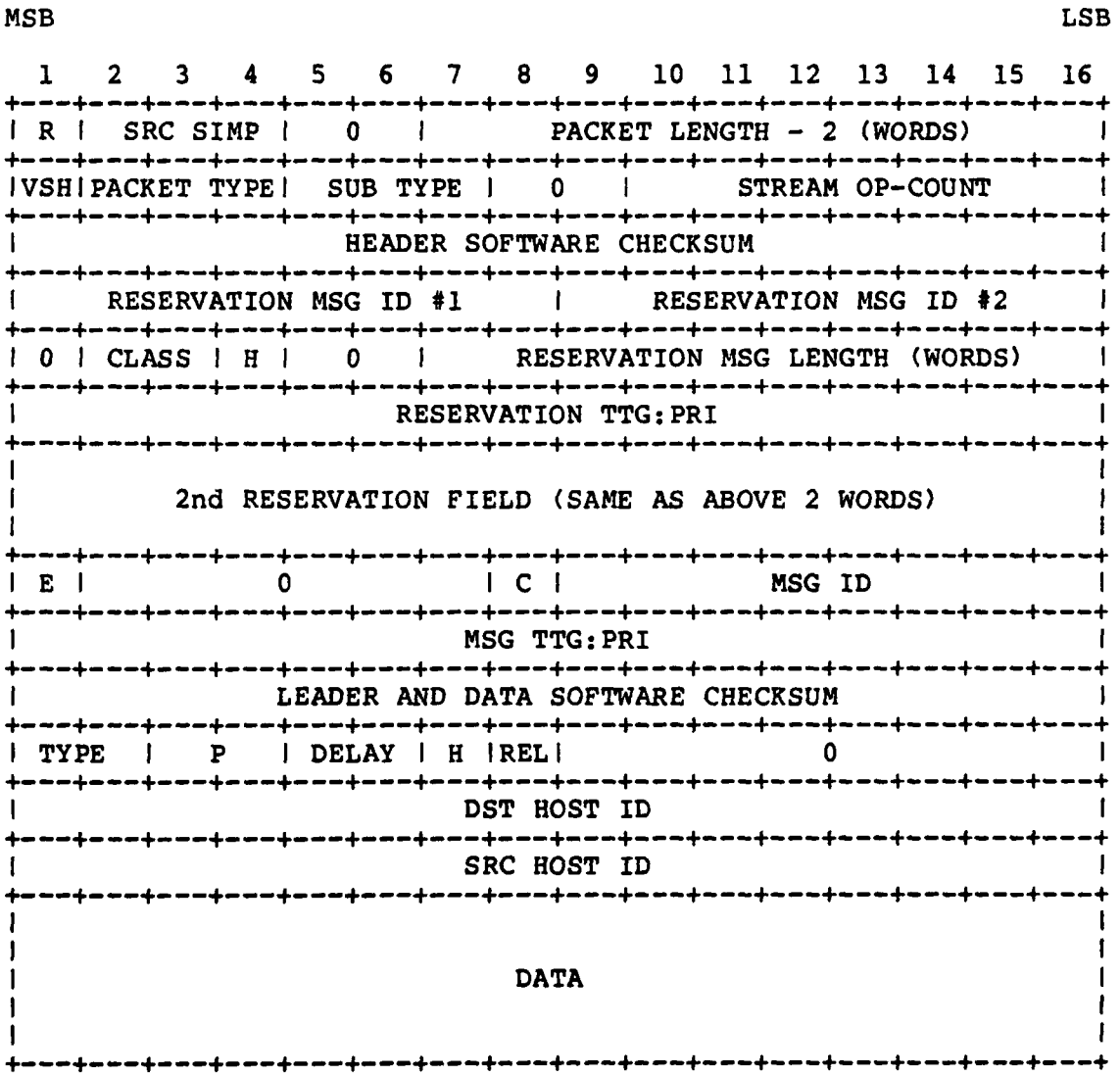
software. Contributing to the difficulty in checkout in the beginning was our failure to recognize the possibility of the wakeup timer causing transmissions to be aborted due to truncation of the event time.

6.3 MATNET Satellite Channel Packet Formats

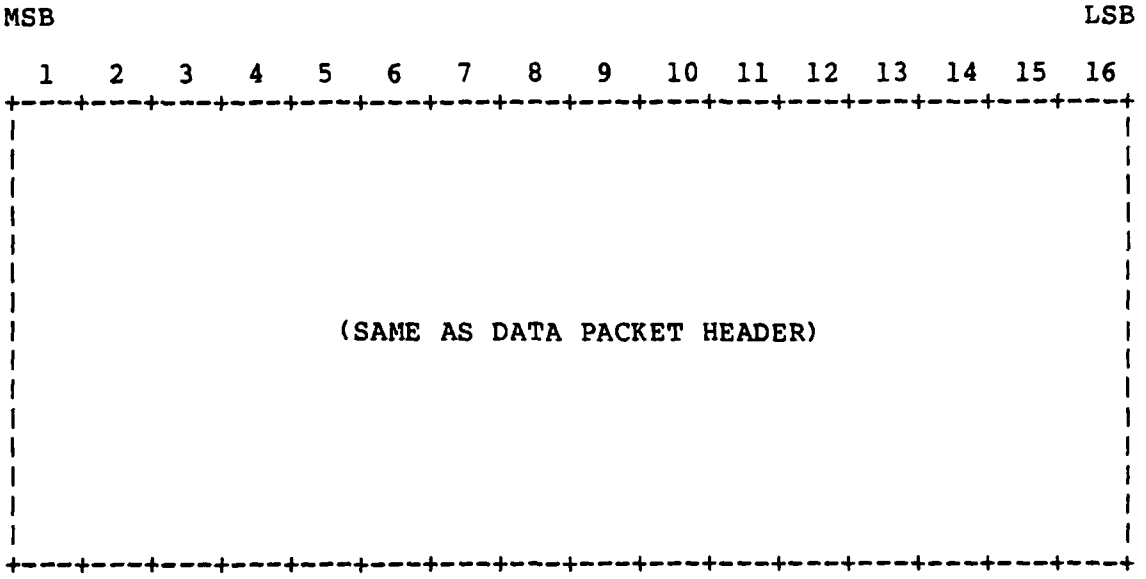
Given the low channel data rate (9.6 Kb/s) and the considerable channel packet overhead in MATNET, one of our concerns is that the available channel bandwidth be efficiently used. (Included in the overhead are the software overhead required by the MATNET PODA protocol, the Internet Protocol, and the Transfer Control Procotol, and the hardware overhead required by the Satellite IMP, the COMSEC equipment, the CODEC, the interleaver/deinterleaver, and the AN/WSC-3 radios.) The dominant mechanism for increasing channel efficiency is the use of longer packets, since all the overheads are independent of packet size. Not to be ignored, however, is the matching of channel traffic with interleaver block size for providing a potentially significant increase in channel efficiency.

Inasmuch as the transmission of only an integral number of interleaver blocks is allowed, if a packet does not fit exactly into a countable number of interleaver blocks, the unused space left over in the last block will be wasted. To avoid unnecessary reduction in channel efficiency, we have matched control traffic

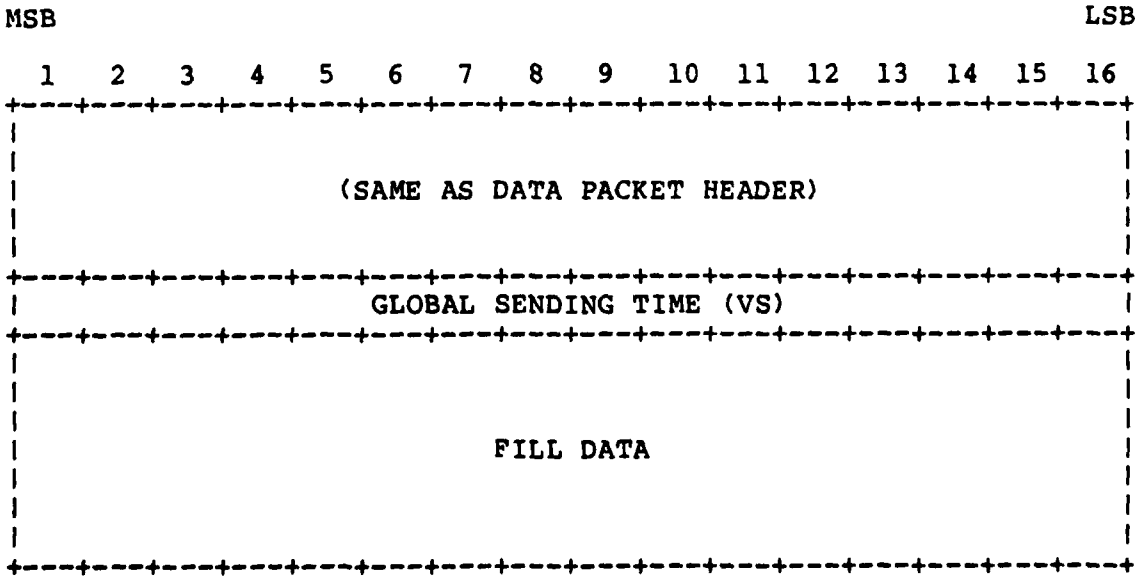
size with interleaver block size. All the channel packet headers defined in SATNET have been modified for MATNET, such that the channel allocation request packet fits exactly into the minimum size possible, namely, one interleaver block; the Hello packet already fits into one interleaver block. Thus, we removed three words in the channel packet headers; two words removed dealt with the operation of the PSP terminal multi-rate capability (a non-implemented feature in MATNET), while the third word provided space for extra packet acknowledgements. An additional benefit of our reworking packet headers is that there is now greater commonality in the header structure of the different control packets and the data packets. Below are displayed the new packet formats, in which packet headers are explicitly depicted.



MATNET Data Packet



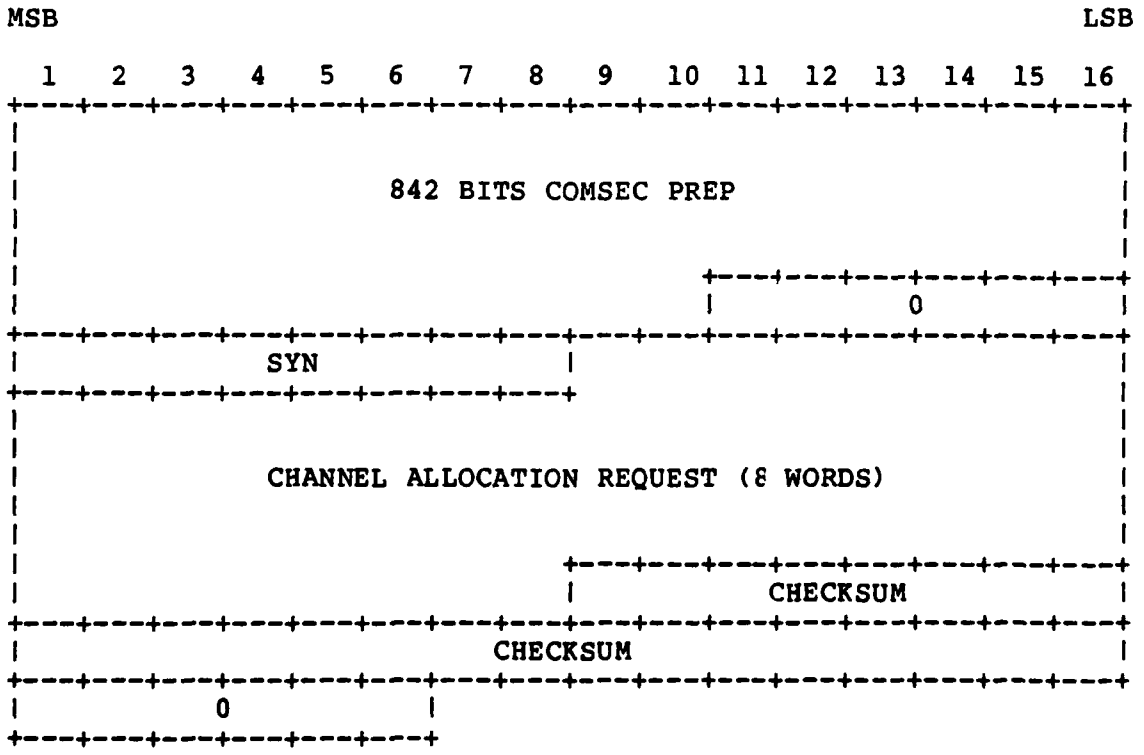
MATNET Channel Allocation Request Packet



MATNET Hello Packet

The transfer of bits from the Red processor to the Black processor for each packet includes additional bits representing

the packet start framing bits, the packet checksum, and fill bits sent during COMSEC prep. This structure is shown below for a channel allocation request packet; data packets have a similar structure.



MATNET Red/Black Bit Transfer Format

7 TCP FOR THE HP3000

7.1 Introduction

This section of the QTR covers the third phase of an ongoing research effort to implement TCP protocols on an HP3000 computer system. Most of our effort during this quarter was dedicated to coding the protocol software. At this point we have completed the coding of the following software modules:

1. The HDH protocol layer (see description below).
2. The 1822 protocol layer.
3. The IP protocol layer. Work includes the 1822 and Internet protocol software.
4. The TCP protocol layer.
5. The interface between the user protocol layers (TELNET and FTP) and the TCP protocol layer. This module was implemented using a pre-release version of the HP3000 operating system.

Testing of these software modules is well underway and will continue into the next quarter. While a number of bugs in the pre-release version of the operating system are causing some problems, these problems were anticipated and our work progress has not been significantly affected.

In addition to our coding effort we have defined a new protocol layer to support an HDLC connection between the HP3000 and an ARPANET TIP. This protocol has been implemented on the HP3000 and is being added to the standard IMP software. The

following sections, which will be included as Appendix J in a forthcoming revision to BBN Report 1822, describe the new protocol in some detail.

7.2 Philosophy of HDLC Host Interface

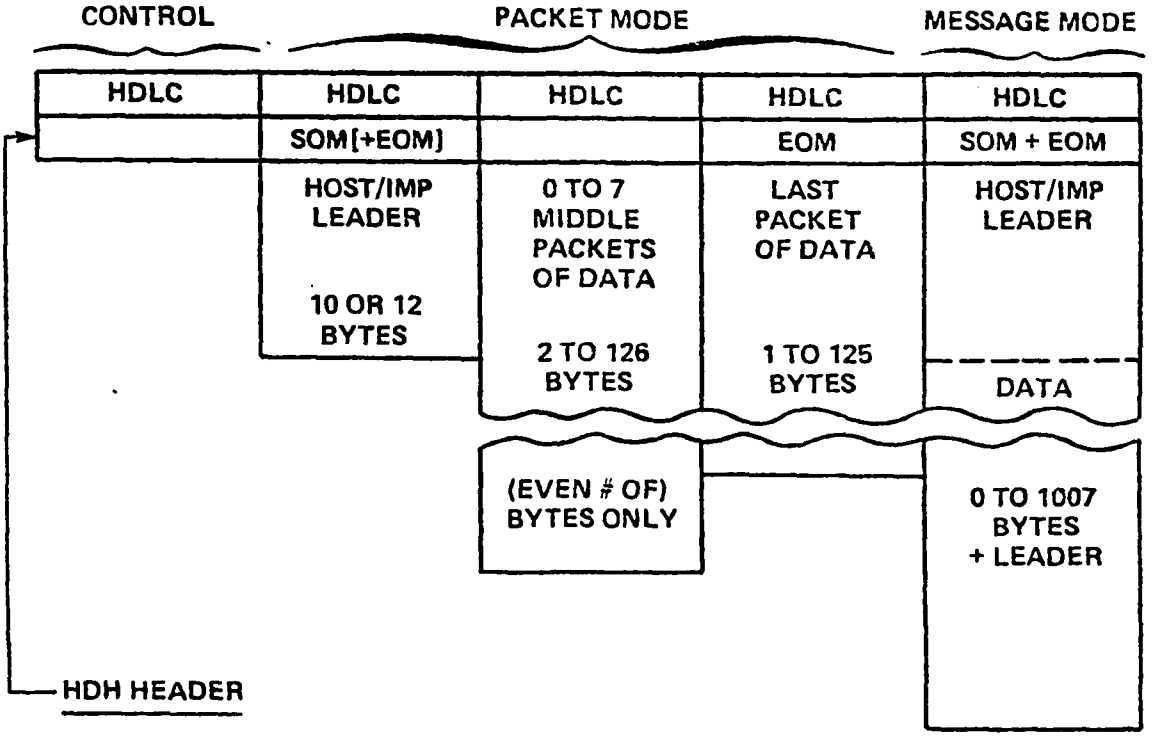
This new kind of host interface (referred to as HDH) is being designed and implemented on C/30 IMPs to support the connection of hosts to the IMP using HDLC at the link level, and the ARPANET host/IMP logical protocol at the network level. This type of access protocol will be general enough to permit any host with an HDLC capability to use the ARPANET host/IMP protocol instead of X.25 level 3. The HDH protocol itself is independent of the data transparency used on the link (bit- or byte-oriented), and of the type and size of the cyclic redundancy checksum (or FCS) although the standard HDH interface will use CCITT HDLC framing and FCS. In the future, the HDH interface may be able to support ADCCP as well as HDLC at the link level.

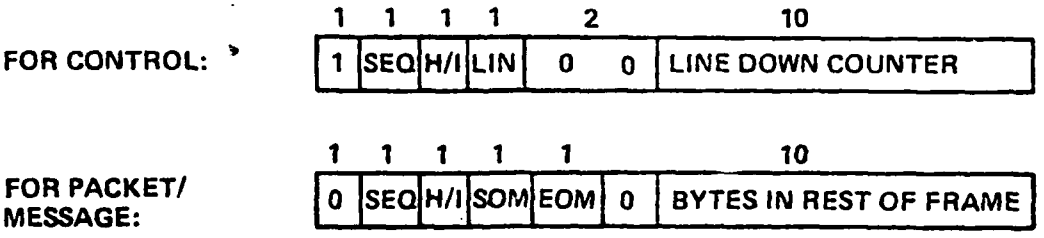
The HDH interface is a close relative of the VDH interface, in that it inserts a reliable transmission protocol underneath the standard ARPANET host/IMP protocol. However, instead of the VDH RTP, it uses HDLC. Each frame also carries in addition to the HDLC header a two-byte HDH header. In order to accommodate both present and future requirements, the HDH protocol will have two modes, shown in Figure 3. One mode is called packet mode and

it requires that the information part of the HDLC frame not exceed 64 16-bit words, and that the host/IMP leader be sent in a separate frame. In this mode, the host explicitly breaks up its messages into packets. The initial implementation of HDH will contain this mode only, because of the nature of the existing H316-like I/O structure. The other mode, called message mode, permits the entire message (up to 1007 data bytes, plus leader) to be sent in a single HDLC frame. This mode will be implemented when the C/30 IMP's I/O structure is revised to provide generalized facilities such as scatter/gather to/from buffers. It will be decided later whether to control the mode for each interface administratively or via negotiations between the host and the IMP (e.g. with NOPs).

7.3 Host/IMP Protocol

The HDH host interface requires the implementation of three layers of protocol: HDLC, HDH, and ARPANET host/IMP. The HDLC protocol is the current CCITT standard using LAP or LAPB modes. For ARPANET HDLC, the parameters are T1>3 seconds, T2 approximately 1 second, N2=20, K=7, and N1=1088 bits (for packet mode) or 8256 bits (for message mode). The values of N1 exceed the requirements of the HDH protocol but are consistent with X.25 level 3. The HDLC protocol does link level flow control, error control, and sequencing. The ARPANET host/IMP protocol is





SEQ = 1 IF SEQUENCE BREAK
H/I = 0 IF HOST ORIGINATED, 1 IF IMP ORIGINATED
LIN = 1 IF LINE IS UP, 0 IF LINE IS DOWN
SOM = 1 IF START OF MESSAGE
EOM = 1 IF END OF MESSAGE

HDH Frame Formats
Figure 3

documented in Section 3 of Report 1822, and is the network-level protocol for exchanging messages across the network between hosts and locally between host and IMP. The HDH protocol is inserted between HDLC and host/IMP and is required at the local host/IMP connection in order to segment messages into packets, rebuild messages from packets, perform line quality monitoring, simulate the host/IMP ready line up/down signalling, and operate the loopback mode. The remainder of this section describes all the aspects of the HDH protocol. Whereas the IMP will implement all of these aspects, there are a few that need not be implemented by the host, and those will be so noted.

The host and IMP send each other frames (via the HDLC level) as illustrated in Figure 3. Each HDLC frame, in addition to the HDLC address and control bytes, contains a 16-bit HDH header. The high order bit of the HDH header distinguishes HDH control frames (to be discussed later) from data frames. For each data frame, there is a bit which, when set, indicates a sequence break in the frame stream between the previous frame and the current one, a bit which indicates if the frame was host or IMP originated, two bits which indicate if the frame is a start of message, end of message, both, or neither, and a count of bytes in the rest of the frame (exclusive of the HDLC and HDH headers).

The sequence break bit is used by the host or IMP to signal to the other side that a discontinuity has occurred in the

message stream. The cause of the discontinuity might be an internal reset within the host or IMP software, or a detected potential data loss such as an HDLC temporary disconnect. Both the host and the IMP should always signal a sequence break when they first start to communicate (e.g. after software restarts, etc.). Since the break might occur in the middle of a message, it is important that the receiver discard any accumulated frames of a message and continue to discard new frames until the next start of message is reached.

The host/IMP bit is used by the IMP to confirm the loopback condition from the IMP to itself. The host should always send its frames with this bit zero, and the IMP with the bit set to one. The host need not implement a loopback mode, but it is recommended that in any case this bit should be examined in order to detect possible inadvertent loopback conditions. The CCITT HDLC specification does not address the issue of a functional loopback condition, and implies looped frames will always be discarded. In practice, however, it is possible to have a functional loopback by exchanging the HDLC addresses A and B on output, and this practice may be observed by various HDLC implementations (as it is in the IMP).

The SOM and EOM bits are used to delimit messages in packet mode. In host/IMP type 0 message of zero length, or in a non-type 0 (host/IMP control, such as RFNM), both bits are turned on

and the leader frame constitutes the entire message. Otherwise, only the SOM bit is turned on in the leader, neither bit is turned on in middle packets, and EOM is turned on in the last packet. In message mode, both SOM and EOM are always turned on, and the leader and data portions of the message are included contiguously in the same frame.

The byte count field indicates the number of valid bytes in the remainder of the frame. This field must always be filled in, and may indicate fewer or the same number of bytes as the physical frame contains. This is to provide for host word sizes that may not always align with the desired packet size. A count larger than the amount in the physical frame is considered a protocol violation and on the IMP receiving side will generate an error in data indication. In packet mode, the byte count of leader frames must be either 10, for non-type 0 messages, or 12, for type 0 messages. The byte count for middle packets can be any even number from 2 to 126, and there may be up to seven middle packets, none of which needs to be maximum length. The reason for the restriction to even byte sizes in middle packets is that the IMP and its serial interface can only address 16-bit words, and cannot concatenate intermediate packets that are not multiples of 16 bits. The byte count for last packets can be any number from 1 to 125. This number must be less than 126 because the IMP on input from the host uses the byte length information to insert the "host padding" bit that signals the end of the

message. At least one byte must be left in the frame to add this padding bit. In message mode, the byte count must be at least 10 (for non-type 0 messages), and may be any number up to and including 1019. Again, the IMP inserts the padding bit on input, and the length must be no more than one byte less than the maximum ARPANET message size to allow for the padding to fit in the HDLC frame. Thus, for HDH hosts, transmitted and received messages may not exceed 8,056 bits plus leader, or a total of 8,152 bits.

HDH control frames, distinguished by having the high-order bit of the HDH header set to 1, are used for measuring the quality of the link and determining its up/down status. HDH control frames contain no additional bytes beyond the HDH header. In addition to the previously described sequence break and host/IMP bits, the HDH header contains a line up/down bit and a line down timer parameter value. The control frames are used by the following algorithm to decide when the connection between the host and the IMP is dead or alive, and to simulate the operation of the host/IMP ready line up/down to the host/IMP level.

Periodically, the IMP sends a control frame to the host. Since the frame is sent by the IMP, the host/IMP bit is one. The host responds immediately to this frame by changing the host/IMP bit to zero and returning the frame to the IMP. These frames may be freely intermixed with data packet frames, including between

packets of the same message.

The IMP expects to receive the echo to its control frame before it goes to send the next control frame or within 2 seconds, whichever is sooner, and if it does not, records a miss. If more than K misses occur within N tries, the IMP declares the line down. No data is sent on the line while it is down.

While a line is down, the IMP continues to send control frames and the host responds to those it receives. The IMP will declare the line up if it receives M responses in a row with no intervening misses. The host must obey the declarations of the IMP, taking the line down or up as instructed by the setting of the line up/down bit in the control frame. In particular, the host should not send data on the line while control frames indicate that the line is down.

While the line is up the host should also maintain a timer which is reset every time the host receives a control frame from the IMP. If the timer ever expires, the host will consider the line down. In this way the host will declare a line down which has broken so completely that control frames are not received at all from the IMP. Since the value of the line down timer depends on the line speed, and since the host may not be aware of the line speed, the reset value for the timer is communicated to the host in the control frames sent by the IMP. The timer is expressed in units of one second and can have reset values

ranging from 3 to 1023.

Typical values for the IMP's parameters in this protocol are K=4, N=20, and M=60. Also, the interval for sending test messages will vary depending on the speed of the line. The interval will range from as much as 6.4 seconds on a 1.2KB line to 640 milliseconds on a 230KB line.

8 TCP-TAC

8.1 Introduction

Work on the TCP TAC has progressed in two areas: 1) a design document has been written describing how the Internet and Transmission Control protocols will be implemented, and 2) the 1822 interface and packet core protocol software has been converted to work with the new data structures.

The existing code that was taken from the H-316 TIP for the 1822 Interface and Packet Core Protocols has been converted to use the new buffering system. This was done in order to have a uniform buffering system in the TAC, instead of the old one for NCP and a new one for TCP. This will permit a uniform interface to the buffers in the parts of the software that are not NCP and TCP specific (i.e., Terminal handling, Network interface, Telnet, etc.). A description of these new data structures for buffering is included later in this section. Substantial work remains to be done to convert the rest of the TIP software to the new buffer system.

The design document includes sections describing Overall Data Flow, Receiving Data, Sending Data, Control and Priority, Data Structures, 1822 Protocol, Internet Protocol, and Transmission Control Protocol. The design document is included as the remainder of this section of the Quarterly Technical

Report.

8.2 Overall Data Flow

A basic premise in the design of TAC is that data should not be moved between buffers, rather the pointers to the data should be passed between program modules. Thus, when a message is read into a buffer, pointers to it are passed between the different protocol modules. When a character is read from the MLC and put into a buffer, the protocol modules manipulate the buffer pointers, not the data itself. This is illustrated in Figure 4.

8.2.1 Receiving Data

To receive data from the network, a message is read from the 1822 host interface into a Message Block (MBLK). If the message will not fit in one MBLK, the remainder will be read into other free MBLKs until the message has been completely read in. All the MBLKs containing the same message will be linked together. A Protocol Data Block (PDB) will be created to point to the MBLKs. The pointers that are passed between the protocol modules will point to the PDBs. More details on the PDBs and MBLKs can be found in the section on "Data Structures".

```
                         +----------+
                         +          +
          +------------- + Message  + <-----------------+
          |+------------ + Buffers  + <----------------+ \
          ||             +          +                   \ \
          VV             +----------+                    \ \
         +---+                                            \ \
    +--- +   + Tumble                                      \ \
    |+-- +   + Table       +-----+    +----+                \ \
    ||   +---+             |     |    |    |                 \ \
    ||                   +>| TCP |<-->| IP |<-+               \ \
    VV          +--------+ | |     |    |    | |   +------+    \ \
  +++++++       |        | | +-----+    +----+ |   |      |   +++++++
  | MLC |<----->| Telnet |<-+                  +-->| 1822 |<->| IMP |
  +++++++       |        | | +-----+             |   |      |   +++++++
    ||          +--------+ | |     |             |   +------+    /\/\
    ||                   +>| NCP |<------------+               / /
    ||   +---+             |     |                            / /
    |+-->+   + Tumble      +-----+                           / /
    +--->+   + Table                                        / /
         +---+                                             / /
           ||           +----------+                      / /
           ||           +          +                     / /
           |+---------> + Message  + -----------------+ /
           +----------> + Buffers  + -------------------+
                        +          +
                        +----------+

            Control ----->                         Data ----->
                                                        ----->
```

Figure 4   Data and Control Flow

8.2.1.1 1822 Module

The 1822 module is given a pointer to a PDB. This module will act directly on the message if it is an 1822 control message (i.e., a RFNM). It will update the appropriate data structure to initiate the action to be taken. If the PDB contains an 1822 data message, it will be passed on to the next protocol module.

The one to which it is passed depends on the Link number in the 1822 message. The Link number is the upper 8 bits of the "Message ID" field in the 1822 leader. The Link numbers to protocol mapping are as follows:

Link #	Protocol
0	NCP Control
2-71	NCP Data
155	Internet

8.2.1.2 NCP Module

The NCP module implements the ARPANET Host-Host Protocol. Its function is essentially identical to the TIP's NCP. The only difference is that it will be modified to work with the new PDB and MBLK data structures. This will be done with a new interface and hopefully will have a small impact on efficiency. When the NCP module is done with a message, it will pass the pointer to the PDB to the Telnet Module.

8.2.1.3 Internet Module

When the Internet Protocol (IP) module gets a pointer to a PDB it first checks the checksum in the IP leader and then checks that the destination address is correct (it should be the address of the TAC running the code). If either of these checks fails, then the datagram is discarded. Also, if the destination address

was incorrect then an IP error report will be sent to the source of the datagram. The next check is whether or not the datagram is fragmented. If so, then the IP module will perform reassembly. This is described in detail in the section on "Internet Protocol". When the IP module gets a complete datagram (either received whole or reassembled) it will pass it on to the next protocol module. Which module it is depends on the "Protocol" field in the Internet header. If a datagram is received for a protocol that is not supported, it will be discarded and an IP error report sent to the datagram source. The protocols supported are as follows:

Protocol #	Protocol Name
3	Gateway to Gateway Protocol
6	Transmission Control Protocol
71	Packet Core
*	Host Measurements

8.2.1.4 TCP Module

When the Transmission Control Protocol (TCP) receives a PDB it first checks the checksum of the message and the validity of the TCP header. If the message passes the check for a valid connection, and its sequence number is in the receiving window, the TCP module will set it up for the open connection. The data will be sequenced if necessary at this point. This is described

* Number is not yet assigned.

in detail in the section on the "Transmission Control Protocol". The next module is then informed that there is data to be processed.

Flow control is implemented by using the TCP Acknowledgement (ACK) and the Window size parameters. A fixed number of characters will be buffered for each connection. As characters are accepted they will be ACKed until the limitation is reached. As the ACK value is advanced, the window will be shrunk correspondingly. When the next module takes data from the message, buffer space becomes available. This causes TCP to advance its window, allowing the distant host to send more data.

Normally the new ACK and window values will be sent out with the next data message from that connection. If nothing is pending for this connection, a message with just the updated ACK and window values will be sent. This is described in more detail in the section "Transmission Control Protocol".

8.2.1.5 Telnet Module

The Telnet module is given a pointer to a PDB when there is data to be processed. This data may be from the NCP or TCP protocol modules. It takes characters out of the MBLKs, looks for Telnet commands, and outputs them to the MLC. This output is done using the existing TIP's "Tumble Tables". This will work

using "OIs", which means that every time an "OI" comes in for a port, Telnet will look to see if there is another character to output.

8.2.2 Sending Data

Data that is sent out to the network normally comes in from the MLC and is received in "Tumble Table" format. This is a block which is filled by the MLC. Its format is one word for each character input. The low order byte of the word is the character and the high order byte is the line number that the character came in on.

When the block is received it is passed to the Telnet module. This module takes the characters out and processes them. As this is happening another block is being filled by the MLC.

8.2.2.1 Telnet Module

When the Telnet Module gets a character for a port, it first checks if there is an open connection for that port. If not, it discards the character and outputs a bell character to the port. Next, it checks to see if there is room in the MBLK for another character. If not, then the character is discarded and the bell rung. If there is room, the character is put into the MBLK and

the proper pointers are advanced. Telnet then indicates to the next protocol module that there is data to send. Depending on which protocol is being used for the connection, this is either NCP or TCP.

8.2.2.2 TCP Module

When the TCP module gets a signal that there is new data that should be sent, it first checks if there is room in the sending window to send more data. This done by checking if the last sent but unacknowledged data is at the right edge of the sending window. If there is no room, then nothing will be sent. Otherwise, the TCP module will adjust the pointers in the PDB and MBLKs to point to the correct data and update the TCP header.

Flow control in the sending direction is done by maintaining three pointers in the PDB. These are pointers to data in the MBLKs. They are pointers to the last ACKed character, the last sent but unACKed character, and the last not-yet-sent character in the MBLK. As data is ACKed, sent, or put into the MBLK, the appropriate pointer is advanced.

When the TCP module is ready to send the data, it checks to see if the 1822 module can send the data (i.e., there are not more than eight outstanding messages). If the data can be sent, then the TCP module will compute a checksum for the message and

pass a PDB pointer to the IP module.

8.2.2.3 Internet Module

When the IP module gets a pointer to a PDB it first checks to see if it knows where to send the datagram. If the destination is on the same network as the TAC, the Internet module will use that as the address. If the destination is on a different network, then it will send it to a gateway. The procedure to decide which gateway to use is discussed in more detail in the section "Internet Protocol".

The IP module will then build an IP leader in the MBLK and compute the checksum of the leader. It will then pass a pointer to the message to the 1822 module.

8.2.2.4 1822 Module

When the 1822 module gets a pointer to a PDB, it will always send the message it contains to the destination specified in the PDB. The destination host will either be a server host or a gateway. The 1822 module will keep track of the number of outstanding (no RFNMs received) messages sent to a host. This will be used by the NCP and TCP protocol modules to insure that the IMP will never block the TAC's host interface due to having more than eight outstanding messages.

When the 1822 module sends the message, it will build an 1822 leader in the MBLK. It will then send the message to the IMP via the host interface hardware.

8.3 Control and Priority

The code in the TAC will run either at the interrupt level or at the background loop. The interrupt routines will support the host interface, MLC, and clock. In addition, high priority protocol routines will run at the task interrupt level.

The background loop will contain most of the TAC code. The protocol modules will run here. They will be executed in the following order: 1822 Input, IP Input, TCP Input, NCP Input, Telnet Input, Telnet Output, NCP Output, TCP Output, IP Output, and 1822 Output.

Each protocol module will have an input queue. When it runs, it checks for an entry on its queue. If it finds something, it takes it off the queue and processes it. Some of the protocol modules will be written to process all entries on their queue before exiting; others will process one entry and then exit. The NCP, TCP, and Telnet modules will process one entry. The 1822 and IP modules will process all entries.

8.4 Data Structures

A new system of buffers will be used in the TAC. It consists of two types of blocks, the Message Block (MBLK) and Protocol Data Block (PDB). These are used both for receiving and transmitting messages and for buffering characters on input and output.

The structure of these buffers is such that when a protocol module is passed a message it is given a pointer to a PDB. The PDB includes a link to the first MBLK. The main function of the PDB is to save frequently accessed things in the message and to point to the message. The MBLKs contain the actual message. They also have fields to facilitate reassembly and sequencing.

8.4.1 Message Block

The main function of the Message Block (MBLK) is to hold messages. It will be used for all protocols. If a message will not fit in one MBLK, then the remainder will be put into a second MBLK. The second will be linked to the first. The length of the MBLK will be either 30 or 60 words. The 30 word MBLK is used for sending data and the 60 word MBLK is used for receiving.

The header of the MBLK consists of 4 words. See Figure 5 for the format of the block. The "Link" is used to point to other MBLKs. The "Offset" field is used for reassembling IP

fragments and sequencing TCP data. During these operations it contains the offset of where this data is relative to the data in the previous MBLK. Zero means that there is no missing data. This is discussed in detail in the section on "Reassembly".

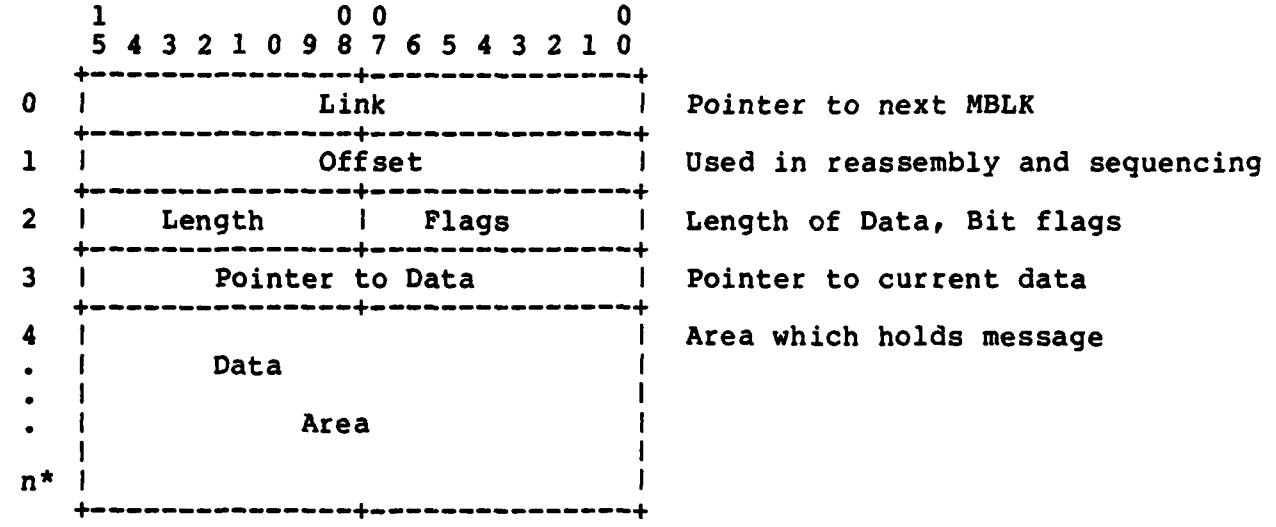


Figure 5 Message Block Format

The "Length" and "Pointer to Data" fields are used to indicate where and how much data is in the MBLK. The meaning of these is always relative to the protocol module currently processing the message. For example: when a message is read in from the host interface the "Pointer to Data" will point to the 1822 leader and the "Length" will be the length of all the data

---

* Where "n" is either 29 or 59, depending whether the block is used for sending or receiving.

in this MBLK. When the 1822 module is ready to pass the data to the next protocol module, it adjusts these fields to refer to the data after the 1822 leader. In this way, a protocol module need not know what, if any, protocol preceded it.

The "Flags" is a bit field containing such things as End of message, I/O in progress, Read or Write, small or large block, etc. The "Data Area" is where the actual message is stored. The small size MBLK (30 words) is sized to contain an 1822, IP, and TCP leader, but no data. The large size (60 words) can contain the leaders plus up to 60 bytes of data.

8.4.2 Protocol Data Block

The Protocol Data Block is a header block for one or more MBLKs that make up a message. It contains pointers to the first MBLK, pointers to specific leaders in the MBLKs, frequently accessed items from the message, and a link to the next PDB.

As previously stated, it is pointers to PDBs that are passed between protocol modules. When a protocol module gets a PDB, it expects to find one PDB, which points to one or more MBLKs. The data in the MBLKs is expected to be in sequence and non-fragmented. This requires that each protocol module insure that the data it passes to the next module be contiguous. This is best described with the following example:

When the Internet module gets two fragments of the same datagram, it needs to reassemble them before it can pass them to the next protocol module. What it does is to take the MBLKs containing the second fragment and link them into the proper places in the list of MBLKs of the first fragment. As it does this, it adjusts the fields in the MBLKs to point to the correct data. When it has linked in all the MBLKs from the second fragment, it puts the PDB, which controlled the second fragment, back on the free list of PDBs.

The TCP module performs a similar operation to sequence the data before it passes it to the Telnet module. The format of the PDB is shown in Figure 6.

The first field in the PDB, "Link to next PDB", is a pointer to another PDB. This is used for reassembly and sequencing. The next field in the PDB is the address field. This is either the source of the message if it was received or the destination if it is to be sent. The "Identification" field is the internet identification which is used in assembling internet fragments. The "Flags" field is a bit array used for things like datagram complete, EOL, Urgent, Read or Write, block free, in use, hole, etc. The "Protocol" field is the host-to-host protocol the message is for. The "Time Stamp" field is used for timing out messages.

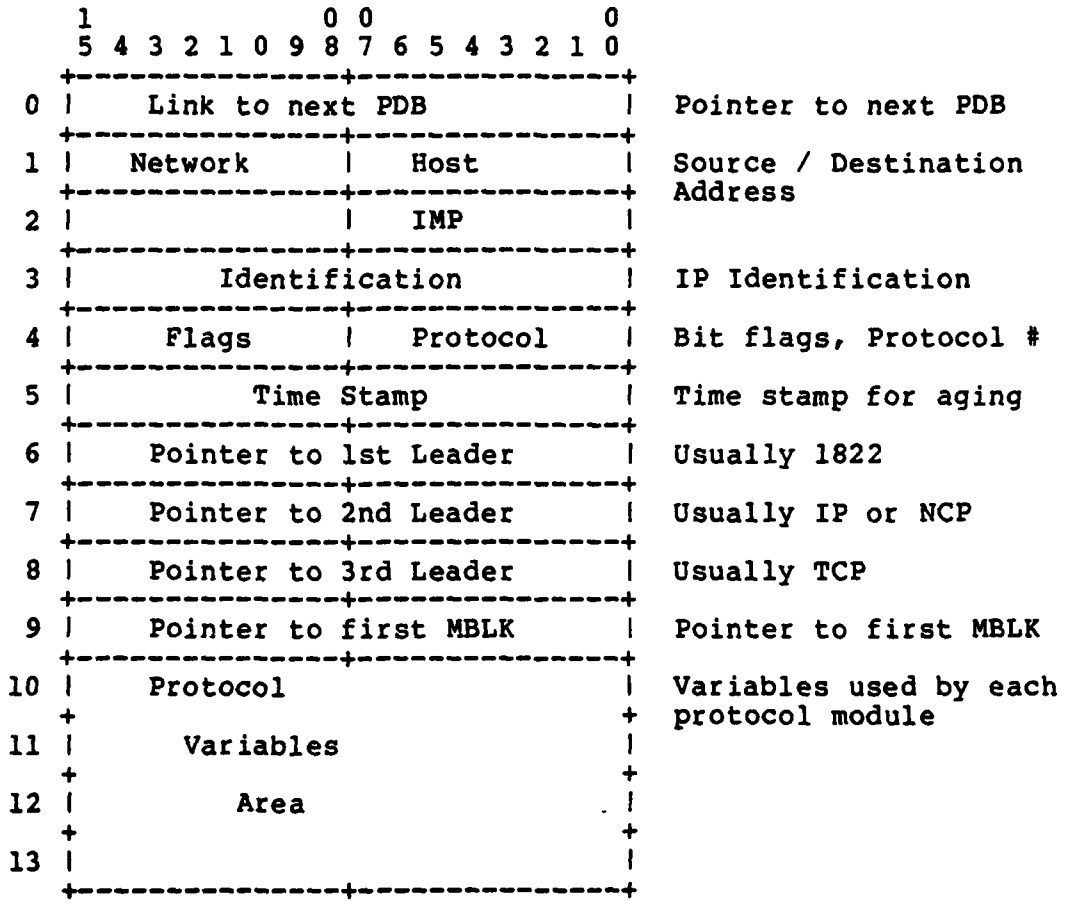


Figure 6 Protocol Data Block Format

The "Pointer Leader" fields are used to point to different leaders in the MBLKs. This is done to make it easier to find a particular leader in the message. They are set up by a particular protocol message and refer to different leaders depending on which protocols are in use.

The "Pointer to first MBLK" field is the pointer to the first MBLK of the message. The "Protocol Variables Area" is a temporary area that any protocol module can use while it is processing the PDB. As long as it controls the PDB, no other module will change these fields.

8.5 1822 Protocol

All 1822 data messages will be passed directly to the next protocol module. When the 1822 module gets a control message it will call a routine supplied to it by the next protocol module. For example, the IP module will supply a routine to be called when the 1822 module receives a "RFNM" on an Internet link number. The routines will be called with a pointer to the PDB of the message. When the routine returns the 1822 module will discard the message.

8.6 Internet Protocol

8.6.1 Identifier Assignment

When the Internet module gets a message to send, it generates a value for the "Identifier" (ID) field in the internet header. It does this by keeping a 16-bit counter called the ID counter. When it needs a new value it increments the counter by one and uses the result. The ID counter will not be initialized

when the TAC is reloaded or restarted to insure that the values are sequential.

8.6.2 Option Support

The Internet module will only actively support the "Error Report" IP option. None of the other currently defined options will require any action to be performed by the TAC.

8.6.3 Reassembly

When the Internet module gets a PDB which is a datagram fragment, it must reassemble it. It first looks at the fragments on the Internet-Reassembly queue to see if there are any other fragments of the same datagram. It does this by comparing the source address, ID, and protocol number of the two fragments. If it does not find a match, it adds the new PDB to the queue. At this point, it also puts a time stamp in the PDB. This will be used to timeout unassembled fragments.

The actual reassembly process consists of adding the MBLKs of the new datagram to the list of MBLKs of the datagram on the queue. This is done using the "Offset", "Length" and the "Pointer to Data" fields in the MBLK. At this point in the processing of the fragment, the fragment consists of one or more MBLKs linked together. The IP header will always be in the first

MBLK. The "Offset" fields in the MBLKs are all zero (there is no missing data in the new fragment itself). The "Length" fields contain the length of the IP data (not including the IP header) in each MBLK. The "Pointer to Data" fields point to the IP data in each MBLK.

The MBLKs of the new datagram are added to the datagram on the queue by comparing the "Fragment Offset" in the IP header of the new datagram to the "Fragment Offset" in the IP header of the datagram on the reassembly queue. This is done by taking the first MBLK on the list and adding the "Fragment Offset" from the IP header to the "Length" and "Offset" fields in the first MBLK. If this sum is greater than the "Fragment Offset" in the IP header of the new datagram, then the MBLKs of the new datagram should go before the first MBLK of the datagram in the original fragment. If not, then they should be added in after. In this case, the comparative process is repeated with the rest of the MBLKs on the list. When the proper place is found, the new MBLKs are linked in and the fields of the new and old MBLKs are adjusted. If the new fragment overlaps the existing, then by adjusting the "Pointer to Data" and "Length" fields, the overlap can be skipped. This may result in one or more MBLKs being discarded.

When the datagram is completely reassembled, it can then be taken off the Reassembly queue and passed to the next protocol

module.

8.6.4 Routing

The decision about where to send a message is twofold. If the destination host is on the same net as the TAC, then the message is sent directly to that host. If not, then it must be sent to a gateway.

The Internet module will maintain a table that will facilitate routing messages to hosts on other networks. The table is a list of all networks (256) and the gateways to get to the networks. This table, called the Network-Gateway table, will be initially loaded into the TAC and will be dynamically maintained by the TAC.

When the Internet module needs to find an address, it looks in the Network-Gateway table to get the gateway address for the network it wants to send to. If it finds one it uses it. If the entry for the network it wants is empty (i.e., no gateway address specified), then the Internet module will use an arbitrary gateway.

If the Internet module receives a Redirect message from a gateway, it will update the Network-Gateway table to indicate the correct gateway. This will insure that the Network-Gateway table contains current information.

If a gateway goes down during a connection the Internet module will clear that network's entry in the Network-Gateway table. It will then try an arbitrary gateway. If at a later time, the Internet module receives a Redirect message telling it to use a new gateway to get to a network, it will set that network's entry in Network-Gateway table to the new gateway's address.

8.6.5 Gateway to Gateway Messages

The Internet module will support the "Gateway to Gateway" protocol in a passive sense. If it receives a "Destination Unreachable Packet" or a "Redirect Packet" it will take appropriate action. If it receives anything else, it will discard the message. In particular, if the Internet module receives a "Source Quench Packet" it will discard the message. This strategy is used due to the TAC's limited amount of buffering. The buffers would soon fill up because of the data not being acknowledged (in TCP). This will effectively limit the transmission.

8.6.6 Timeouts

Internet fragments will be discarded if they are not reassembled within 60 seconds. When a new fragment is

reassembled into an existing one, the "Timeout" field in PDB of the existing fragment will be updated with the current time. This will, in effect, reset the timer for that fragment.

8.7 Transmission Control Protocol

8.7.1 Connection Opening and Closing

The TCP module will have a finite state machine which will be used for establishing and closing connections. The procedure to open a connection is to pass the required information (Destination address, socket, etc.) to the TCP module. It will then run the finite state machine, which will set up the required data structures and open the connection. Closing the connection will be done in a similar way. The states of the finite state machine will be similar to what is described in "IEN-129, DOD Standard Transmission Control Protocol".

All TCP Port numbers assigned by the TAC will consist of the upper 8 bits set to the terminal number for the connection (1-64.) and the lower 8 bits set to 23. All connections made to or from the TAC will use this format.

8.7.2 Initial Sequence Number Assignment

The TCP module will maintain a 32-bit counter that will be used to generate Initial Sequence Numbers (ISN). The counter will be incremented by a constant value every time the H-316 clock ticks, which is every 25.6MS. The counter will be incremented by 64. This will then wrap around approximately every 4.55 hours. When the TCP module needs an ISN it reads the counter and gets a value.

8.7.3 Option Support

The TCP module will understand the format of all TCP options. It will support the "No-Operation" and "End of Option List" options. It will not support the "Buffer Size" option. If it receives a "Buffer Size" option with anything greater than size one, it will not accept the connection but will reset it.

8.7.4 Urgent Data

When the TCP module receives a message with a valid Urgent Pointer, it sets a bit in the "Flag" word in the PDB and saves the offset to the end of the urgent data. When the next protocol module takes data out of the MBLK it will get an indication that the data it is getting is urgent.

Likewise, when the TCP module is given data to send, the protocol module supplying the data can include an indication that the data is urgent. The TCP module will include this information in all messages it sends until the urgent data is sent.

8.7.5 End of Letter Handling

The TCP module will not do anything special when it receives a message with End of Letter (EOL) set. The TCP module always presents all data to the next protocol module as soon as it is available. Consequently, no special handling is necessary.

The TCP module will accept data to be sent with an EOL indication. It will send this data with EOL set in the message. No additional data will be sent in the message. If the data is required to be retransmitted, it will be transmitted with EOL preserved.

8.7.6 Retransmissions

Data that is transmitted by the TCP module will be held until it has been ACKed by the remote host. If an ACK is not received for the data within three seconds it will be retransmitted. All data in the buffer that is not ACKed will be retransmitted (except if EOL is set; see previous section). If the data is still not ACKed for another seven seconds,

retransmission will occur again. This procedure will continue using the series 3,7,15,15,30 . If there is still no ACK, then the user will be notified. The retransmissions will continue every 30 seconds until either the user closes the connection or the TCP module receives an ACK.

8.7.7 Acknowledgement and Window Strategy

The Acknowledgement (ACK) and Window parameters are used to control how much data the remote host can send to the TAC. The TCP module will ACK data up to the limit it is willing to buffer for a connection. Data received after this limit is reached will be discarded. As the data is received, but before the next protocol module takes it out of the buffer, the window will be closed by the amount received. When the buffer limit is reached, the TCP module will not ACK any new data and will be advertising a zero window. When the next protocol module takes data out of the buffer, the window will be opened. This will allow the remote host to send more data.

When data is sent on the connection, the current ACK and Window values are always included in the same message. In the case where no data is being sent and the ACK and/or Window values have changed, a different strategy is used. When the ACK pointer is advanced, the TCP module will wait one second to see if there is data to send. If there has been no data sent for one second,

then the ACK will be sent without data. A new window value will not be sent until the buffer is at least half empty. This strategy is designed to insure that the remote host sends blocks of data and to eliminate unnecessary retransmissions.

8.7.8 Sequencing

When the TCP module gets a data message for a connection, it must insure that the data is sequenced before it passes the data to the next protocol module. The sequencing is done by comparing the sequence number of the message to the current "Left Window Edge" (LWE) and manipulation of the "offset", "length", and "pointer to data" fields of the MBLKs that make up the message. For each connection a list is maintained of MBLKs that are being sequenced. The MBLKs are in order but there may be missing data.

At the point when the TCP module is ready to sequence the data, the message consists of a PDB, followed by one or more MBLKs linked together. The "pointer to data" field points to the actual TCP data. The "offset" fields are all zero because the data in the message is sequential relative to itself. The "length" field is the amount of data in each MBLK.

The sequencing is done by linking the MBLKs of the new message into the sequencing list for the connection for which the data message is intended. This is done via the following steps:

1. Subtract the LWE from the Sequence number of the message. The result is the offset from the LWE. Then set the "offset" field of the first MBLK to this result. If it is zero, this means that there is no missing data. Otherwise, the result is the amount of missing data.

2. Add the MBLKs to the existing list. If there are no MBLKs on the list, then the new MBLKs become the list. Otherwise, the insertion is done in the following steps:

    a) Find the position in which to add the new MBLK by adding together the "offset" and "length" of first MBLK in the list. If "offset" of new MBLK is less than the sum, then the new MBLK goes before the MBLK in the list. Otherwise, add "offset" and "length" of the next MBLK in the list to the previous sum. Repeat this procedure until a fit is found or the end of list.

    b) Link the new MBLK into the list.

    c) Subtract the ("offset" + "length") of the previous MBLK from "offset" of the new MBLK.

    d) Subtract the ("offset" + "length") of the new MBLK from "offset" of the next MBLK.

Overlapping data will be handled by adjusting the "length" field in the MBLKs as the new MBLKs are linked in.

The result of these operations is a list of MBLKs. The "offset" field in the MBLKs contains the number of missing data bytes before it. When the MBLK is at the front of the list, a zero "offset" means the data is sequenced and can be passed to the next protocol module. As data is taken by the next protocol module, the "length" field of the first MBLK should be decremented. When it is zero, the block is empty and can be discarded.

9 TCP FOR VAX-UNIX

9.1 Introduction

The purpose of this section of the Quarterly Technical Report is to summarize the progress of the VAX-UNIX Networking Support Project. The overall purpose of this project is to provide the capability for the VAX to communicate with other computers via packet-switching networks such as the ARPANET. Specifically, the project centers around an implementation of the DoD standard host-host protocol, the Transmission Control Protocol (TCP) [1]. TCP allows communication with ARPANET hosts, as well as hosts on networks outside the ARPANET, by its use of the DoD standard Internet Protocol (IP) [2]. The implementation is designed for the VAX, running VM/UNIX, the modified version of UNIX 32/V developed at the University of California, Berkeley [3]. This version of UNIX includes virtual paging capabilities.

In the following paragraphs we will discuss some features and design goals of the implementation, and its organization.

9.2 Features of the Implementation

9.2.1 Protocol-Dependent Features

9.2.1.1 Separation of Protocol Layers

The TCP software that we are developing for the VAX incorporates several important features. First, the implementation provides for separation of the various protocol layers so that they can be accessed independently by various applications*. Thus, there is a capability for access to the TCP level, which will provide complete, reliable, multiplexed, host-host communications connections. In addition, the IP level is also accessible for applications other than TCP, which require its internet addressing and data fragmentation/reassembly services. Finally, the implementation also allows independent access to the local network interface (in this case, to the ARPANET, whose host interface is defined in BBN Report No. 1822 [4]) in a "raw" fashion, for that software which wishes to communicate with hosts on the local network and do its own higher level protocol processing.

---

* In this context, the terms application and user refer to any software that is a user of lower level networking services. Thus, programs such as FTP and TELNET can be considered applications when viewed from the TCP level, and TCP itself may be viewed as an application from the IP level.

9.2.1.2 Protocol Functions

Another feature of the implementation is to provide the full functionality of each level of protocol (TCP and IP), as described in their specifications [1,2]. Thus, on the TCP level, features such as the flow control mechanism (windows), precedence and security levels, and "rubber EOL" option processing will be supported. On the IP level, datagram fragmentation and reassembly will be supported, as well as IP option processing, gateway-host flow control (source-quenching) and routing updates. However, it is anticipated that some of these features (such as handling IP gateway-host routing updates and IP option processing) will be implemented in later stages of development, after more basic features (such as TCP flow control and IP fragmentation/reassembly) are debugged.

9.2.2 Operating System-Dependent Features

9.2.2.1 Kernel-Resident Networking Software

There are several features of the implementation which are operating system dependent. The most important of these is the fact that the networking software is being implemented in the UNIX kernel as a permanently resident system process, rather than a swappable user level process.

This organization has several implications which bear on performance. The most obvious effect is that since the networking software is always resident, it can more efficiently respond to network and user initiated events, as it is always available to service such events and need not be swapped in. In addition, residence in the kernel removes the burden of the use of potentially inefficient interprocess communication mechanisms, such as pipes and ports, since simpler data structures, such as globally available queues, can be used to transmit data between the network and user processes. Kernel provided services (e.g., timers and memory allocation) also become much easier and more efficient to use.

The large address space of the VAX makes this organization practical and allows the avoidance of expedients like the NCP split kernel/user process implementation that have been necessary in previous UNIX networking software implementations on machines with limited address space, like the PDP 11/70. It is hoped that the kernel-resident approach will contribute to the speed and efficiency of this TCP.

9.2.2.2 User Interface

Use of the "traditional" UNIX file-oriented user interface is another operating system-dependent feature of this implementation. The user will access the network software by

means of standard system file I/O calls: open, close, read, and write. This entails modification of certain of these calls to accommodate the extra information needed to open and maintain a connection. In addition, the communication of exceptional conditions to the user (such as the foreign host going down) must also be accommodated by extension of the standard system calls. In the case of open, for example, use of the call's mode field will be extended to accommodate a pointer to a parameter structure. In the case of exceptional conditions, the return code for reads and writes will be used to signal the presence of exceptional conditions, much like an error. An additional status call will be provided for the user to determine detailed information about the nature of the condition.

In this way, the necessary additional information needed to maintain network communications will be supported, while still allowing the use of the functionality that the UNIX file interface provides, such as the pipe mechanism.

In the initial versions, this interface will be the standard UNIX blocking I/O mechanism. Thus, outstanding reads for data which has not been accepted from the foreign host, and writes which exceed the buffering resources of a connection, will block. It is expected that when and if non-blocking I/O modifications to the VM/UNIX kernel are introduced, such as the Version 6 await/capacity mechanism, this restriction will be lifted.

9.3 Design Goals

Several design goals have been formulated for this implementation. Among these goals are efficiency and low operating system overhead, promoted by a kernel-resident network process, which allows for reduced process and interprocess communication overhead.

Another goal of the implementation is to reduce the amount of extraneous data movement (copying) in handling network traffic. To achieve this, a buffer data structure has been adopted which has the following characteristics: intermediate size (128 bytes); low overhead (6 bytes of control information per buffer); and flexibility in data handling through the use of data offset and length fields, which reduce the amount of data copying required for operations like IP fragment reassembly and TCP sequence space manipulations.

The use of queueing between the various software levels has been limited in the implementation by processing incoming network data to the highest level possible as soon as possible. Thus, an unfragmented message coming from the network is passed to the IP and TCP levels, with queueing taking place at the device driver only until the message has been fully read from the network. Similarly, on the output side, data transmission is only attempted when the software is reasonably certain that the data will be accepted by the network.

Finally, it is planned that the inclusion of the network software will entail relatively little modification of the basic kernel code beyond that provided by Berkeley. The only modifications to kernel code outside the network software will be slight changes to the file I/O system calls to support the user interface described above. In addition, an extension to the virtual page map data structure in low core will be necessary to support the memory allocation scheme, which makes use of the kernel's page frame allocation mechanisms.

9.4 Organization

9.4.1 Control Flow

The network software can be viewed as a kernel-resident system process, much like the scheduler and page daemon of Berkeley VM/UNIX. This process is initiated as part of network initialization. Its main flow of control is an input loop which is activated (via wakeup) by the network interface device driver when an incoming message has been completely read from the network. (It can also be awakened by a TCP user or timer events, as described below.) The message is then taken from an input queue and dispatched on the basis of local network format (e.g., 1822 leader link number). ARPANET IMP-host messages (RFNMs, incompletes, IMP/host status) are handled at this level. For other types of messages, the local network level input handler

calls higher level "message handlers." The "standard message handler" is the IP input routine. However, handlers could be included for other protocols at this level (such as UNIX NCP), or for a "raw message" service.

At the IP level, the fragment reassembly algorithm is executed. Unfragmented messages with valid IP leaders are passed to the higher level protocol handler in a manner similar to the lower level dispatch, but on the basis of IP protocol number. The "standard handler" is TCP. Another protocol handler interprets IP gateway-host flow control and routing update messages. Fragmented messages are placed on a fragment reassembly queue, where *incoming fragments* are separated by connection and IP identification. The reassembly software handles fragment overlaps and duplications. A timer is associated with this queue, and incomplete messages which remain after timeout are dropped and their storage is freed. Completed messages are passed to the next level.

At the TCP level, incoming datagrams are processed via calls to a "TCP machine." This is the TCP itself, which is organized as a finite state machine whose states are the various states of the protocol as defined in [1], and whose inputs include incoming data from the network, user open/close/read/write requests, and timer events*. Input from the network is handled directly,

* An interesting characteristic of the TCP machine is that it is generated from a formal specification, described in [5], using

passing through the above-described levels. User requests and timer events are handled through a work queue.

When a user process executes a network request via system call, the relevant data (on a read or write) is copied from user to kernel space (or vice versa), a work entry is enqueued, and the network process is awakened. Similarly, when timers associated with TCP (such as the retransmission timer) go off, timer requests are enqueued and the network input process is awakened. Once awakened, it checks for the presence of completed messages from the network interface and processes them, as described above. After these inputs are processed, the TCP machine is called to handle any outstanding requests on the work queue. The network process then sleeps, waiting for more network input or work requests. Thus, the TCP machine may be called directly with network input, or awakened indirectly to check its work queue for user and timer requests.

On the output side, TCP requests for data transmission result in calls to the IP level output routine. This routine does fragmentation, if necessary, and makes calls on the local network output routine. Outgoing messages are then placed on a hardware buffering queue, for transmission to the network interface by the device driver. In data transmission, an attempt is made to ensure that data moving from the highest level (TCP), finite state parser generator techniques.

will not be sent unless there is reasonable certainty that the lower levels will have the necessary resources to accept the message for transmission to the network. TCP messages are also placed on a retransmission queue and resent at timed intervals until they are acknowledged by the foreign TCP.

9.4.2 Buffering Strategy

As mentioned earlier, all data is passed from the network to the various protocol software layers in intermediate sized buffers. The allocation of these buffers is handled by the network software. Buffers are obtained by "stealing" page frames from the kernel's free memory map (CMAP). In VM/UNIX, these page frames are 1024 bytes long, and thus have room for eight 128 byte buffers. The advantage of using kernel paging memory as a source of network buffers is that their allocation can be done totally dynamically, with little effect on the operation of the overall system. Buffers are allocated from a cache of free page frames, maintained on a circular free list by the network memory allocator. As the demand for buffers increases, new page frames are stolen from the paging freelist and added to the network buffer cache. Similarly, as the need for pages decrease, free pages are returned to the system. To minimize fragmentation in buffer allocation within the page frames, the free list is sorted.

The number of pages that can be stolen from the system is limited to a moderate number (in practice 128-256). To enforce fairness of network resource utilization between connections, the number of buffers that can be dedicated to a particular connection at any time is limited. This limit can be varied to some small degree by the user when a connection is opened. Thus, a TELNET user may open a connection with the minimum 1K bytes of send and receive buffering; while an FTP user, anticipating larger transfers, might desire up to 4K of buffering. The effect of this connection buffering allocation is to place a limit on the amount of data that the TCP may accept from the user for sending before blocking, and the amount of input from the network that the TCP may acknowledge. Note that in receiving, the network software may allocate available buffers beyond the user's connection limit for incoming data. However, this data is considered volatile, and may be dropped when buffer demands go higher. Incoming data is acknowledged by TCP only until the user's connection buffer limit is exhausted. The advertised TCP flow control window for a connection is set on the basis of the remaining amount of this buffering.

Thus, the network software must insure that it has enough buffering for 1) its own internal use in processing data on the IP and local network levels; 2) retaining acknowledged TCP data that have not been copied to user space; and 3) retaining data accepted by the TCP for transmission which have not yet been

acknowledged by the foreign host TCP. Other data, such as unacknowledged TCP input from the network and fragments on the IP reassembly queue, are vulnerable to being dropped when demand for more buffers makes necessary the recycling of buffers on these queues. Since there is an absolute limit on the number of page frames that may be stolen from the paging system, and hence the total number of buffers available, there is a resultant limit on the total number of simultaneous connections.

Several data structures are required for stealing page frames from the kernel and maintaining the buffer free list. These include enough page table entries for mapping the maximum number of page frames which can be stolen from the system, an allocation map for allocating these page table entries, and the free page list itself. For a 256 page maximum, this requires 2K bytes of page tables, 1K bytes for page frame allocation mapping, and another 1K bytes for the network freelist. The maximum page parameter and others, including the minimum and maximum amount of buffering that the user may specify, are modifiable constants of the implementation.

9.5 Current Status

The design effort for this project commenced in September 1980. At this writing (early December), major design of all modules has been substantially completed and coding has begun.

Most of the code for the IP, local network, and storage allocation is finished. Work on the TCP machine itself is in progress, as is coding of the user interface. These parts should be complete by mid-December. After that, integration of the code with the rest of the operating system remains. Features such as IP and TCP option handling and the direct IP and local network level interfaces will be added in stages after testing begins. Testing should start in late December. We are aiming for a working system in close to final form by mid-February. This will allow time for final debugging and tuning in advance of the planned March 15, 1981 release date.

9.6 References

[1] Postel, J. (ed.), "DoD Standard Transmission Control Protocol," Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC 761, IEN 129, January 1980.

[2] Postel, J. (ed.), "DoD Standard Internet Protocol," Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC 760, IEN 128, January 1980.

[3] Babaoglu, O., W. Joy, and J. Porcar, "Design and Implementation of the Berkeley Virtual Memory Extensions to the UNIX Operating System," Computer Science Division, Dept.

of Electrical Engineering and Computer Science, University of California, Berkeley, December 1979.

[4] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," Bolt Beranek and Newman Inc., Report No. 1822, May 1978 (Revised).

[5] Tenney, R., J. Burruss, and G. Pearson, "Formal Specification of the Transport and Session Protocols," Bolt Beranek and Newman Inc., Report No. 4445, June 1980.

DISTRIBUTION
[QTR 19]

ARPA
Director (3 copies)
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Attn: Program Management

R. Kahn
V. Cerf
J. Dietzler
D. Adams

Defense Documentation Center (12 copies)
Cameron Station
Alexandria, VA 22314

Defense Communications Engineering Center
1850 Wiehle Road
Reston, VA 22090
Attn: Lt. Col. Frank Zimmerman

Department of Defense
9800 Savage Road
Ft. Meade, MD 20755
R. McFarland R17 (2 copies)
M. Tinto S46 (2 copies)

ELEX3101
Naval Electronics Systems Command
Department of the Navy
Washington, DC 20360
Attn: Barry Hughes

ELEX3301
Naval Electronics Systems Command
Department of the Navy
Code 3301
Washington, DC 20360
J. Machado
F. Deckleman

BOLT BERANEK AND NEWMAN INC.
1701 North Fort Myer Drive
Arlington, VA 22209
E. Wolf

DISTRIBUTION (cont'd)
[QTR 19]

BOLT BERANEK AND NEWMAN INC.
50 Moulton Street
Cambridge, MA 02238

R. Alter
A. Owen
G. Falk
R. Bressler
A. Lake
J. Robinson
A. McKenzie
F. Heart
P. Santos
R. Brooks
W. Edmond
J. Haverty
D. McNeill
M. Brescia
A. Nemeth
B. Woznick
R. Thomas
R. Koolish
W. Milliken
S. Groff
M. Hoffman
R. Rettberg
W. Mann
P. Carvey
D. Hunt
P. Cudhea
L. Evenchik
D. Flood Page
J. Herman
J. Sax
R. Hinden
G. Ruth
S. Kent
R. Gurwitz
Library