BOLT BERANEK AND NEWMAN INC

CONSULTING • DEVELOPMENT • RESEARCH

Report No. 2913 October 1974

INTERFACE MESSAGE PROCESSORS FOR THE ARPA COMPUTER NETWORK

QUARTERLY TECHNICAL REPORT NO. 7

1 July 1974 to 30 September 1974

Author:
Dave Walden

(there are some close similarities between this report and McQuillan/Walden 1977)

pp. 59ff. important!
considerations about the dual role of the clocks in the face of apps requiring no reliability but rather a high throughput (with a guaranteed minimum rate) — speech!!

Submitted to:

IMP Program Manager
Range Measurements Lab.
Building 981
Patrick Air Force Base
Cocoa Beach, Florida 32925

This research was supported by the Advanced Research Projects Agency of the Department of Defense and monitored by the Range Measurements Laboratory under Contract No. F08606-73-C-0027.

TABLE OF CONTENTS

1. OVERVIEW

This Quarterly Technical Report, Number 7, describes aspects of our work on the ARPA Computer Network under Contract No. F08606-73-C-0027 during the third quarter of 1974. (Work performed from 1969 through 1972 under Contract No. DAHC-69-C-0179 has been reported in an earlier series of Quarterly Technical Reports, numbered 1-16).

During this quarter, two new network nodes were installed. These were 316 IMPs delivered to Purdue University and the Research Computer Center at BBN. Also this quarter, a third installation, a TIP also at the Research Computer Center, became a new first in using the IMP technology for an intra-site network. This machine is owned by the Research Computer Center and provides connections between PDP-10 processors and the local line printers and file system as well as providing access to the network.

We continued implementation of the plan for ARPA Network access and usage control which was described in Quarterly Technical Report No. 6. The necessary code for Host accounting is mostly complete and we expect it to be fully operational in the fourth quarter. The code required in the TIP for TIP access control and user accounting is also almost complete.

The coming addition of this code to the operational TIP program is expected to further aggravate the lack of buffer space for the terminals. This problem is already visible at several sites (particularly NBS and USC) because of the popularity of 1200-baud terminals. The available buffer space is insufficient to keep these terminals active in the face of normal Host delays. We have therefore recommended increasing the amount of memory in those TIPs.

During this quarter a problem was detected with terminals which do not provide a "Data Terminal Ready" signal to the TIP. If such a terminal remains "logged-in" to a network Host and then emits a continuous stream of break characters (because of a break in the line when some terminals are turned off), the TIP passes this continuous stream of break characters to the Host, using an inordinate amount of processor bandwidth in both the TIP and the Host. We have responded to this problem by beginning development of a modification to the Multi-Line Controller which causes all but the first in a sequence of break characters to be ignored. We expect to distribute the change when it is fully debugged, but probably only to sites at which it is actually needed.

Release of new versions of the IMP system have continued during this quarter, although at a lower rate than expected due to a desire to maintain a stable network program for some time. We did, however, release an incompatible version without unusual interference with the network.

Our involvement with the International Network Working Group (INWG) continued during this quarter and included our participation in the INWG meeting in August.

Subsequent sections of this Quarterly Technical Report (QTR) describe some new aspects of the IMP system; the status of the Pluribus IMP development and some aspects of programming the IMP system on the Pluribus machines; aspects of our satellite effort; the Private Line Interface; new developments in the Network Control Center (NCC); and some basic network design issues.

2. IMP PROGRAM DEVELOPMENT

The third quarter has been one of considerable activity in IMP program development. For instance, the IMP program has been changed to provide for better Host-to-IMP information about situations of Host unavailability as mentioned on page 17 of QTR 5. (In addition, the TIP has been programmed to report this information back to the TIP user. The version of the TIP containing this feature is presently undergoing the pre-release procedures.) Also, the implementation of logical subnetworks as discussed on page 13 of QTR 6 was about half done at quarter's end, with the remainder to be done during about the first two weeks of the fourth quarter.

In another area, the IMP development group cooperated with the Network Speech Group in several experiments with sending packetized speech over the network. These experiments resulted in the detection of a number of network characteristics which were suboptimal for speech transmission. This is not surprising since the network was not originally designed for that purpose; however, these characteristics will be the subject of further study in the fourth quarter with a goal of better optimization of the network for speech.

The "single packet message turbulence" phenomenon discussed by Opderbeck in [9] was investigated at some length in the third quarter, and two changes were made to the IMP system to try to deal with it: a) an interim fix was implemented which forces return to normal operation when the phenomenon happens rather than letting it persist, and b) the Pending Packet Table was slightly increased in size as a diagnostic effort to determine whether source IMP resource conflicts are causing the phenomenon.

Another, more significant, IMP program development is described below.

Packet Core Transfer

The IMP system was initially designed to have only one mechanism for effecting memory reloads, namely a facility for complete memory reloads from a neighboring IMP in a single 16 thousand 16-bit word packet. Later the IMP debugging program, DDT, came to be used for partial memory reloads in certain circumstances (such as loading the TIP program), although DDT was initially designed to be only a debugging tool and not a reloading tool. As the network has become increasingly large and varied, the mechanisms of complete memory reload from a neighbor and of selective reload via DDT have become increasingly awkward and inadequate. Several examples of needs not well satisfied by the existing mechanisms follow:

- The NCC staff would like to be able to propagate new releases from the NCC by sending simple commands to each target IMP telling it to accept core transfers from some other IMP.
- To examine an IMP that crashed, the NCC staff would like to be able to ship the core image of the failed IMP back to the NCC to be examined later by a programmer, or to a spare IMP at BBN to be examined using DDT locally.
- We would like to use a general reloading scheme to load "add-on" pieces such as the TIP, VDH, or Satellite IMP code, which are not part of the basic IMP system, and Hosts for which the NCC has full responsibility such as the PLI.

- In addition, with the possible arrival of the new Pluribus machines on the ARPA Network (Satellite IMP and High Speed Modular IMP), the existing method of loading an IMP from its neighbor is insufficient. Once a Pluribus IMP has been placed outside BBN, most network configurations will require a general way to load a remote IMP across the network through dissimilar machines.
- Loading or dumping a complete core image of a Satellite IMP using a broadcast satellite link is not convenient since it prevents use of the link by other nodes for several seconds.
- A logical adjunct to sending core images through dissimilar machines is sending core images through (perhaps identical) machines which may be running different software. Thus, a new compatible software release can be accomplished in a more modular fashion than at present by loading individual IMPs at the most appropriate time for them, without being constrained to first load at least one neighbor IMP with the new software.

The general scheme for accomplishing all of the above is the "packet core transfer", which consists of two endpoints — the font (source of the core image) and the sink (destination of the core image) — and a protocol governing the reliable transmission of the packetized core image. The font puts pieces of the core image into packets and sends them over the network; the sink receives packets from the network and builds the appropriate core image.

Either font or sink can take one of three basic forms. The first is that of a real network Host, such as the PDP-1D at the NCC. Thus the PDP-1D could be the font for sending "add-on" (i.e. TIP, VDH, etc.) pieces to running IMPs, or the sink for receiving core images of running IMPs for periodic (preventive maintenance) verification of correct software. The second form is that of the "block" fake Host in a running IMP. This fake Host is the font for any transmission of core from the running IMP (i.e. for verification at the NCC or for loading another dead IMP of a similar machine type), and the sink for any pieces of the core image loaded into the live IMP (i.e. "add-on" pieces). The third form is a stand-alone program running in a non-functioning IMP, in conjunction with a live neighbor IMP's block fake Host, which acts as the port into the network for the failed IMP, and is explained more fully below. The block fake Host is able to distinguish between packets addressed to itself (form two) which require it to load or dump its own IMP's core, and packets addressed to a failed neighbor IMP (form three) which it reformats and sends over the connection to that neighbor.

To bring up a non-functioning IMP, the required stand-alone program (that may have to be loaded on-site) should be as short and simple as possible. Therefore, it should know nothing about network procedures such as acknowledging, routing, RFNMs, etc. It should have the smallest possible program to do I/O to its modems, and just enough logic to understand the formats and protocol to do core dumping and loading over a modem. Therefore, it talks over its modems in *blocks*, a data format which is different from any other type of packet, and which carries much less header information. The live neighbor IMP at the other end of the modem also understands blocks. The block fake Host converts packets arriving in standard forms for its non-functioning neighbor into blocks and sends them over the modem to that neighbor. It also converts blocks arriving from its non-functioning neighbor into packets which are then transmitted over the network. When operating in this third form, the block fake Host is completely unaware of the nature and extent of the core transfer; thus, the internal structure of the failed IMP and its neighbor can be completely dissimilar.

The protocol calls for an initiation handshake, followed by a one-way flow of pieces of core image, with possible resynchronization messages in the reverse direction, followed by a termination handshake. The initial request for a core transfer may originate at the NCC, or in a healthy IMP who has a dead neighbor, or from a dead or dying IMP, and goes to the font. The font is now open and ready to send core, and sends a request towards the sink, which then becomes open and ready to receive core. The font follows the initial request with sequence-numbered *pieces* of the core image, and then a "last message" message. If piece-numbers ever get out of sequence, or a piece is excessively tardy, the sink sends back a piece-number reset request, and waits for the correct piece to arrive. When the sink gets the "last message" message, it echos it back to the font, causing the procedure to be terminated.

The general packet core transfer system described above has been implemented and installed in the operational IMP system. It is now beginning to be used routinely by the NCC.

3. PLURIBUS IMP

3.1 Status Repor

The two production copies of the 14-processor system are now running; they execute our system test programs and the operational IMP system. As IMPs they have communicated with one another through a 1-megabit "modem-interface coupler" and, simultaneously, through a 50Kb line to a 316 IMP. As far as the hardware is concerned, these machines are essentially ready for delivery and are now being used primarily for program debugging.

The 14-processor prototype has had some parts excised (still leaving a running machine with fewer processor busses) in order to create a small machine which has been running on the ARPA Network to verify routine performance in a real environment. This machine has run steadily on the net since it was assembled several weeks ago. It was taken down once for a day in order to use it as a test bed and once to reload an updated version of the program to keep pace with a new 316 release.

The logic design of all cards is felt to be quite solid at this point. We recently found and corrected a minor bug in the Bus Coupler, but this is the first such bug in a long time, so it is now possible to convert these cards from wire-wrap to Multiwire form. (Multiwire is an alternative to multi-layer printed circuit cards; we have had considerable success with it on another of our card types.) What remains to be done to the hardware, therefore, is final polishing. As previously reported, we have cultivated our in-house manufacturing and testing (in conjunction with preliminary outside testing), and now have a manufacturing facility which stands ready to provide Pluribus IMPs (or other Pluribus-based systems such as the PLI discussed in Section 5) in a routine manner. We have constructed a price list for these systems which was enclosed with our recent proposal for contract extension.

As far as the Pluribus IMP software is concerned, the basic IMP code has been operational for several months and has been used on the network in the BBN test cell. It is known to have some remaining bugs (not yet isolated), but it is reasonably solid. Those features of the code which allow it to use multiple processors have also been tested on the network but require additional shakedown. The reliability code which attempts to provide recovery for all of the myriad things that can go wrong has been largely written and portions have been debugged; however, considerable additional debugging is required. This code is subject to continuing improvement but it has already reached a level of success such that many hardware resources (processors, memory, I/O devices, etc.) can be traumatically removed from the system while it is running without jeopardizing operation. The system then automatically reabsorbs these units if they are later reconnected. This dramatically robust survival behavior is discussed in greater detail in the following section.

3.2 Pluribus Reliability Software

From a logical point of view, the Pluribus IMP program can be viewed as being composed of three main segments: 1) the IMP system; 2) Pluribus (not IMP) reliability and configuration code; and 3) IMP data structure reliability code. Part 2 is somewhat independent of the IMP program and, as such, is of general interest and therefore worth exploring in some detail.

As it is futile to try to find all possible errors directly at their source, troubles are routinely detected at a fairly high system level. In some sense the system assumes that it is constantly in jeopardy and is perpetually on the lookout for changes in hardware configuration (loss or addition of memory, processors, etc.) or difficulties with software structures (which might indicate either hardware or software failures).

In order to deduce that a problem is associated with a particular shared resource, and not with a processor, it is necessary for the processors to communicate with each other. Thus, if a problem is seen by all the processors, the problem is attributable to a resource (such as a memory) and not a processor. The region in which the processors communicate must be in shared memory, but must be dynamically assigned so that it is resilient to the failure of any particular location.

The Pluribus reliability code assumes that memories are used in the following ways: there is some memory local to each processor that will contain code to handle processor-related events such as unanswered memory references (quits), illegal instruction traps, and interrupts, as well as some application code. Some part of the system reliability code will also live in the processor local memory. Memory common to all processors is divided into pages which may contain one of many kinds of information. There is a *type* associated with each kind of information which defines how it is to be treated. In the Pluribus IMP the different types include: code necessary to the IMP (called warm code); DDT, TTY and display code; data structure reliability code; common variables; buffers; and spare copies of each of the kinds of code. Memories that contain code also contain the limits of the code and a checksum associated with it. There is a precedence to the types, so that if there is insufficient memory, there is no question which type is to be sacrificed.

Given that the processors can communicate among themselves via common memory it is necessary to single out one processor to perform tasks which must be done before the locks have been declared reliable; using only one processor bypasses the need for a lock function for these tasks. Since each processor has a unique identifying number (the address of its bus coupler), at any given time there is always a lowest numbered active processor which can perform those singular tasks.

Each processor has a sequence of stages that it must perform in order to be eligible to run the IMP system. Entry to each stage is dependent on the successful completion of the previous ones. Thus if a processor fails any stage, that processor will be unable to finish the sequence and will not participate in the system tasks. In order, the stages are:

a. Verify that the local memory has a good checksum. The local memory contains some of the IMP code, reliability code, and code necessary for requesting a reload of code from the network. These sections are checksummed independently to reflect the differences in these pieces -- i.e. the reload code may still be used even if the IMP code is broken.

b. Find enough memory which all processors agree is common. The processors perform a brief test to certify the memory and the coupler path to it.

c. Find the communication region being used for the processor-to-processor communication. At this stage, the processors also maintain a set of flags that identify the active processors so that the lowest numbered one may be ascertained. This is done in such a way that the lowest numbered processor may change as necessary. Also, at this stage, processors maintain and rotate a privilege which permits processors to search through their address space for memory, I/O devices, and other processors.

d. Verify the checksums of common memory pages where appropriate. Each program page carries a checksum and address limits for the checksum. Variables and buffer pages are not checksummed. Any pages with bad checksums have their type changed to indicate that they should not be used.

e. Verify that the system's map of pages accurately reflects which pages exist and that there is a page of every required type. If stage (d) has found a bad checksum and has changed the type word, this stage will notice the change and set a flag indicating that a substitute page needs to be found. When all active processors have reached this decision, the last to notice the failure will replace the page with the spare copy, where possible. If it is a spare copy that goes bad or if the spare copy is used, a new spare will be made. If a type such as buffers or variables is needed, either a previously unused memory or one of the least necessary pages will be used. Forcing processors to agree insures that the problem really is with the memory (the shared resource) and not with a particular processor (e.g., its bus coupler).

f. Configure I/O space. Each processor checks to be sure it can reference each device the system is using. The privileged processor mentioned in stage (c) searches through I/O space for other devices, and adds them to the system list. Only devices every processor sees are made available to the IMP system.

g. Configure the processors. The system continually checks to be sure that all processors are running. If one has stopped, it is reloaded and then restarted.

h. Check to see if IMP system initialization needs to be performed. For example, if stage (e) has just installed a new variables page, it must signal the IMP system that it will need to perform its data structure initialization.

i. Run the IMP system. If the system is running properly, the IMP will call the reliability code every slow time-out period (625 milliseconds). Processors use their 60-cycle interrupts to be sure that they have been running their reliability code and, if not, try to determine what may have gone wrong. If they can not find a problem, they reset their reliability code so that they may reenter the system gracefully.

This sequence of steps is done in its entirety when a processor first starts up, and is done periodically, in small pieces, from then on. If at any time one stage fails, the processor ceases to run the system, which is implied by reaching stage (i), and runs only the stages up to and including the one that failed. Thus if the primary copy of a shared code page is destroyed, each processor would detect that fact and cease running the system until the last processor to notice the failure replaces the failing page with a spare copy.

The system configuration code forms part of this sequence. Stages (f) and (g) are responsible for maintaining the status of all active resources in the system. This includes processors, PIDs, Clocks, Hosts, modems, and other I/O devices. A device is accessed in the Pluribus via an 8-word status block located in the top 8K of the processors' address space. If a device is present in the system, referencing its device type and status registers will enable it to be identified. Thus by checking each possible block in the address space, all available devices will be located, at which point they can be initialized and made available to the operational system. This process is always going on (on a periodic basis) so that new devices will always be found.

Each processor maintains a bit in shared memory showing it is running up through stage (c). If a processor has a local reliability failure, for example, if its local checksum is found to be broken, it will cease to run stage (c) and will thus stop setting its bit. This will result in the other running processors reloading and restarting it. It is important to note that this creates a passive system where the operational program notes the absence of a resource, instead of the resource (in this case a processor) having to request attention, which it may be unable to do.

The reliability code also includes the treatment of processor interrupts. This includes non-responding memory, illegal instructions execution, 60-cycle interrupts, power fail, and power recovery interrupts. Non-responding memory references (quits) can be either expected, e.g., searching for new memory or I/O, or unexpected. The latter may indicate a memory or coupler failure and thus serve as input to the memory checking code. Illegal instructions of certain types are used for system error counters (in the manner of PDP-10 UUOs). Otherwise they may indicate broken code or a broken processor; in which case, the memory checking code is called in. Generally, an error that does not disappear will cause the processor to cycle through its reliability stages, allowing fast response to the loss of system components.

Power detection interrupts are divided into two classes. The power fail interrupt can signal either loss of power on the processor bus or loss of power on one of the memory or I/O busses. In the latter case the interrupt comes at least 2 1/2 milliseconds before the bus is actually unusable. This is sufficient time for certain important data to be saved. The other type, power recovery, indicates that the bus has just had power restored and the processor should certify its resources before proceeding. To this end, the processor starts at the beginning of the multi-stage procedure described above.

The last type of interrupt, which happens on the 60-day clock, is used to detect some types of control structure failures. For example, when the operational program is waiting for a resource lock, it remains in a tight loop until it succeeds. To catch the possibility of the resource lock being broken, the interrupt code monitors the status of the processor as it entered the interrupt. If it has not run through the reliability stages in a given period of time, and is presently at a recognizable lock instruction, then the lock is presumed to be broken, and remedial action is initiated. If there is no lock instruction, the processor assumes it has been in an infinite loop, retries its reliability stages, and reenters the system cleanly.

The system reliability code has been designed to be as independent of the IMP program as possible. Failures that do not prohibit running the IMP program altogether will be communicated to the IMP program through flags maintained by the reliability stages. Likewise, the configuration code will maintain tables accessible to the IMP system. Stages (h) and (i) really are part of the IMP system. Stage (e), although not a part of the IMP system, was coded in an application-dependent fashion. Thus, we need only tailor this one stage to allow the same reliability code, stages (a) to (g), to drive some new, non-IMP application of the Pluribus.

4. SATELLITE ACTIVITIES

Over the past two years we have been discussing with COMSAT a proposed satellite experiment which would introduce a broadcast satellite connection across the Atlantic. As proposed, this connection would consist of a Satellite IMP in the COMSAT ground station at Etam, West Virginia, conversing with an identical Satellite IMP in the Goonhilly, U.K., ground station, with the possible addition of other nodes at a later time. The hardware considerations of connecting a Satellite IMP to the satellite channel have been resolved, with COMSAT taking responsibility for the necessary changes to the satellite channel equipment and BBN taking responsibility for the interface in the Honeywell 316 minicomputer.

In addition to these interconnection negotiations, we have been involved in the resolution of the political problems involving the ownership of, and control over, the Satellite IMP and shortly we expect COMSAT to file the necessary tariff with the FCC. At that time, the United States end of the experiment will be ready.

The software in the Satellite IMP has been complete and ready for delivery for several months. We have had a pair of Satellite IMPs with a satellite simulator (which provides a quarter-second delay) between them on the network at various times during this period. Once, they were placed between an in-house TIP and the network. The behavior of this configuration was quite stable and reliable. We are currently anticipating no further work on the program except for that necessary to maintain compatability with new IMP program releases.

In parallel with the previously mentioned efforts, we have been developing Pluribus-based Satellite IMPs for both medium (50Kbs) and high speed (up to 1.5Mbs) satellite channels. Development of the software for these new machines has proceeded very well. Much of the code specific to the satellite channel has been developed and tested, including the implementation of a Follow-the-Leader slotting algorithm (as is used in the 316 Satellite IMP) and the TDMA, Slotted ALOHA, and rudimentary Reservation ALOHA protocols. The Pluribus program is being written so that the speed of the satellite line (which influences the acknowledgment protocol used and various timing parameters) is a program variable that can be changed when the program is executed. To test the program, a small routine has been included which simulates the action of the satellite interface and the satellite channel, accepting the packet given to it by the program, delaying the packet for 1/4 second, and returning the packet to the input portion of the program. Some of the remaining portions of the program which have not been implemented include the interface with an IMP program in the same machine, statistics, garbage collection of packets assigned to a dead neighbor, part of the Hello I-Heard-You algorithm, and the reliability code for the Satellite algorithms.

4.1 Dynamic Protocols

Both the 316-based and Pluribus-based Satellite IMPs implement the satellite channel protocols in a manner which permits them to be changed while the system is in operation. While these algorithms are active, the retransmission structure of Slotted ALOHA remains at the base of the program. The

Satellite IMP always watches the channel for packets which it sent a quarter second ago. If it fails to hear a packet, it prepares the packet for retransmission. As every packet is received, the program records the number of the transmitting node in a table which is indexed by slot. This mechanism continuously monitors the potential owners of the various slots. With these input routines always active, specification of a protocol only affects the decision of whether a node should transmit into a particular slot. The naturally robust Slotted ALOHA protocol operating below any other protocol permits recovery from the conflicts caused by differing protocols.

4.2 Checksum Considerations

The ALOHA protocols assume that when transmission conflicts occur, destroying the packet, the checksum will detect the data errors due to this interference. It is important that the checksum detect this interference in order to prevent erroneous packets from entering the network. ALOHA protocols, however, stress the checksum much more than ordinary line error rates. While a typical line error rate is one bit error in 10^6 bits, or one packet in error out of 1000, a broadcast channel operating with Slotted ALOHA *at maximum throughput* has approximately 1 packet in error out of 3 slots. The effect of conflicts clearly dominates other line errors on a broadcast channel. We can estimate the effectiveness of an n-bit checksum in this new environment.

In order for a packet to be accepted by the Satellite IMP, the following must occur: the satellite channel modem must acquire the sub-carrier modulation and establish bit synchronization, the interface must correctly receive a 24-bit start-of-text sequence, the contents of the n-bit check register must be zero at the end of the packet, and the software checksum (if any) must be valid.

When a transmission conflict occurs, the signals from both bursts are added together and presented to the receiving satellite modem. The sub-carrier signal resulting from this summation has an amplitude between zero and twice the amplitude of either signal. It is likely that the receiving modem will be able to accept this composite signal in most cases. Bit time synchronization in the modem then requires the observance of a preamble containing a unique word. Without knowing the exact nature of the modem, we can assume that if the conflicting bursts start at very nearly the same time, then the bits sent will match, and the signal resulting from their combination will be acceptable to the modem. If the protocol in use were Random ALOHA, the chance for such exact bit synchronization would be very small, and the probility that this modem would accept the burst would be very small; however, when using a protocol with slotting, the slotting algorithm synchronizes the transmissions, and this increases the probability of a match close enough to be accepted by the modem. We have estimated that the slotting algorithms in the Satellite IMP will maintain synchronization to within one half a bit time (assumed as the maximum deviation which will be accepted by the modem) approximately one-sixth of the time.

If the signals are synchronized accurately enough that the modem acquires bit synchronization, then it is also very likely that the 24-bit unique word required for character synchronization in the satellite interface will also match.

While the preambles of the two bursts involved in a transmission conflict are identical, the text of the bursts is quite different and we may assume that the text resulting from the combination of the two bursts will be completely random. With such a high effective bit error rate, the details of the checksum implemented become irrelevant and the contents of the checksum register at the end of the burst can be assumed to be random. Thus, with an n-bit checksum, the probability that the checksum register contains all zeros is 2^{-n}. This then is the probability that the packet is found acceptable by the checksum logic.

Before the packet is accepted by the Satellite IMP, the software checksum must be validated. The current Satellite IMP uses a 16-bit additive checksum which further reduces the probility of accepting a transmission conflict. This checksum, however, represents a higher layer of protocol, and failures in this checksum should represent failures of the system operating below that protocol; in particular, the software checksum is needed to help isolate hardware failures. Since we will continue to need this diagnostic function, we ignore the software checksum in the calculations below.

Combining the previous effects, we can estimate the expected rate of accepting erroneous packets due to transmission conflicts. We will assume that the broadcast channel is operating at 80% of the maximum possible throughput and that the channel, therefore contains 1 conflict in ten slots. Then, for a 1.5 megabit channel, the rate of accepting erroneous packets is:

$$\frac{1500 \text{ slots}}{\text{second}} \times \frac{1 \text{ conflict}}{10 \text{ slots}} \times \frac{1 \text{ synchronized}}{6 \text{ conflicts}}$$

$$\times \frac{2^{-n} \text{ accepted}}{\text{synchronized}} \times \frac{8.64 \times 10^{4} \text{ seconds}}{1 \text{ day}}$$

For a 24-bit checksum, this is 8 days between errors; with a 32-bit checksum, it would be one undetected error in 5 years. It appears, therefore, that for a megabit satellite line used in this way a 24-bit checksum is inadequate while a 32-bit checksum probably is adequate. A similar calculation for a 56Kbs line indicates that a 24-bit checksum permits one undetected error in 200 days, also probably adequate. Thus, our design for the 56Kbs satellite interface contains the usual 24-bit checksum logic, but we have begun implementation of the megabit satellite interface for the Pluribus Satellite IMPs with a 32-bit checksum. R. Kahn of the ARPA office, who was involved in the design of the 24-bit checksum presently in use in the ARPA Network, is helping with the selection of the 32-bit checksum.

5. THE PRIVATE LINE INTERFACE

The effort this quarter on the Private Line Interface (PLI) included the design of the Synchronous Modem Simulator card for the PLI/2 ("bitstream PLI") and the programming and checkout of the basic PLI code. We are now in position to give preliminary functional specifications for the PLI/1 ("secure PLI") and PLI/2 software. These are included below, along with a description of the internal design of the basic PLI program.

5.1 PLI Functional Specification

The Private Line Interface is a Pluribus mini-Host capable of accepting data from various sources and passing the data over the ARPA Network to a matching PLI. The interfacing logic, both hardware and software, must be tailored for each application. The basic mini-Host code is usable with a variety of interfaces. Currently, a Teletype interface is being used for testing and checkout.

The PLI/1 will be used to transmit secure Host-Host traffic over the network. It requires several data interfaces: first, a Host-PLI interface which makes the PLI appear to the Host to be an IMP; secondly, separate interfaces to the red and black halves of a KG-34 key generator unit.

Specifications for the Host-PLI interface are identical to the specification of Host-IMP communications (see BBN report 1822, Specifications for the Interconnection of a Host and an IMP). The standard Host-PLI interface has been demonstrated, and the Very Distant Host interface will be completed next quarter. These two interfaces use standard Pluribus hardware.

Final specifications for the KG-34 interfaces will be developed when funding becomes available. The current hardware interface specifications have been based on NSA CSEEB-9B plus extensive discussions with COMSEC personnel. These interfaces require the design, construction, and testing of RFI shields and isolators. An overview of the current design was presented in our Quarterly Technical Report No. 5. The current design for the software interface is presented below.

The PLI/2 will interface to Lincoln Laboratory CVSD (Continuously Variable Slope Delta-modulation) Vocoding devices through a Pluribus SMS (Synchronous Modem Simulator). This is a general purpose synchronous interface card designed to provide a range of functions in various possible PLI interfaces.

Initially, the PLI/2 software interface will simply poll the SMS card, gather the data into packets, and transmit them. At the receiving site, the data will be output as received. The output clocks will be set a fraction faster than the input clocks so that data does not accumulate in either PLI. The difference will be made up by occasional short bursts of silence. After this setup is working, it will be possible to experiment with improvements such as silence detection and smoothing.

5.2 The Basic PLI Software

To a large extent the basic PLI has been written by adapting code and algorithms from the Pluribus IMP. It uses a PID, has its code arranged in strips, and uses similar primary data structures. The code anticipates, but does not require, a multi-bus, multi-processor configuration. The code is read-only. It includes a quit routine to handle references to non-responsive memories or devices, some dynamic data structure checking routines, and a code checksumming module. Timing functions are derived from the 60-Hertz interrupt provided to the Pluribus processor.

Internally the PLI uses a common pool of dynamically allocated, fixed length buffers to hold data packets. Each buffer includes a block of control data associated with its current packet. Hardware and polling interfaces use these data buffers directly. When a buffer has been filled or emptied, a PID level is set to dispatch to the appropriate strip. Each PID level has a statically allocated work area where the associated strip maintains the logical state of the interface.

A typical strip works as follows:

1) The work area and device are initialized
2) A buffer is obtained from the device input queue (or an empty buffer for an input strip)
3) The device is activated
4) A complete packet is formatted or sent
5) The buffer is placed on the TASK queue for routing to another strip (or freed if an output strip)
6) Processing returns to state 2.

The task queue handler determines the destination of the packets on its queue and moves them to the required output device queue for transmission, activating that device when necessary. RFNMs are handled using dynamically allocated transaction blocks to record the transmission and completion of messages in the Host-PLI interface. Otherwise RFNMs are ignored. Retransmissions are not attempted by the PLI. All data structures and program states are timed out by software clocking mechanisms and reset when necessary.

Each packet sent between PLIs includes a three-word leader which contains the packet length in bytes, a sub-routing word to allow for a number of PLI interfaces at each end, a word used to hold the original message identifier for the Host-PLI interface, and various control bits.

5.3 The KG-34 Software Interface Design

The PLI/1 is really two PLIs in series. The secure Host communicates with the red-half PLI via a normal Host-PLI interface. The red-half PLI signals the KG-34 unit to generate and send a key-sequence. The black-half PLI supplies clocking to the KG-34. It scans the incoming data stream for a key-sequence, compresses the key, and stores it as the first few bytes of the message. Then the red-half sends a message segment (initially somewhat less than 1000 bits) through the KG unit. The segment is padded with SYN characters if needed to bring it to a fixed size. The black-half adds this data block to the key, assigns a fixed destination, message identifier, and sub-routing code, then transmits it over the network. It also notifies the red-half of the message identifier assigned via a uni-directional data and status link (from black to red). When a RFNM or other status indicator is received from the IMP the red-half is notified over this same data link.

When the black-half receives an encrypted message from the IMP, it preps the KG-34, expands the key-sequence in the message, then transmits the data through the KG unit to the red-half. Since the black-half provides a clocking signal, the red-half must be prepared to accept the decrypted data as fast as it is sent. There is no way for the red-half to indicate overrun or other errors to the black-half, although errors are reported to the Host. This design requires the Host-Host protocol to effect retransmission should errors of this type occur.

Both KG-34 interfaces operate on the polling principle. This limits their average bandwidth, as does the long delay (milliseconds) involved in preping the KG unit. The use of strips does not affect the bandwidth since the hardware includes enough buffering to smooth out this effect.

6. NETWORK MAINTENANCE AND THE NETWORK CONTROL CENTER

During the past year, and particularly during the third quarter, we have undertaken a number of changes to the operation of the Network Control Center (NCC) and to our approach to network hardware maintenance. We will discuss several of those changes below.

6.1 Hardware Maintenance

First, as mentioned in our Quarterly Technical Report No. 5, a BBN hardware engineer was added as a full-time member of the NCC staff; during the third quarter a high-level field service technician was also added to the staff. The installation team (one engineer and two high-level technicians) has thereby been freed to spend full time on preparation of new equipment to be installed. This additional manpower has further increased our ability to respond quickly to problems, with a resultant increase in the average IMP availability. The table below summarizes the average IMP percent down due to hardware and software failures over the last nine quarters; the figures clearly indicate that this additional effort is bearing fruit.

Quarter	Average Percent Down
3rd, 1972	2.27
4th, 1972	1.88
1st, 1973	1.95
2nd, 1973	1.07
3rd, 1973	2.09
4th, 1973	1.51
1st, 1974	.80
2nd, 1974	.85
3rd, 1974	.57

In addition to the hardware maintenance and trouble-shooting staff members discussed above, both of whom work from Cambridge, we added a high-level technician to the BBN office in Arlington, Virginia during the third quarter; and we have, with ARPA's permission, assigned responsibility for maintenance of the Washington, D.C. area to this technician (with Cambridge backup). We have cancelled the Honeywell maintenance on the machines at ARPA, Belvoir, ETAC, Mitre, NBS, and SDAC and, when this cancellation takes effect at the beginning of the fourth quarter, we will perform all necessary maintenance on these machines ourselves. This change in maintenance responsibility for a "cluster" of machines will provide a yardstick for evaluation of Honeywell maintenance in other areas of high node concentration such as Boston, Los Angeles, and San Francisco. After some experience, we expect to be able to recommend either reversion to the previous arrangements in Washington or possibly expansion of BBN responsibility to cover other locations.

6.2 Host Monitoring

During the second and third quarters we have also expanded the functions of the NCC to include monitoring the accessibility of certain "critical" Hosts. ARPA has selected the set of service Hosts which are deemed critical and the NCC program has been modified to display the status of these Hosts on the lighted display panel. Although we have no authority over the operation of these Hosts, the NCC operators now communicate with, and offer assistance to, the operations personnel at the site whenever one of these Hosts goes "down" to the Network. The NCC operators also maintain an on-line list of scheduled down time (available through the TIP @N command), for all Hosts which choose to participate, but with special attention to the "critical" Hosts.

6.3 Traffic Measurements

We last reported on network traffic in our Quarterly Technical Report No. 1 at the end of the first quarter of 1973. At that time the traffic rate was an average of about two million (internode) packets per day, with exponential growth at the rate of an order of magnitude every 13 months. Since that time the traffic growth rate has slowed considerably, rising irregularly to roughly 3.75 million packets per day at the present time. In fact, so far in 1974 the average monthly internode traffic has ranged between a low of 2.9 million packets per day (January) and a high of 3.8 million packets per day (July). This growth appears to be more a result of limited available computing capacity, or limited desire for network access, rather than network saturation, since the network is currently limited by circuit capacity rather than IMP capacity, and the busiest circuits are currently used at less than 50% of capacity during peak hours (under the most pessimistic assumptions about message length) and probably less than 15% (using the NMC's measurement of average packet length as reported in the literature).

While discussing traffic statistics, we should note that at the beginning of the third quarter we began using the traffic data base which is stored on the BBN TENEX system for production of traffic summaries. The NCC Host computer transfers this data to TENEX once each hour, using a specialized Host-to-Host protocol. Desired summaries are then produced by FORTRAN programs running on TENEX. During the fourth quarter we hope to complete additional FORTRAN programs which will produce summary graphs which have been requested by ARPA.

6.4 Software Release Procedures for Incompatible Systems

In our Quarterly Technical Report No. 14 (Contract No. DAHC-15-69-C-0179) we described in some detail our difficulties in performing an "incompatible" change to the IMP system software (by "incompatible" we mean that the two versions of the system cannot intercommunicate). Since that time we have changed the system more than 30 times, but we have always been able to keep the systems compatible. During the third quarter we found it necessary to change the inter-IMP packet format; the system which implemented this change was clearly incompatible with the previous system. (The packet format change was necessitated by the plans for "fairness" and "logical subnetworks" described in our Quarterly Technical Report No. 6.) Normally, the hours between 7 and 9 a.m. (ET) each Tuesday are reserved for IMP software changes; in this case the release time was extended to run between 6 and 9:30 a.m. The mechanisms we used were basically the same as those described in Report No. 14 with two exceptions:

1. Report No. 14 described a method for using two IMPs, with identical IMP numbers, to control the release process. During the incompatible release this quarter, due to the greatly increased network size and consequent probability of confusion, we also used two NCC Hosts, one connected to each of the two incompatible portions of the network, to monitor our progress.

2. Report No. 14 described a rather clumsy technique for releasing the new system to sets of IMPs which have only one (operational) path to BBN. The technique involved "patching" the software in each IMP in the set so that it would fail to honor reload requests, then ordering each IMP to request a reload. Once one IMP was finally reloaded (overwriting the patch) it would then honor the request from its neighbor, and the new system would then propagate down the chain. Since that time we have implemented a special inter-IMP message which demands that the receiving IMP attempt to reload itself from the sending IMP. Since this mechanism remained compatible, we used it to release the new system to portions of the network with only a single path to BBN.

The incompatible release procedure worked well, with the release completed just after 8:30. None of the problems which we described in Report No. 14 occurred, thus strengthening our confidence in our ability to make incompatible releases smoothly when necessary.

7. NETWORK DESIGN ISSUES

Over the past few years there has been much examination of the ARPA Network algorithms in view of operational problems observed, alternative algorithms chosen for other networks, and the large body of theoretical work on packet-switching networks being carried on at universities and elsewhere.

In this section we consider a few (by no means all) of the packet-switching design issues now under widespread consideration.

7.1 Design Requirements

We begin by giving the properties central to the ARPA Network design. The key assumption here is that the *packet processing algorithms* (acknowledgment/retransmission strategies used to control transmission over noisy circuits, routing, etc.) result in a virtual network path with the following characteristics:

a. finite, fluctuating, delay -- a property of the circuits

b. finite, fluctuating, bandwidth -- a property of the circuits

c. finite error rate (duplicate or lost packets) -- a property of the acknowledgment ("ack") system (this is a different use of the term "error rate" than in traditional telephony)

d. disordering of packets -- a property of the ack system and routing

Duplicate packets are caused when an IMP goes down after receiving a packet and forwarding it without having sent the ack. The previous IMP then generates a duplicate with its retransmission of the packet. Packets are lost when an IMP goes down after receiving a packet and acknowledging it before the successful transmission of the packet to the next IMP. These four properties describe what we term the *store-and-forward subnetwork*.

There are also two basic characteristics of the source and destination IMPs in the virtual path described above:

e. finite storage -- a property of the IMPs

f. differing source and destination bandwidths -- largely a property of the Hosts.

A slightly different treatment of this subject can be found in [1] by L. Pouzin.

The fundamental requirements for the ARPA Network *message processing algorithms* are dictated by the six factors enumerated above. These requirements include:

a. Buffering -- Because of the *finite delay* of the network, it may be desirable to have buffering for multiple messages in flight to increase throughput, that is, a system without buffering (one message at a time) may have unacceptably low throughput due to long round-trip delays.

b. Pipelining -- The *finite bandwidth* of the network may necessitate the pipelining of messages through the network, by breaking them up into packets, in order to decrease delay. The bandwidth of the circuits may be low enough so that forwarding the entire message at each hop in the path results in excessive delay. By packetizing the message, the IMPs are able to forward the first packets of the message through the network ahead of the later ones. For a message of P packets and a path of H hops, the delay is proportional to P + H - 1 instead of P * H.

c. Error Control - The destination may need to exercise some controls to detect missing and duplicated packets, which would appear as incorrect data to the end user. Further, reports on delivery or non-delivery of messages may be useful, possibly to trigger retransmission. This mechanism in turn requires error control and retransmission itself, since the delivery reports can be lost or duplicated. The usual technique is to assign some unique number to identify each data unit and to time out unanswered units.

d. Sequencing -- Since *packet sequences are received out of order* the destination must use a sequence number technique of some form to deliver messages in correct order, and packets in order within messages, despite any scrambling effect that takes place while several messages are in transit.

e. Storage allocation -- The *finite storage* in both the source and destination means that each must exercise control over its use. The two basic approaches are for the source to hold a copy of the message which then allows the destination to treat each message as discardable if it does not have storage for it, and for the source to request or otherwise obtain reserved storage at the destination which removes the need for the source to hold a copy. The first approach allows low delay since there is no setup time in obtaining a storage reservation in the destination, while the second allows high throughput (if reservations are long-lived) by eliminating retransmissions caused when storage is over-subscribed. Of course, storage need not be committed at either the source or the destination if reliable transmission of messages is unimportant, and storage can be reserved at both source and destination for simultaneous optimization of delay, throughput, and reliability, although at some added cost.

f. Flow Control -- The *different source and destination data rates* may necessitate implicit or explicit flow control rules to prevent the network from becoming congested when the destination is slower than the source. These rules can be tied to the sequencing mechanism (no more messages after a certain number are acceptable) or to the storage allocation technique (no more messages can flow until a certain amount of storage is free) or can be independent of these features.

In performing these functions, the message processing system is often exercising contention resolution rules to allocate scarce resources among several users. The twin problems of any such facility are:

- fairness -- resources should be used by all users fairly
- deadlock prevention -- resources must be allocated so as to avoid deadlocks.

(We have also come to believe that it is essential to have a reset mechanism to prevent "impossible" deadlocks and other conditions that may result from hardware or software failures.)

Having considered the design requirements, we now turn to a number of specific design issues.

7.2 No Message Processing in the Subnetwork

There has been considerable discussion in the packet-switching community about the amount and kind of message processing that should be done in communications subnetworks. An important part of the ARPA Network design which has become controversial is the ARPA Network system of messages and packets within the subnetwork, ordering of messages, guaranteed message delivery, and so on. In particular, the idea has been put forth that many of these functions should be moved from subnetwork to Host level. As we attempt to show in this section, we feel the ARPA Network design is a good one, as good overall as any other recent proposal for a network design. We recognize deficiencies in the present implementation, but we feel that each problem is relatively easy to solve in a natural way within the framework of the philosophy of the present design. We also recognize that there are no absolutes at this stage in the development of computer networks, and we look forward with anticipation to the successful conclusion of the construction of those networks which have chosen divergent design principles so that we may better compare the results of differing design philosophies. Differing views on specific design choices in this area have been proposed by the designers of the Cyclades network [2], writers on the subject of inter-network communication [3], and ARPA Network researchers commenting on their experience [4], as well as others.

We summarize the principles usually given for eliminating message processing from the communications subnetwork:

a. For complete reliability, Hosts must do the same jobs and therefore, the IMPs should not.

b. Host/Host performance may be degraded by the IMPs doing these jobs.

c. Network interconnection may be impeded by the IMPs doing message processing.

d. Lockups can happen in subnetwork message processing.

e. The IMP would become simpler and have more buffering capacity if it did not have to do message processing.

The last point is true (if we can ignore the issue of whether the IMP *must* do message processing for its own control, statistics, and debugging messages), but the other statements are certainly subject to some question.

Much of the criticism of the ARPA Network design is based on misconceptions about the design or is criticism of specific network characteristics which could be fixed easily. For instance, a major motivation for many of the changes from the ARPA Network design has been the *perception* that the ARPA Network design limits bandwidth and suffers from lockups. This perception, however, is inaccurate; except for implementation bugs and deficiencies, the ARPA Network design is sound and free from lockups with no inherent bandwidth limitation--we have chosen the structure and operating parameters of the ARPA Network algorithms so that the limit they impose on Host throughput is higher than that imposed by the basic bit rates of the network circuits and Host interfaces. Another motivation has been the apparent coupling of the Hosts on an IMP in such a way that they can relatively easily interfere with each other because they share the same message number pipe to a common destination. In fact, such coupling is not fundamental to the ARPA Network design and there are plans to decouple the Hosts on an IMP. These two subjects are treated in detail in a later section.

Whatever is done at the Host level, it is critical for the communications subnetwork to handle its own problems or else it will perform poorly in many dimensions; e.g., message loss, message ordering, etc. Without such controls, the Hosts will be in conflict with each other, degrading their performance, with no way to resolve these conflicts. (For example, if the subnetwork handles congestion by merely discarding packets whenever congestion occurs, it is as likely to discard packets of "well behaved" Hosts, whose I/O rates are closely coordinated, as to discard packets from streams between Hosts which are not coordinating their rates). The following reasoning indicates why this is so:

- The function of flow control is to prevent a source from overflowing the finite buffer capacity of a slower destination.
- All messages go through the destination IMP before going to the destination Host.
- Even with great buffering capacity, the IMP buffers would *sometimes* overflow.
- The Hosts are intrinsically unable to allocate IMP storage precisely.
- When destination IMP buffer overflow occurs, performance is drastically reduced because the source must retransmit, increasing delay and decreasing throughput.

In general, in large networks with many Hosts, the majority of Hosts will find a communications subnetwork insufferable if it is too prone to message loss. Despite the fact that some Hosts may not care how prone to message loss the subnetwork is, and despite the fact that many Hosts may need to duplicate functions also performed in the subnetwork for those (hopefully rare) instances when the subnetwork does fail, the majority of Hosts will want the subnetwork to do the best job it can so as to minimize Host performance of these functions. Every channel is noisy; and the IMP must do what it can to minimize noise. Furthermore, this is basically inexpensive to do in the IMP, especially when compared to the cost of doing it in *all* the Hosts and, generally, what is done at IMP level is in no way detrimental to the Host level, even if it does not help the Host level.

Moving these functions to Host level also does not solve the perceived efficiency and deadlock problems associated with performing these functions in the subnetwork. The problems, if they really exist, are merely moved out one level, although with larger memories in some Hosts, the problems may be less frequent. (In the ARPA Network case, the problems are also pushed to the fake Hosts [processes within IMPs] and the Terminal IMP.)

In the few areas where the present ARPA Network design does present real limitations to some desired types of network use (e.g., transmission of fixed, very low delay traffic such as speech), mechanisms can be added to the ARPA Network in parallel to the existing mechanisms for the purpose of accommodating these desired but presently unobtainable types of traffic service.

One can easily imagine that if very many of these functions were removed from the IMP and given to the Hosts a large number of Hosts would add a mini-Host between the IMP and Host to perform these functions. In fact, the trend today, with declining computer costs, is to remove functions from large computers and to put them in peripheral processors, not to add functions to the large computers. Notice, however, that if a peripheral processor is to take on the message processing jobs which the IMPs now perform, then it will require a large storage device to buffer out-of-order packets or else it will suffer the same type of contention for buffer space which occasionally occurs in the IMPs. In order to obtain desirable delay and bandwidth characteristics, such a peripheral processor will be obliged to work with the set of all other such processors and Hosts and reinvent mechanisms similar to those currently in the IMPs. But with many *independent* implementations of this kind, not only will initial implementation cost be high, but coordination and testing of any necessary or desirable changes will be extraordinarily difficult. This is in sharp contrast to the relative ease of making corrections, improvements, and even fundamental design changes in the subnetwork.

We now summarize our main contentions about the place of message processing facilities in networks:

a. A layering of functions, a hierarchy of control, is essential in a complex network environment:

- For efficiency, IMPs must control subnetwork resources, and Hosts must control Host resources.
- For reliability, the basic subnetwork environment must be under the effective control of the IMP program -- Hosts should not be able to affect the usefulness of the network to other Hosts.

- For maintainability, the fundamental message processing program should be IMP software, which can be changed under central control and much more simply than all Host programs.
- For debugging, a hierarchy of procedures is essential, since otherwise the solution of any network difficulty will require investigating *all* programs (including Host programs) for possible involvement in the trouble.

b. The nature of the problem of message processing does not change if it is moved out of the network and into the Hosts; the Hosts would then have this very difficult job even if they do not want it.

c. Moving this task into the Hosts does not alleviate any network problems such as congestion, Host interference, or suboptimal performance but, in fact, makes them worse since the Hosts cannot control the use of IMP resources such as buffering, CPU bandwidth, and line bandwidth.

d. It is cheap to do message processing in the IMPs, and it has very few detrimental effects.

One party who has been particularly active in support of removing message processing altogether from the communications subnetwork has been Cerf, especially in his paper assessing the ARPA Network protocols [4]. Specific conclusions that Cerf reached in his paper include:

a. Multi-packet messages can be eliminated without loss of potential throughput;

b. Ordering at the destination IMP can lead to lockups, and since the Hosts must do ordering anyway if they are to recover in the cases where the subnetwork fails, the IMPs need not do it;

c. If the destination IMP doesn't reorder or reassemble, arbitrarily long messages are possible;

d. Retransmission, end-to-end acknowledgments, error detection, duplicate detection, and message ordering at Host level eliminate this need in the IMP subnetwork.

It is our contention that all four of these conclusions are flawed. At the risk of some duplication of what was said above, we respond to each of Cerf's points:

a. With regard to eliminating multi-packet messages while still maintaining high Host throughput, Cerf's conclusion is based on three ideas, to which we offer rebuttals:

   1) that use of multi-packet messages presents a buffering limit -- the limit Cerf is concerned with is actually a bit-coding limit and one that can easily be removed;

   2) that Hosts use the network suboptimally since they have agreed among themselves to have only one message at a time outstanding (on a given "connection"); by using the network better, the Hosts could buy back any loss caused by eliminating multi-packet messages -- in fact the Hosts could improve their use of the network, but this improvement should be added to the network's already good performance rather than wasted in recouping the loss in network performance resulting from removing multi-packet messages (the fact that they have not done so is at least partly due to the difficulties in coordinating desirable

changes to independent implementations which we discussed above) and,

3) that eliminating multi-packet messages would cause only minimal performance degradation -- in fact, even without multi-packet messages the subnetwork would have to perform similar functions on groups of packets or else great loss in performance would be incurred.

b. With regard to the idea that ordering in the subnetwork can cause lockups:

   1) the fundamental idea of the IMP reordering scheme wherein all destination storage is allocated is sound, and the notion that the IMP can lock up with out-of-order messages is incorrect;

   2) doing reordering at the Host level is also prone to lockups. Although there is freedom at the outermost level to discard troublesome packets to resolve deadlocks, this will result in the same inefficiencies as it would if done too frequently at inner levels; and,

   3) customers invariably want their packets and messages delivered in the order sent (potential commercial networks and military networks confirm this desire) and therefore this job should be done in a centralized place, for reasons of economy in implementation.

c. The idea of obtaining arbitrarily long messages is so weak that even Cerf notes its fatal flaw: it requires the nodes to know the Host/Host protocol(s). Furthermore, it is possible to have arbitrarily long data streams in the ARPA Network made up of finite-size messages.

d. The fact that the Hosts can do the same functions that are done in the subnetwork does not mean that they all want to, that it is as efficient for them to do so, or that even if they did do these functions the subnetwork would not still have to do them if it was not to perform very badly. In fact, it is clear that the subnetwork must protect itself against local congestion, mismatched input and output rates, and the like.

7.3 Single-Packet Messages Only

The following question is often asked: "If one increases packet size, and decreases message size until they become the same, will not the difficult reassembly and flow control problems be removed?" The answer is that, unfortunately perhaps, message size and packet size are almost unrelated to flow control and reassembly, and we discuss each of these four issues below.

Packet Size. Network delay is a complicated function of line speeds, propagation delay, utilization of the lines, the priority scheme used, IMP processing time, and the possibility of preemption. In the present ARPA Network, delay for a small priority message is, to first order, proportional to the packet size of the other traffic in the network. Thus, small packets are desirable. This attitude changes when lines become so long or fast that propagation delay is larger than transmission time.

Message Size. Message size needs to be large because the overhead on messages is significant. It is inefficient for Hosts to have too many message interrupts, and for the IMPs to have to address too many messages. The upper limit on message size in the current network is what can conveniently be reassembled, given IMP storage and network delays.

Reassembly. When a channel has an appreciable delay, several pieces of data must be in progress in the channel at one time in order to achieve full utilization of the channel. It makes little difference whether these pieces are called packets which must be reassembled or messages which must be delivered in order. In the ARPA Network without satellite links, the amount of data which must be "in the pipe" between the source and destination IMP to fully utilize it is on the order of thirty packets. With satellite links, many more packets are necessary. Unless one is willing to endure a three percent (100%/30) utilization of the source-to-destination pipe, there must be reassembly, in the IMPs, or if not in the IMPs, in the Hosts.

Flow Control. Flow control requires there to be space to hold all the traffic in the "pipe" should the destination suddenly slow its rate of acceptance of the traffic. Since there are many "pipes" in a network, flow control becomes complex. In the ARPA Network, flow control is currently done on "chunks" of that space which happen to be the same size as a message because that is a convenient size and simplifies bookkeeping. Of course, this is not entirely coincidental as the same forces work to determine reasonable sizes for both the message units and the units of flow control. Packet size seems to have little relation to flow control in the ARPA Network implementation.

7.4 Packet Size

There has been much thought given in the packet-switching community to the proper size for packets.

For instance, in his thesis [5], Metcalfe points out that larger packets have a harder time getting across an error-prone telephone circuit, and this drives the packet size down. At the same time, Metcalfe points out, overhead considerations (longer packets have a lower percentage overhead) drive packet size up. For a set of assumptions about the ARPA Network circuit error characteristics, Metcalfe then calculates the packet size which optimizes effective throughput (i.e., not too much overhead, and not too much lost to retransmissions due to errors. In fact, for his assumed error rates, Metcalfe concludes that the ARPA Network packet size of about 1000 bits is within the flat range on the optimization curve and is thus sufficiently well chosen.

At another point in his thesis, Metcalfe notes that store-and-forward delay is reduced as packet size becomes smaller. In other words, the delay-lowering effects of pipe-lining become more pronounced as packet size decreases. However, again, as the packet size goes down, effective throughput also goes down due to overhead considerations. In this case, unfortunately, we are trading off delay against effective throughput and it is not possible to define a natural optimum.

In their book, Davies and Barber [6] argue for a certain minimum length "packet" (about 2000 bits) because they have concluded that most of the messages currently exchanged within banks and within airlines will nicely fit in one packet of this size. This does not mean, however, that they believe ARPA Network packets are too small, since they use the term "packet" for the unit of information which is called a "message" in ARPA Network terminology, i.e., the basic unit of information passed from Host to IMP. In fact, an ARPA Network message clearly exceeds this recommended minimum size.

A recent paper on measurements by Kleinrock and Naylor [7] suggested that the ARPA Network packet size was suboptimal and should perhaps be reduced to about 250 bits. This was based on optimization of IMP buffer utilization for the observed traffic mix in the network.

The argument in [7] neglects two important considerations. First, the relative cost of IMP buffer storage and 50Kb circuits is such that one should probably not try to optimize IMP buffer storage. The true trade-off which governs packet size might well be efficient use of phone line bandwidth (driving packets larger) vs. delay characteristics (driving packets smaller). If buffer storage is limiting, perhaps one should just buy more.

Second, given that the IMP has sufficient buffering to handle the phone lines, one can see that this buffer space will often go unused. In fact, since there is a direct relation between storage needed and bandwidth supported, the buffers will go unused except for the case of high bandwidth utilization -- i.e., file transfers. But in this case one would hope the user is sending multi-packet messages. In short, the IMP buffers are set up to handle file transfers -- the hard case -- and coast along under the easy interactive cases currently prevalent in the net.

It might be possible to saturate an IMP with tiny packets -- although it is hard to imagine enough interactive users to do so. If so, the overhead one is paying for the space on the phone lines is so enormous* that there would be tremendous incentive to pack the tiny messages up into longer units -- and we are back to the correct operating point. In summary: being busy implies use of large packets which implies full buffers; when idle the buffer size does not matter.

As is implicit in the above, the choice of packet size is influenced by many factors. Since some of the factors are inherently in conflict, making an optimum difficult to define, much less find, the current ARPA Network packet size is a good compromise. Other packet sizes (e.g., 2000 bits) would also probably work, but there is certainly no alternative choice for packet size which has clear superiority over the 1000-bit size currently used.

7.5 Lockups, Interference, and Other Bugs and Shortcomings

We next turn to a brief point-by-point examination of some flaws and apparent flaws in the ARPA Network. There are problems that can be described as lockups, others which are inefficiencies due to congestion or interference, and others which might be called bugs, oversights, or shortcomings.

---
* Up to a factor of 100 times the true data rate.

One point made by Cerf in [4] is that the system will lead to lockup "because the destination IMP will accept packets which arrive out of order and buffers them until they can be reordered for transmission to the destination Host". This simply is not true. The IMP only accepts packets for which storage has been allocated, or for which storage can be allocated. Since it honors requests for allocation strictly in message number sequence, no lockups can result.

The apparent problem pointed to by Cerf is related to the bug noted by Kleinrock and Opderbeck in [8]. Here if a single-packet message arrives out of order at the destination IMP as a request for storage, in the midst of a series of multi-packet messages, a lockup could result unless the IMP follows this simple rule: a multi-packet allocate must never use up the last free buffer on the IMP. With this rule (subsequently installed in the ARPA Network), there is always room for a single-packet allocate to be returned. With this breathing space, the IMP is able to deal with arbitrarily many single-packet messages which arrive out of order. It can be noted here that the ease with which it was possible to fix this bug (which had never occurred in practice) is a strong argument for keeping the message processing functions in the IMP. We should also mention that this does not represent a flaw in the storage allocation concept but rather a bug in our implementation.

Another problem discovered by Opderbeck [9] is also related to the interaction of the sequencing and flow control mechanisms. This shortcoming of the system does not lead to a lockup, but rather to degraded performance due to retransmissions. Opderbeck shows that if an IMP is sending a continuous stream of single-packet messages to another IMP, and one arrives out of order, it is necessary for the destination IMP to return an ALLOCATE instead of RFNM message. Subsequently, as long as the traffic persists, all messages will appear to be out of order, since they arrive at the destination prior to the retransmitted copy of the previous message. Thus all messages will go through an allocate/retransmission cycle, cutting bandwidth and increasing delay. Several approaches are possible to solve this inefficiency; one is simply to allocate a few buffers in the IMP for accepting out-of-sequence messages, in order to escape from this syndrome when it occurs. To date we have implemented a temporary fix, but we have postponed a final solution to this problem until it is better understood (see Section 2).

A different kind of problem has been noted by us and others in the ARPA Network as a more serious and present difficulty. In exchanging the earlier idea of a link table for a message number table between each pair of IMPs, we coupled the Hosts at an IMP quite tightly. That is, if two Hosts at IMP A communicate with two Hosts at IMP B, they share the same message number sequence. This means they can interfere with each other, if their rates are very different. Further, neither Host has access to the full window of four messages in flight. We knew about this situation when we installed the tables two years ago, and felt it was just one instance of the inescapable fact that all of an IMP's resources are shared among the Hosts to which it is connected, and that it was not worth the extra storage to decouple their message numbers. In the time since that change, the Network has grown dramatically, and we have come to feel that any unnecessary or artificial coupling of the Hosts at an IMP is undesirable. In fact, we are in the process of putting in several explicit measures to provide fairness in resource allocation among the Hosts as discussed in our Quarterly Technical Report No. 6. Foremost among these is a change to allow separate message number sequences for each Host-to-Host pair. Other measures will include checks on the use of storage at the source and destination IMPs so that no one Host can get an unfair share of buffering if there are other Hosts which also need it. We will also continue to organize the processing of messages in the IMP so that no Host has priority over any other.

Another complaint, somewhat less justified in our view, is that the restriction to four messages in flight between two IMPs is limiting the performance of some Hosts. In the case of 8000-bit messages, and a round-trip delay of 1/2 second, this allows a throughput of 64Kbs, which is not unreasonable given the current ARPA Network. The number of messages in flight cannot be extended indefinitely, since the IMPs provide buffering by allocating storage for the messages. Also, there is a significant amount of bookkeeping associated with each message. Nevertheless, we have decided to extend the message number window to eight at the time that we change the system to maintain independent message numbers for each Host pair.

[illegible] IMP

Some Hosts have stated that one aspect of the IMP-Host interface is undesirable, namely that the IMP is free to stop accepting bits from the Host whenever it is not finished processing a message. This happens, for instance, when the source IMP requests storage from the destination IMP for a multi-packet message. No other messages can be sent during this period, since the source IMP deals with one message at a time. The same situation exists when the source IMP needs to acquire a message number when none are free, or to find a free source buffer, and so on.

Hosts which serve one network process have no difficulty here, but Hosts with many users, time-sharing systems for example, would prefer never to have their Host-to-IMP interface stopped. They may have other users to service while the IMP is acquiring the resources necessary to deal with an earlier message. This situation could be corrected by offering an optional message type from the Host to the IMP which says, in effect, "Please take this message now, or else discard it and tell me when you can take it." Alternatively, we could allow the Host to send the IMP a message which is a notice of intent to send, which the IMP will answer with a permit to send, guaranteeing that the IMP will never refuse to take Host messages.

Yet another problem for some Host applications, related to the Host being stopped, is the 30-second incomplete transmission timeout on message numbers. Some Hosts need to have accurate accounting of all messages, while others would prefer to go ahead, giving up on a lost message or reply in much less than 30 seconds. That is, they are willing to trade off a higher rate of lost messages, declaring messages older than a few seconds as lost, for the ability to always send a message every few seconds. Here again, we are considering allowing the Hosts to specify their preference in this regard, once message number sequences are kept separately for each Host pair.

An example of a Host application with special performance requirements is the transmission of real-time, synchronous data such as vocoded speech, the output of physical sensing devices, and so on. These users represent a new demand for network performance -- low delay and high throughput at the same time. The current ARPA Network is optimized to deliver short messages with low delay, and long messages with high throughput. Speech transmission, for instance, requires a "guaranteed" minimum throughput level (depending on the vocoding technique), and a "guaranteed" maximum delay (depending on the tolerance of the people speaking). Users are probably willing to trade a slightly higher message loss rate for the "guarantees". This is a good example of an environment in which reliability, in terms of accurate reporting of message loss, is less important than steady high performance. Of course, the reverse is true in transferring a data file from one system to another. It has been argued that the requirements of such applications as speech are so stringent that the network should do no message processing at all, to avoid introducing artificial performance limits, and leave these functions to the Hosts. We feel this argument is specious, for many of the general reasons stated above about the question of message processing in the subnetwork. Furthermore, we believe we can modify the algorithms in the subnetwork to provide appropriate service to such Hosts, without prejudicing the network services provided to other Hosts (as would happen if the subnetwork abandoned all message processing).

[illegible] IMP

Another area of interest is network interconnection. It can be argued that (1) message processing facilities offered by the several interconnected networks vary greatly and there may be no need for elaborate processing in one network if it is not provided in another; and (2) some functions, such as sequencing, might lead to degraded overall performance if performed in each network (total delay is increased if each net in the path performs sequencing in turn instead of passing unordered data to the ultimate destination for sequencing). We have several reactions to these arguments:

- There are not yet very many packet networks. For all the reasons discussed above, new networks which avoid subnetwork ordering may have more problems than they expect. Some operating experience should perhaps be obtained before concluding that subnetwork ordering is easy to avoid.
- If networks are interconnected where some nets perform ordering on multi-packet messages and other nets do not, it is still feasible to deal with single-packet internetwork traffic without incurring degradation.
- Some networks may not function well if services are removed which do not exist in other networks. For instance, flow control might be important to keep in one network even though some other network does not perform flow control.
- In the ARPANET case, if it is politically or technically desirable to avoid some of the standard message processing for messages that are destined for internetwork transmission, it is relatively easy to allow parallel mechanisms which are subsets of the full message processing facilities. (We are currently implementing such a parallel mechanism to permit experiments of this type).

A different kind of Host application is one that is highly sensitive to errors of any kind, such as program or data file transfers. For these Hosts, a scheme such as end-to-end checksumming might be appropriate, if the cost were not too high. The IMP might be able to interact with such a scheme, to the minimal extent of simply passing a Host-generated checksum along, or to a greater extent by generating and verifying checksums for the Host.

There is a potential use of the ARPA Network which, though desirable from some points of view, is difficult to implement at present. That is the connection of a Host to more than one IMP, for the purpose of increasing reliability. Improving performance by using multiple connections simultaneously is also a possibility. The question of how and where to translate a Host's logical name into one of its several physical addresses can be answered in several ways; we think the subnetwork could easily perform this translation as an extension of the current routing algorithm. Dynamic translation tables could be maintained at all IMPs. A much more difficult problem concerns the message processing for that Host. How can sequencing be performed if different messages (or even packets) of a data stream are sent to different physical addresses? We do not have a good answer as to how to maintain multiple connections of a Host to the Network while maintaining sequencing, error control, and so on. Either these message processing facilities would have to be disabled, or else only one Host connection could be active at any one time, or the translation from physical to logical address would have to be performed by the source Host. Of course, with Host-level ordering, multiple connections pose no additional problems.

The final topic to consider under the general heading of problems with the current ARPA Network approach is that of adapting the various techniques described for use in large networks (hundreds of IMPs, thousands of Hosts). The key point is that the cost of keeping linear tables indexed by IMP, Host, or anything else becomes prohibitive at some network size. The simplicity and reliability of such tables, and their resulting freedom from deadlock, are naturally desirable, but cannot be maintained at arbitrarily high cost. The current tables are structured for a network of up to 63 IMPs, 4 Hosts per IMP (plus 4 fake Hosts internal to each IMP), with line bandwidth of 50Kbs typically, and round-trip delays of roughly 1/2 second. With the growth of the ARPA Network, and the introduction of new technology in IMPs and circuits, including satellites, all the parameters above must be reexamined.

REFERENCES

1. "Basic Elements of a Network Data Link Control Procedure (NDLC)," L. Pouzin, INWG 54, NIC 30375, January 1974.

2. "Presentation and Major Design Aspects of the Cyclades Computer Network," L. Pouzin, Proceedings of the Third ACM Data Communications Symposium, November 1973, pp. 80-88.

3. "A Protocol for Packet Network Intercommunication," V. Cerf and R. Kahn, IEEE Transactions on Communications, Vol. COM-22, No. 5, May 1974, pp. 637-648.

4. "An Assessment of ARPANET Protocols," V. Cerf, RFC 635, NIC 30489, April 1974.

5. "Packet Communication," R.M. Metcalfe, Massachusetts Institute of Technology Project MAC Report MAC TR-114, December 1973.

6. Communication Networks for Computers, D.W. Davies and D.L.A. Barber, London: John Wiley and Sons, 1973.

7. "On Measured Behavior of the ARPA Network," L. Kleinrock and W. Naylor, Proceedings AFIPS 1974 NCC, Vol. 43, pp. 767-780.

8. "On a Possible Lockup Condition in the IMP Subnet Due to Message Sequencing," L. Kleinrock and H. Opderbeck, RFC 626, NIC 22161, March 1974.

9. "Throughput Degradations for Single Packet Messages," H. Opderbeck, RFC 632, NIC 30239, May 1974.