

ARPA

The interface message processor for the ARPA computer network*

by F. E. HEART, R. E. KAHN, S. M. ORNSTEIN, W. R. CROWTHER and D. C. WALDEN

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

For many years, small groups of computers have been interconnected in various ways. Only recently, however, has the interaction of computers and communications become an important topic in its own right.** In 1968, after considerable preliminary investigation and discussion, the Advanced Research Projects Agency of the Department of Defense (ARPA) embarked on the implementation of a new kind of nationwide computer interconnection known as the ARPA Network. This network will initially interconnect many dissimilar computers at ten ARPA-supported research centers with 50-kilobit common-carrier circuits. The network may be extended to include many other locations and circuits of higher bandwidth.

The primary goal of the ARPA project is to permit persons and programs at one research center to access data and use interactively programs that exist and run in other computers of the network. This goal may represent a major step down the path taken by computer time-sharing, in the sense that the computer resources of the various research centers are thus pooled and directly accessible to the entire community of network participants.

Study of the technology and tariffs of available communications facilities showed that use of conventional *line switching* facilities would be economically and technically inefficient. The traditional method of routing information through the common-carrier switched network establishes a dedicated path for each conversation. With present technology, the time required for this task is on the order of seconds. For

voice communication, that overhead time is negligible, but in the case of many short transmissions, such as may occur between computers, that time is excessive. Therefore, ARPA decided to build a new kind of digital communication system employing wideband leased lines and *message switching*, wherein a path is not established in advance and each message carries an address. In this domain the project portends a possible major change in the character of data communication services in the United States.

In a nationwide computer network, economic considerations also mitigate against a wideband leased line configuration that is topologically fully connected. In a non-fully connected network, messages must normally traverse several network nodes in going from source to destination. The ARPA Network is designed on this principle and, at each node, a copy of the message is stored until it is safely received at the following node. The network is thus a store and forward system and as such must deal with problems of routing, buffering, synchronization, error control, reliability, and other related issues. To insulate the computer centers from these problems, and to insulate the network from the problems of the computer centers, ARPA decided to place identical small processors at each network node, to interconnect these small processors with leased common-carrier circuits to form a *subnet*, and to connect each research computer center into the net via the local small processor. In this arrangement the research computer centers are called *Hosts* and the small processors are called *Interface Message Processors*, or *IMPs*. (See Figure 1.) This approach divides the genesis of the ARPA Network into two parts: (1) design and implementation of the IMP subnet, and (2) design and implementation of protocols and techniques for the sensible utilization of the network by the Hosts.

Implementation of the subnet involves two major

* This work was sponsored by the Advanced Research Projects Agency under Contract No. DAHC 15-69-C-0179.

** A bibliography of relevant references is included at the end of this paper; a more extensive list may be found in Cuadra, 1968.

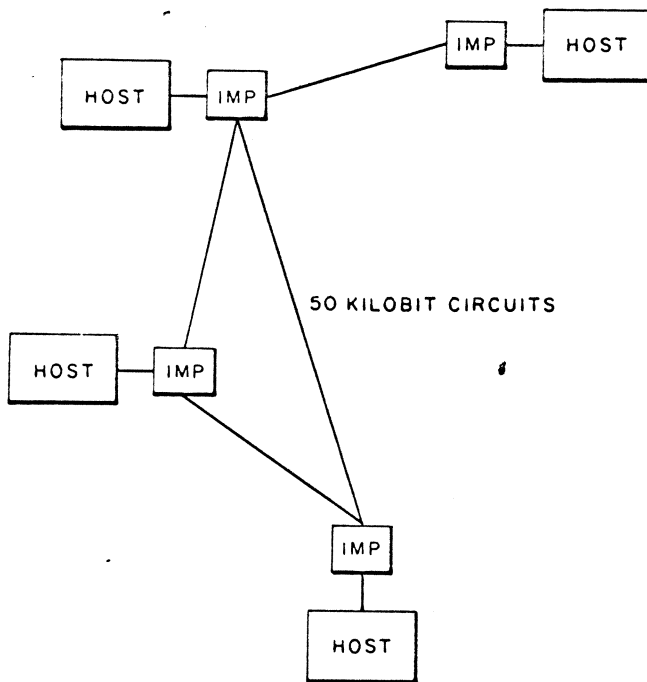


Figure 1—Hosts and IMPs

technical activities: providing 50-kilobit common-carrier circuits and the associated modems; and providing IMPs, along with software and interfaces to modems and Host computers. For reasons of economic and political convenience, ARPA obtained common-carrier circuits directly through government purchasing channels; AT&T (Long Lines) is the central coordinator, although the General Telephone Company is participating at some sites and other common carriers may eventually become involved. In January 1969, Bolt Beranek and Newman Inc. (BBN) began work on the design and implementation of IMPs; a four-node test network was scheduled for completion by the end of 1969 and plans were formulated to include a total of ten sites by mid-1970. This paper discusses the design of the subnet and describes the hardware, the software, and the predicted performance of the IMP. The issues of Host-to-Host protocol and network utilization are barely touched upon; these problems are currently being considered by the participating Hosts and may be expected to be a subject of technical interest for many years to come.

At this time, in late 1969, the test network has become an operating reality. IMPs have already been installed at four sites, and implementation of IMPs for six additional sites is proceeding. The common carriers have installed 50-kilobit leased service con-

necting the first four sites and are preparing to install circuits at six additional sites.

The design of the network allows for the connection of additional Host sites. A map of a projected eleven-node network is shown in Figure 2. The connections between the first four sites are indicated by solid lines. Dotted lines indicate planned connections.

NETWORK DESIGN

The design of the network is discussed in two parts. The first part concerns the relations between the Hosts and the subnet, and the second part concerns the design of the subnet itself.

Host-subnet considerations

The basic notion of a subnet leads directly to a series of questions about the relationship between the Hosts and the subnet: What tasks shall be performed by each? What constraints shall each place on the other? What dependence shall the subnet have on the Hosts? In considering these questions, we were guided by the following principles: (1) The subnet should function as a *communications system* whose essential task is to transfer bits reliably from a source location to a specified destination. Bit transmission should be sufficiently reliable and error free to obviate the need for special precautions (such as storage for retransmission) on the part of the Hosts; (2) The average transit time through the subnet should be under a half second to provide for convenient interactive use of remote computers; (3) The subnet operation should be completely autonomous. Since the subnet must function as a store and forward system, an IMP must not be dependent upon its local Host. The IMP must

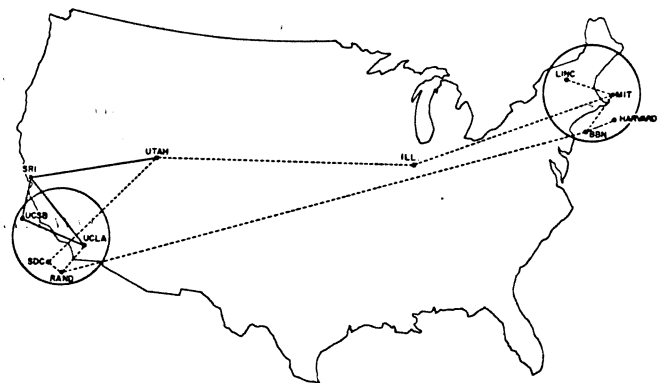


Figure 2—Network map

continue to operate whether the Host is functioning properly or not and must not depend upon a Host for buffer storage or other logical assistance such as program reloading. The Host computer must not in any way be able to change the logical characteristics of the subnet; this restriction avoids the mischievous or inadvertent modification of the communication system by an individual Host user; (4) Establishment of Host-to-Host protocol and the enormous problem of planning to communicate between different computers should be an issue separated from the subnet design.

Messages, links, and RFNMs

In principle, a single transmission from one Host to another may range from a few bits, as with a single teletype character, up to arbitrarily many bits, as in a very long file. Because of buffering limitations in the subnet, an upper limit was placed on the size of an individual Host transmission; 8095 bits was chosen for the maximum transmission size. This Host unit of transmission is called a *message*. The subnet does not impose any pattern restrictions on messages; binary text may be transmitted. Messages may be of variable length; thus, a source Host must indicate the end of a message to the subnet.

A major hazard in a message switched network is congestion, which can arise either due to system failures or to peak traffic flow. Congestion typically occurs when a destination IMP becomes flooded with incoming messages for its Host. If the flow of messages to this destination is not regulated, the congestion will back up into the network, affecting other IMPs and degrading or even completely clogging the communication service. To solve this problem we developed a quenching scheme that limits the flow of messages to a given destination when congestion begins to occur or, more generally, when messages are simply not getting through.

The subnet transmits messages over unidirectional logical paths between Hosts known as *links*. (A link is a conceptual path that has no physical reality; the term merely identifies a message sequence.) The subnet accepts only one message at a time on a given link. Ensuing messages on that link will be blocked from entering the subnet until the source IMP learns that the previous message has arrived at the destination Host. When a link becomes unblocked, the subnet notifies the source Host by sending it a special control message known as *Ready for Next Message* (or RFNM), which identifies the newly unblocked link. The source Host may utilize its connection into the subnet to transmit messages over other links, while waiting to

send messages on the blocked links. Up to 63 separate outgoing links may exist at any Host site. When giving the subnet a message, the Host specifies the destination Host and a link number in the first 32 bits of the message (known as the *leader*). The IMPs then attend to route selection, delivery, and notification of receipt. This use of links and RFNMs also provides for IMP-to-Host delivery of sequences of messages in proper order. Because the subnet allows only one message at a time on a given link, Hosts never receive messages out of sequence.

Host-IMP interfacing

Each IMP will initially service a single Host. However, we have made provision (both in the hardware and software) for the IMP to service up to four Hosts, with a corresponding reduction in the number of permitted phone line connections. Connecting an IMP to a wide variety of different Hosts requires a hardware interface, some part of which must be custom tailored to each Host. We decided, therefore, to partition the interface such that a standard portion would be built into the IMP, and would be identical for all Hosts, while a special portion of the interface would be unique to each Host. The interface is designed to allow messages to flow in both directions at once. A bit serial interface was designed partly because it required fewer lines for electrical interfacing and was, therefore, less expensive, and partly to accommodate conveniently the variety of word lengths in the different Host computers. The bit rate requirement on the Host line is sufficiently low that parallel transfers are not necessary.

The Host interface operates asynchronously, each data bit being passed across the interface via a *Ready For Next Bit/There's Your Bit* handshake procedure. This technique permits the bit rate to adjust to the rate of the slower member of the pair and allows necessary interruptions, when words must be stored into or retrieved from memory. The IMP introduces between bits a (manually) adjustable delay that limits the maximum data rate; at present, this delay is set to 10 μ sec. Any delay introduced by the Host in the handshake procedure further slows the rate.

System failure

Considerable attention has been given to the possible effects on a Host of system failures in the subnet. Minor system failures (e.g., temporary line failures) will appear to the Hosts only in the form of reduced rate of service. Catastrophic failures may, however, result in the loss of messages or even in the loss of

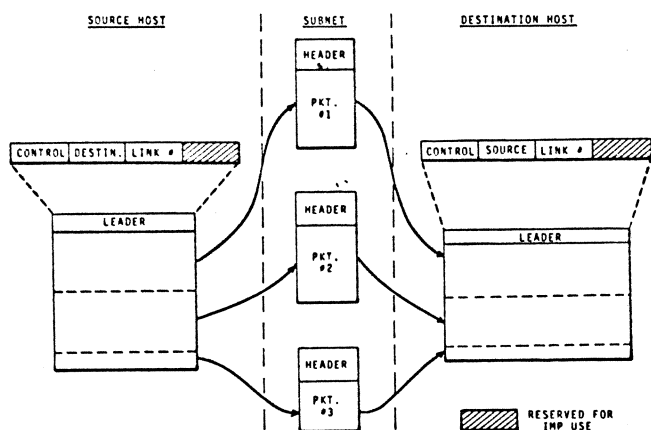


Figure 3—Messages and packets

subnet communication. IMPs inform a Host of all relevant system failures. Additionally, should a Host computer go down, the information is propagated throughout the subnet to all IMPs so they may notify their local Host if it attempts to send a message to that Host.

Specific subnet design

The overriding consideration that guided the subnet design was reliability. Each IMP must operate unattended and reliably over long periods with minimal down time for maintenance and repair. We were convinced that it was important for each IMP in the subnet to operate autonomously, not only independently of Hosts, but insofar as possible from other IMPs as well; any dependency between one IMP and another would merely broaden the area jeopardized by one IMP's failure. The need for reliability and autonomy bears directly upon the form of subnet communication. This section describes the process of message communication within the subnet.

Message handling

Hosts communicate with each other via a sequence of messages. An IMP takes in a message from its Host computer in segments, forms these segments into *packets* (whose maximum size is approximately 1000 bits), and ships the packets separately into the network. The destination IMP reassembles the packets and delivers them in sequence to the receiving Host, who obtains them as a single unit. This segmentation of a message during transmission is completely in-

visible to the Host computers. Figures 3, 4, and 5 illustrate aspects of message handling.

The transmitting Host attaches an identifying leader to the beginning of each message. The IMP forms a *header* by adding further information for network use and attaches this header to each packet of the message.

Each packet is individually routed from IMP-to-IMP through the network toward the destination. At each IMP along the way, the transmitting hardware generates initial and terminal framing characters and parity check digits that are shipped with the packet and are used for error detection by the receiving hardware of the next IMP.

Errors in transmission can affect a packet by destroying the framing and/or by modifying the data content. If the framing is disturbed in any way, the packet either will not be recognized or will be rejected by the receiver. In addition, the check digits provide protection against errors that affect only the data. The check digits can detect all patterns of four or fewer errors occurring within a packet, and any single error burst of a length less than twenty-four bits. An overwhelming majority of all other possible errors (all but about one in 2^{24}) are also detected. Thus, the mean time between undetected errors in the subnet should be on the order of years.

As a packet moves through the subnet, each IMP stores the packet until a positive acknowledgment is returned from the succeeding IMP. This acknowledgment indicates that the message was received without error and was accepted. Once an IMP has accepted a packet and returned a positive acknowledgment, it holds onto that packet tenaciously until it in turn receives an acknowledgment from the succeeding IMP. Under no circumstances (except for Host or IMP malfunction) will an IMP discard a packet after it has generated a positive acknowledgment. However, an IMP is always free to refuse a packet by simply not returning a positive acknowledgment. It may do this for any of several reasons: the packet may have

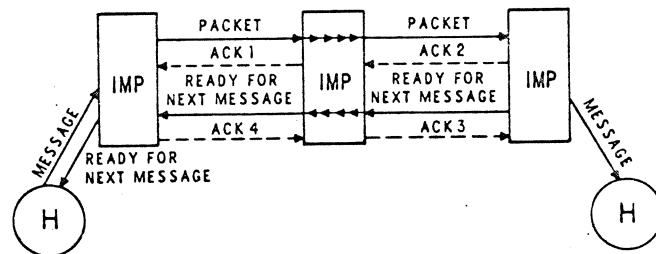


Figure 4—RFNMs and acknowledgments

been received in error, the IMP may be busy, the IMP buffer storage may be temporarily full, etc.

At the transmitting IMP, such discard of a packet is readily detected by the absence of a returned acknowledgment within a reasonable time interval (e.g., 100 msec). Such packets are retransmitted, perhaps along a different route. Acknowledgments themselves are not acknowledged, although they are error checked in the usual fashion. Loss of an acknowledgment results in the eventual retransmission of the packet; the destination IMP sorts out the resulting duplication by using a message number and a packet number in the header.

The packets of a message arrive at the destination IMP, possibly out of order, where they are reassembled. The header is then stripped off each packet and a leader, identifying the source Host and the link, followed by the reassembled message is then delivered to the destination Host as a single unit. See Figure 3.

Routing algorithm

The routing algorithm directs each packet to its destination along a path for which the total estimated transit time is smallest. This path is not determined in advance. Instead, each IMP individually decides onto which of its output lines to transmit a packet addressed to another destination. This selection is made by a fast and simple table lookup procedure. For each possible destination, an entry in the table designates the appropriate next leg. These entries reflect line or IMP trouble, traffic congestion, and current subnet connectivity. This routing table is updated every halfsecond as follows:

Each IMP estimates the delay it expects a packet to encounter in reaching every possible destination over each of its output lines. It selects the minimum delay estimate for each destination and periodically (about twice a second) passes these estimates to its immediate neighbors. Each IMP then constructs its own routing table by combining its neighbors' estimates with its own estimates of the delay to that neighbor. The estimated delay to each neighbor is based upon both queue lengths and the recent performance of the connecting communication circuit. For each destination, the table is then made to specify that selected output line for which the sum of the estimated delay to the neighbor plus the neighbor's delay to the destination is smallest.

The routing table is consistently and dynamically updated to adjust for changing conditions in the network. The system is adaptive to the ups and downs of lines, IMPs, and congestion; *it does not require the*

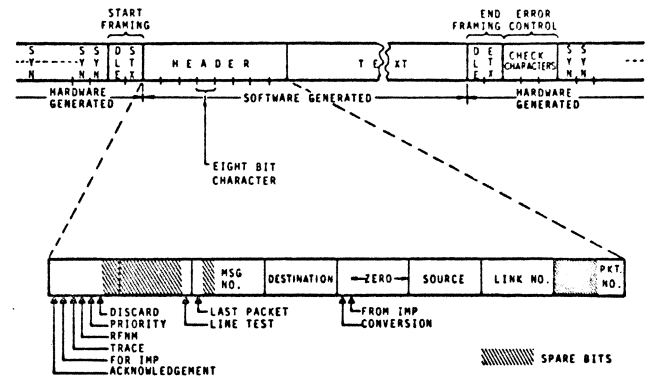


Figure 5—Format of packet on phone line

IMP to know the topology of the network. In particular, an IMP need not even know the identity of its immediate neighbors. Thus, the leased circuits could be reconfigured to a new topology without requiring any changes to the IMPs.

Subnet failures

The network is designed to be largely invulnerable to circuit or IMP failure as well as to outages for maintenance. Special status and test messages are employed to help cope with various failures. In the absence of regular packets for transmission over a line, the IMP program transmits special *hello* packets at half-second intervals. The acknowledgment for a *hello* packet is an *I heard you* packet.

A *dead line* is defined by the sustained absence (approximately 2.5 seconds) on that line of either received regular packets or acknowledgments; no regular packets will be routed onto a dead line, and any packets awaiting transmission will be rerouted. Routing tables in the network are adjusted automatically to reflect the loss. We require acknowledgment of thirty consecutive *hello* packets (an event which consumes at least 15 seconds), before a dead line is defined to be alive once again.

A dead line may reflect trouble either in the communication facilities or in the neighboring IMP itself. Normal line errors caused by dropouts, impulse noise, or other conditions should not result in a dead line, because such errors typically last only a few milliseconds, and only occasionally as long as a few tenths of a second. Therefore, we expect that a line will be defined as dead only when serious trouble conditions occur. If dead lines eliminate all routes between two IMPs, the IMPs are said to be *disconnected* and each

of these IMPs will discard messages destined for the other. Disconnected IMPs cannot be rapidly detected from the delay estimates that arrive from neighboring IMPs. Consequently, additional information is transmitted between neighboring IMPs to help detect this condition. Each IMP transmits to its neighbors the length of the shortest existing path (i.e., number of IMPs) from itself to each destination. To the smallest such received number per destination, the IMP adds one. This incremented number is the length of the shortest path from that IMP to the destination. If the length ever exceeds the number of network nodes, the destination IMP is assumed to be unreachable and therefore disconnected.

Messages intended for dead Hosts (which are not the same as dead IMPs) cannot be delivered; therefore, these messages require special handling to avoid indefinite circulation in the network and spurious arrival at a later time. Such messages are purged from the network either at the source IMP or at the destination IMP. Dead Host information is regularly transmitted with the routing information. A Host computer is notified about another dead Host only when attempting to send a message to that Host.

An IMP may detect a major failure in one of three ways: (1) A packet expected for reassembly of a multiple packet message does not arrive. If a message is not fully reassembled in 15 minutes, the system presumes a failure. The message is discarded by the destination IMP and both the source IMP and the source Host are notified via a special RFNM. (2) The Host does not take a message from its IMP. If the Host has not taken a message after 15 minutes, the system presumes that it will never take the message. Therefore, as in the previous case, the message is discarded and a special RFNM is returned to the source Host. (3) A link is never unblocked. If a link remains blocked for longer than 20 minutes, the system again presumes a failure; the link is then unblocked and an error message is sent to the source Host. (This last time interval is slightly longer than the others so that the failure mechanisms for the first two situations will have a chance to operate and unblock the link.)

Reliability and recovery procedures

For higher system reliability, special attention was placed on intrinsic reliability, hardware test capabilities, hardware/software failure recovery techniques, and proper administrative mechanisms for failure management.

To improve intrinsic reliability, we decided to ruggedize the IMP hardware, thus incurring an approxi-

mately ten percent hardware cost penalty. For ease in maintenance, debugging, program revision, and analysis of performance, all IMPs are as similar as possible; the operational program and the hardware are nearly identical in all IMPs.

To improve hardware test capabilities, we built special *crosspatching* features into the IMP's interface hardware; these features allow program-controlled connection of output lines to corresponding input lines. These crosspatching features have been invaluable in testing IMPs before and during field installation, and they should continue to be very useful when troubles occur in the operating network. These hardware test features are employed by a special hardware test program and may also be employed by the operational program when a line difficulty occurs.

The IMP includes a 512-word block of protected memory that secures special recovery programs. An IMP can recover from an IMP failure in two ways: (1) In the event of power failure, a power-fail interrupt permits the IMP to reach a clean stop before the program is destroyed. When power returns, a special automatic restart feature turns the IMP back on and restarts the program. (We considered several possibilities for handling the packets found in an IMP during a power failure and concluded that no plan to salvage the packets was both practical and foolproof. For example, we cannot know whether the packet in transmission at the time of failure successfully left the machine before the power failed. Therefore, we decided simply to discard all the packets and restart the program.) (2) The second recovery mechanism is a "watchdog timer", which transfers control to protected memory whenever the program neglects this timer for about one minute. In the event of such transfer, the program in unprotected memory is presumed to be destroyed (either through a hardware transient or a software failure). The program in protected memory sends a reload request down a phone line *selected at random*. The neighboring IMP responds by sending a copy of its whole program back on the phone line. A normal IMP would discard this message because it is too long, but the recovering IMP can use it to reload its program.

Everything unique to a particular IMP must thus reside in its protected memory. Only one register (containing the IMP number) currently differs from IMP-to-IMP. The process of reloading, which requires a few seconds, can be tried repeatedly until successful; however, if after several minutes the program has not resumed operation, a later phase of the watchdog timer shuts off all power to the IMP.

In addition to providing recovery mechanisms for both network and IMP failures, we have incorporated

into the subnet a *control center* that monitors network status and handles trouble reports. The control center, located at a network node, initiates and follows up any corrective actions necessary for proper subnet functioning. Furthermore, this center controls and schedules any modifications to the subnet.

Introspection

Because the network is experimental in nature, considerable effort has been allocated to developing tools whereby the network can supply measures of its own performance. The operational IMP program is capable of taking statistics on its own performance on a regular basis; this function may be turned on and off remotely. The various kinds of resulting statistics, which are sent via the network to a selected Host for analysis, include "snapshots", ten-second summaries, and packet arrival times. Snapshots are summaries of the internal status of queue lengths and routing information. A synchronization procedure allows these snapshots, which are taken every half second, to occur at roughly the same time in all network IMPs; a Host receiving such snapshot messages could presumably build up an instantaneous picture of overall network status. Ten-second summaries include such IMP-generated statistics as the number of processed messages of each kind, the number of retransmissions, the traffic to and from the local Host, and so forth; this statistical data is sent to a selected Host every ten seconds. In addition, a record of actual packet arrival times on modem lines allows for the modeling of line traffic. (As part of its research activity, the group at UCLA is acting as a network measurement center; thus, statistics for analysis will normally be routed to the UCLA Host.)

Perhaps the most powerful capability for network introspection is *tracing*. Any Host message sent into the network may have a "trace bit" set in the leader. Whenever it processes a packet from such a message, the IMP keeps special records of what happens to that packet—e.g., how long the packet is on various queues, when it comes in and leaves, etc. Each IMP that handles the traced packet generates special trace report messages that are sent to a specified Host; thus, a complete analysis of what has happened to that message can be made. When used in an orderly way, this tracing facility will aid in understanding at a very detailed level the behavior of routing algorithms and the behavior of the network under changing load conditions.

Flexibility

Flexibility for modifications in IMP usage has been provided by several built-in arrangements: (1) provision within the existing cabinet for an additional 4K core bank; (2) modularity of the hardware interfaces; (3) provision for operation with data circuits of widely different rates; (4) a program organization involving many nearly self-contained subprograms; and (5) provision for Host-unique subprograms in the IMP program structure.

This last aspect of flexibility presents a somewhat controversial design choice. There are many advantages to keeping all IMP software nearly identical. Because of the experimental nature of the network, however, we do not yet know whether this luxury of identical programs will be an optimal arrangement. Several potential applications of "Host-unique" IMP software have been considered—e.g., using ASCII conversion routines in each IMP to establish a "Network ASCII" and possibly to simplify the protocol problems of each Host. As of now, the operational IMP program includes a *structure* that permits unique software plug-in packages at each Host site, but no plug-ins have yet been constructed.

THE HARDWARE

We selected a Honeywell DDP-516 for the IMP processor because we wanted a machine that could easily handle currently anticipated maximum traffic and that had already been proven in the field. We considered only economic machines with fast cycle times and good instruction sets. Furthermore, we needed a machine with a particularly good I/O capability and that was available in a ruggedized version. The geographical proximity of the supplier to BBN was also a consideration.

The basic machine has a 16-bit word length and a 0.96- μ sec memory cycle. The IMP version is packaged in a single cabinet, and includes a 12K memory, a set of 16 multiplexed channels (which implement a 4-cycle data break), a set of 16 priority interrupts, a 100- μ sec clock, and a set of programmable status lights. Also packaged within this cabinet are special modular interfaces for connecting the IMP to phone line modems and to Host computers; these interfaces use the same kind of 1 MHz and 5 MHz DTL packs from which the main machine is constructed. In addition, a number of features that have been incorporated make the IMP somewhat resilient to a variety of failures.

Teletypes and high-speed paper tape readers which are attached to the IMPs are used only for mainte-

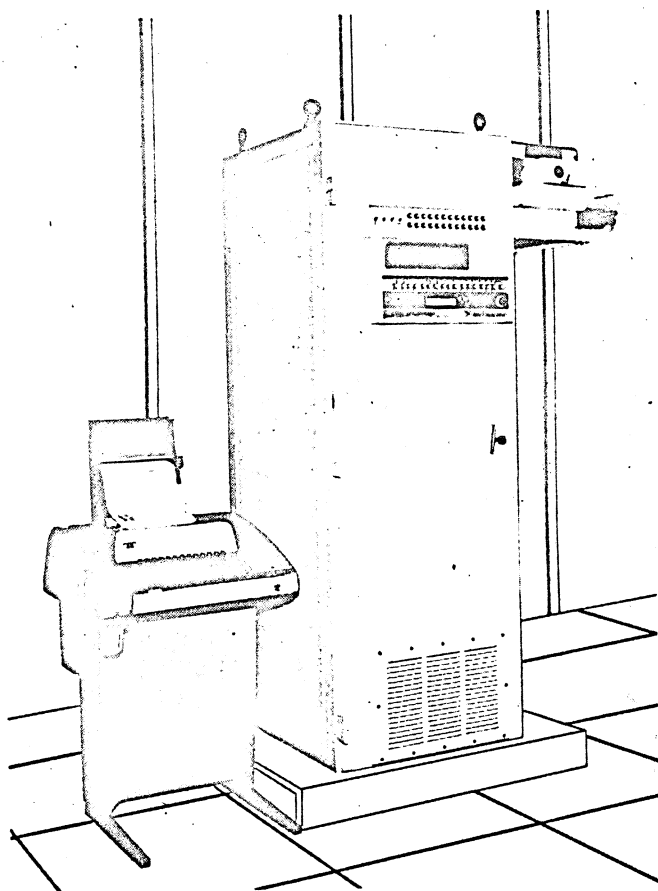


Figure 6—The IMP

nance, debugging, and system modification; in normal operation, the IMP runs without any moving parts except fans. Within the cabinet, space has been reserved for an additional 4K memory. Figure 6 is a picture of an IMP, and Figure 7 shows its configuration.

Ruggedization of computer hardware for use in friendly environments is somewhat unusual; however, we felt that the considerable difficulty that IMP failures can cause the network justified this step. Although the ruggedized unit is not fully "qualified" to MIL specs, it does have greater resistance to temperature variance, mechanical shock and vibration, radio frequency interference, and power line noise. We are confident that this ruggedization will increase the mean time to failure.

Modular Host and modem interfaces allow an IMP to be individually configured for each network node. The modularity, however, does not take the form of pluggable units and, except for the possibility of adding interfaces into reserved frame space, recon-

figuration is impractical. Various configurations allow for up to two Hosts and five modems, three Hosts and four modems, etc. Each modem interface requires approximately one-fourth the amount of logic used in the C.P.U. The Host interface is somewhat smaller (about one-sixth of the C.P.U.).

Interfaces to the Host and to the modems have certain common characteristics. Both are full duplex, both may be crosspatched under program control to test their operation, and both function in the same general manner. To send a packet, the IMP program sets up memory pointers to the packet and then activates the interface via a programmable control pulse. The interface takes successive words from the memory using its assigned output data channel and transmits them bit-serially (to the Host or to the modem). When the memory buffer has thus been emptied, the interface notifies the program via an interrupt that the job has been completed. To receive information, the program first sets pointers to the allocated space in the memory into which the information is to flow. Using a control pulse it then readies the interface to receive. When information starts to arrive (here again bit-serially), it is assembled into 16-bit words and stored into the IMP memory. When either the allocated memory space is full or the end of the data train is detected, the interface notifies the program via an interrupt.

The modem interfaces deal with the phone lines in terms of 8-bit characters; the interfaces idle by sending and receiving a sync pattern that keeps them in character sync. Bit sync is maintained by the modems themselves, which provide both transmit and receive clocking signals to the interfaces. When the program initiates

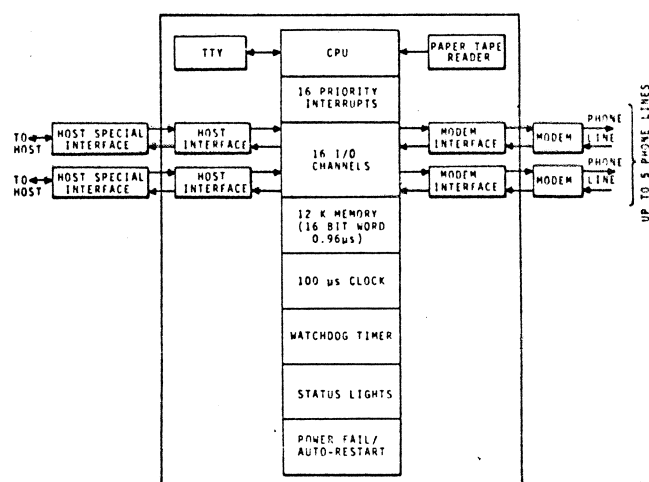


Figure 7—IMP configuration

transmission, the hardware first transmits a pair of initial framing characters (DLE, STX). Next, the text of the packet is taken word by word from the memory and shifted serially onto the phone line. At the end of the data, the hardware generates a pair of terminal framing characters (DLE, ETX) and shifts them onto the phone line. After the terminal framing characters, the hardware generates and transmits 24 check bits. Finally, the interface returns to idle (sync) mode.

The hardware doubles any DLE characters within the binary data train (that is, transmits them twice), thereby permitting the receiving interface hardware to distinguish them from the terminal framing characters and to remove the duplicate. Transmitted packets are of a known maximum size; therefore, any overflow of input buffer length is evidence of erroneous transmission. Format errors in the framing also register as errors. Check bits are computed from the received data and compared with the received check bits to detect errors in the text. Any of these errors set a flag and cause a program interrupt. Before processing a packet, the program checks the error flag to determine whether the packet was received correctly.

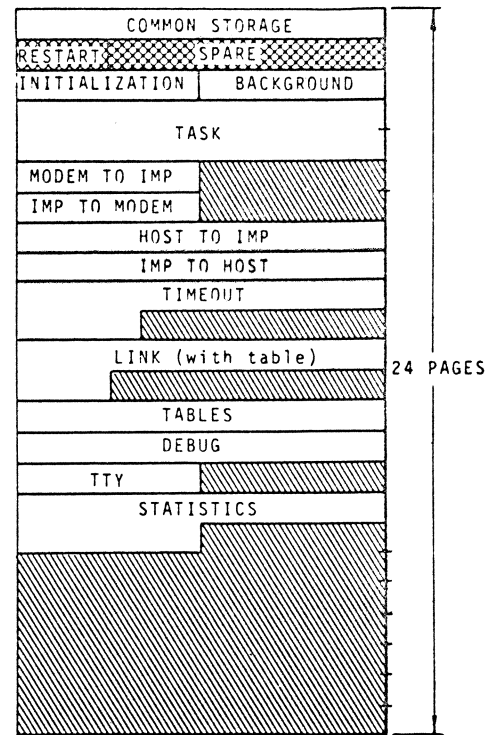
IMP SOFTWARE

Implementation of the IMPs required the development of a sophisticated operational computer program and the development of several auxiliary programs for hardware tests, program construction, and debugging. This section discusses in detail the design of the operational program and briefly describes the auxiliary software.

Operational program

The principal function of the operational program is the processing of packets. This processing includes segmentation of Host messages into packets for routing and transmission, building of headers, receiving, routing and transmitting of store and forward packets, retransmitting of unacknowledged packets, reassembling received packets into messages for transmission to the Host, and generating of RFNMs and acknowledgments. The program also monitors network status, gathers statistics, and performs on-line testing. This real-time program is an efficient, interrupt-driven, involute machine language program that occupies about 6000 words of memory. It was designed, constructed, and debugged over a period of about a year by three programmers.

The entire program is composed of twelve func-



1 PAGE = 512 WORDS

 BUFFER SPACE

 PROTECTED PAGE

Figure 8—Map of core storage

tionally distinct pieces; each piece occupies no more than one or two pages of core (512 words per page). These programs communicate primarily through common registers that reside in page zero of the machine and that are directly addressable from all pages of memory. A map of core storage is shown in Figure 8. Seven of the twelve programs are directly involved in the flow of packets through the IMP: the *task* program performs the major portion of the packet processing, including the reassembly of Host messages; the *modem* programs (IMP-to-Modem and Modem-to-IMP) handle interrupts and resetting of buffers for the modem channels; the *Host* programs (IMP-to-Host and Host-to-IMP) handle interrupts and resetting of buffers for the Host channels, build packet headers during input, and construct RFNMs that are returned to the source Host during output; the *time-out* program maintains a software clock, times out unacknowledged packets for retransmission, and attends to infrequent events; the *link* program assigns and verifies message numbers and keeps track of links. A background loop

TABLE I—Program Data Structures

5000 WORDS—MESSAGE BUFFER STORAGE
120 WORDS—QUEUE POINTERS
300 WORDS—TRACE BLOCKS
100 WORDS—REASSEMBLY BLOCKS
150 WORDS—ROUTING TABLES
400 WORDS—LINK TABLES
300 WORDS—STATISTICS TABLES

contains the remaining five programs and deals with initialization, debugging, testing, statistics gathering and tracing. After a brief description of data structures, we will discuss packet processing in some detail.

Buffer allocation, queues, and tables

The major system data structures (see Table I) consist of buffers and tables. The buffer storage space is partitioned into about 70 fixed length buffers, each of which is used for storing a single packet. An unused buffer is chained onto a free buffer list and is removed from this list when it is needed to store an incoming packet. A packet, once stored in a buffer, is never moved. After a packet has been successfully passed along to its Host or to another IMP, its buffer is returned to the free list. The buffer space is partitioned in such a way that each process (store and forward, traffic, Host traffic, etc.) is always guaranteed some buffers. For the sake of program speed and simplicity, no attempt is made to retrieve the space wasted by partially filled buffers.

In handling store and forward traffic, all processing is on a per packet basis. Further, although traffic to and from Hosts is composed of *messages*, the IMP rapidly converts to dealing with packets; the Host transmits a message as a single unit but the IMP takes it one buffer at a time. As each buffer is filled, the program selects another buffer for input until the entire message has been provided for. These successive buffers will, in general, be scattered throughout the memory. An equivalent inverse process occurs on output to the Host after all packets of the message have arrived at the destination IMP. No attempt is ever made to collect the packets of a message into a contiguous portion of the memory.

Buffers currently in use are either dedicated to an incoming or an outgoing packet, chained on a queue awaiting processing by the program, or being processed. Occasionally, a buffer may be simultaneously found on two queues; this situation can occur when a packet is waiting on one queue to be forwarded and on another to be acknowledged.

There are four principal types of queues:

Task: Packets received on Host channels are placed on the Host task queue. All received acknowledgments, dead Host and routing information, *I heard you* and *hello* packets are placed on the system task queue; all other packets from the modems are placed on the modem task queue. The program services the system task queue first, then the Host task queue, and finally the modem task queue.

Output: A separate output queue is constructed for each modem channel and each Host channel. Each modem output queue is subdivided into an acknowledgment queue, a priority queue, a RFNM queue, and a regular message queue, which are serviced in that order. Each Host output queue is subdivided into a control message queue, a priority queue, and a regular message queue, which are also serviced in the indicated order.

Sent: A separate queue for each modem channel contains packets that have already been transmitted on that line but for which no acknowledgment has yet been received.

Reassembly: The reassembly queue contains those packets that are being reassembled into messages for the Host.

Tables in core are allocated for the storage of queue pointers, for trace blocks, for reassembly information, for statistics, and for links. Most noteworthy of these is the link table, which is used at the source IMP for assignment of message numbers and for blocking and unblocking links, and at the destination IMP to verify message numbers for sequence control.

Packet flow and program structure

Figure 9 is a schematic drawing of packet processing; the processing programs are described below.

The *Host-to-IMP* routine ($H \rightarrow I$) handles messages being transmitted from the local site. The routine uses the leader to construct a header that is prefixed to each packet of the message. It also creates a link for the message if necessary, blocks the link, puts the packets of the message on the Host task queue for further processing by the task routine, and triggers the programmable task interrupt. The routine then acquires a free buffer and sets up a new input. The routine tests a hardware trouble indicator, verifies the message format, and checks whether or not the destination is dead, the link table is full, or the link blocked. The routine is serially reentrant and services all Hosts connected to the IMP.

The *Modem-to-IMP* routine ($M \rightarrow I$) handles inputs from the modems. This routine consists of several identical routines, one for each modem channel. (Such duplication is useful to obtain higher speed.) This routine sets up an input buffer (normally obtained from the free list), places the received packet on the appropriate task queue, and triggers the programmable task interrupt. Should no free buffers be available for input, the buffer at the head of the modem task queue is preempted. If the modem task queue is also empty, the received packet is discarded by setting up its buffer for input. However, a sufficient number of free buffers are specifically reserved to assure that received acknowledgments, routing packets, and the like are rarely discarded.

The *task routine* uses the header information to direct packets to their proper destination. The task routine is driven by the task interrupt, which is set whenever a packet is put on a task queue. The task routine routes packets from the Host task queue onto an output queue determined from the routing algorithm.

For each packet on the modem task queue, the task routine first determines whether sufficient buffer space is available. If the IMP has a shortage of store and forward buffers, the buffers on the modem task queue are simply returned to the free list without further processing. Normally, however, an acknowledgment packet is constructed and put near the front of the appropriate modem output queue. The destination of the packet is then inspected. If the packet is not for the local site, the routing algorithm selects a modem output queue for the packet. If a packet for the local site is a RFNM, the corresponding link is unblocked and the RFNM is put on a queue to the Host. If the packet is not a RFNM, it is joined with others of the

same message on the reassembly queue. Whenever a message is completely reassembled, the packets of the message are put on an output queue to the Host for processing by the IMP-to-Host routine.

In processing the system task queue, the task routine returns to the free list those buffers from the sent queue that have been referenced by acknowledgments. Any packets skipped over by an acknowledgment are designated for retransmission. Routing, *I heard you*, and *hello* packets are processed in a straightforward fashion.

The *IMP-to-Modem* routine ($I \rightarrow M$) transmits successive packets from the Modem output queue. After completing the output, this routine places any packet requiring acknowledgment on the sent queue.

The *IMP-to-Host* routine ($I \rightarrow H$) sets up successive outputs of packets on the Host output queues and constructs a RFNM for each non-control message delivered to a Host. RFNM packets are returned to the system via the Host task queue.

The *time-out* routine is started every 25.6 msec (called the time-out period) by a clock interrupt. The routine has three sections: the fast time-out routine, which “wakes up” any Host or modem interrupt routine that has languished (for example, when the Host input routine could not immediately start a new input because of a shortage in buffer space); the middle time-out routine, which retransmits any packets that have been too long on a modem sent queue; and the slow time-out routine, which marks lines as alive or dead, updates the routing tables and does long term garbage collection of queues and other data structures. (For example, it protects the system from the cumulative effect of such failures as a lost packet of a multiple packet message, where buffers are tied up in message reassembly.) It also deletes links automatically after 15 seconds of disuse, after 20 minutes of blocking, or when an IMP goes down.

These three routines are executed in the following pattern:

FFFF FFFF FFFF FFFF FFFF FFFF ...

M M M M M
S

and, although they run off a common interrupt, are constructed to allow faster routines to interrupt slower ones should a slower routine not complete execution before the next time-out period.

The *link* routine enters, examines, and deletes entries from the link table. A table containing a separate message number entry for many links to every possible Host would be prohibitively large. Therefore, the table contains entries only for each of 63 total out-

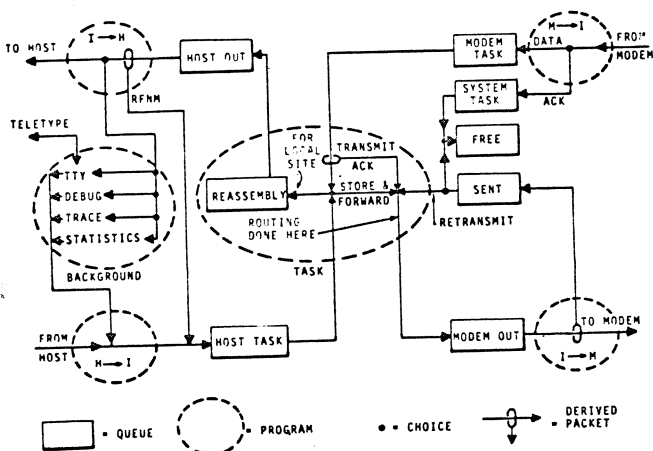


Figure 9—Internal packet flow

going links at any Host site. Hashing is used to speed accessing of this table, but the link program is still quite costly; it uses about ten percent of both speed and space in a conceptually trivial task.

Initialization and background loop

The IMP program starts in an initialization section that builds the initial data structures, prepares for inputs from modem and Host channels, and resets all program switches to their nominal state. The program then falls into the background loop, which is an endlessly repeated series of low-priority subroutines that are interrupted to handle normal traffic.

The programs in the IMP background loop perform a variety of functions: TTY is used to handle the IMP Teletype traffic; DEBUG, to inspect or change IMP core memory; TRACE, to transmit collected information about traced packets; STATISTICS, to take and transmit network and IMP statistics; PARAMETER-CHANGE, to alter the values of selected IMP parameters; and DISCARD, to throw away packets. Selected Hosts and IMPs, particularly the Network Measurement Center and the Network Control Center, will find it necessary or useful to communicate with one or more of these background loop programs. So that these programs may send and receive messages from the network, they are treated as "fake Hosts". Rather than duplicating portions of the large IMP-to-Host and Host-to-IMP routines, the background loop programs are treated as if they were Hosts, and they can thereby utilize existing programs. The "For IMP" bit or the "From IMP" bit in the leader indicates that a given message is for or from a fake Host program in the IMP. Almost all of the background loop is devoted to running these programs.

The TTY program assembles characters from the Teletype into network messages and decodes network messages into characters for the Teletype; TTY's normal message destination is the DEBUG program at its own IMP; however, TTY can be made to communicate with any other IMP Teletype, any other IMP DEBUG program or any Host program with compatible format.

The DEBUG program permits the operational program to be inspected and changed. Although its normal message source is the TTY program at its own IMP, DEBUG will respond to a message of the correct format from any source. This program is normally inhibited from changing the operational IMP program; local operator intervention is required to activate the program's full power.

The STATISTICS program collects measurements

about network operation and periodically transmits them to the Network Measurement Center. This program sends but does not receive messages. STATISTICS has a mechanism for collecting measurements over 10-second intervals and for taking half-second snapshots of IMP queue lengths and routing tables. It can also generate artificial traffic to load the network. When turned on, STATISTICS uses 10 to 20 percent of the machine capacity and generates a noticeable amount of phone line traffic.

Other programs in the background loop drive local status lights and operate the parameter change routine. A thirty-two word parameter table controls the operation of the TRACE and STATISTICS programs and includes spares for expansion; the PARAMETER-CHANGE program accepts messages that change these parameters.

Control organization

It is characteristic of the IMP system that many of the main programs are entered both as subroutine calls from other programs and as interrupt calls from the hardware. The resulting control structure is shown in Figure 10. The programs are arranged in a priority order; control passes upward in the chain whenever a hardware interrupt occurs or the current program decides that the time has come to run a higher priority program, and control passes downward only when the higher priority programs are finished. No program may execute either itself or a lower priority program; however, a program may freely execute a higher priority program. This rule is similar to the usual rules concerning priority interrupt routines.

In one important case, however, control must pass from a higher priority program to a lower priority program—namely, from the several input routines to the TASK routine. For this special case, the computer hardware was modified to include a low-priority hardware interrupt that *can be set by the program*. When this interrupt has been honored (i.e., when all other interrupts have been serviced), the TASK routine is executed. Thus, control is directed where needed without violating the priority rules.

Some routines must occasionally wait for long intervals of time, for example, when the Host-to-IMP routine must wait for a link to unblock. Stopping the whole system would be intolerable; therefore, should the need arise, such a routine is dismissed, and the TIMEOUT routine will later transfer control to the waiting routine.

The control structure and the partition of responsibility among various programs achieve the following

timing goals:

1. No program stops or delays the system while waiting for an event.
2. The program gracefully adjusts to the situation where the machine becomes compute-bound.
3. The Modem-to-IMP routine can deliver its current packet to the TASK routine before the next packet arrives and can always prepare for successive packet inputs on each line. This timing is critical because a slight delay here might require retransmission of the entire packet. To achieve this result, separate routines (one per phone line) interrupt each other freely after new buffers have been set up.
4. The program will almost always deliver packets waiting to be sent as fast as they can be accepted by the phone line.
5. Necessary periodic processes (in the time-out routine) are always permitted to run, and do not interfere with input-output processes.

Support software

Designing a real-time program for a small computer with many high rate I/O channels is a specialized kind of software problem. The operational program requires not only unusual techniques but also extra software tools; often the importance of such extra tools is not recognized. Further, even when these issues are recognized, the effort needed to construct such tools may be seriously underestimated. The development of the IMP system required the following kinds of supporting software:

1. Programs to test the hardware.
2. Tools to help debug the system.
3. A Host simulator.
4. An efficient assembly process.

So far, three hardware test programs have been developed. The first and largest is a complete program for testing all the special hardware features in the IMP. This program permits running any or all of the modem interfaces in a crosspatched mode; it even permits operating together *several* IMPs in a test mode. The second hardware test program runs a detailed phone line test that provides statistics on phone line errors. The final program simulates the modem interface check register whose complex behavior is otherwise difficult to predict.

The software debugging tools exist in two forms. Initially we designed a simple stand-alone debugging program with the capability to do little more than examine and change individual core registers from the

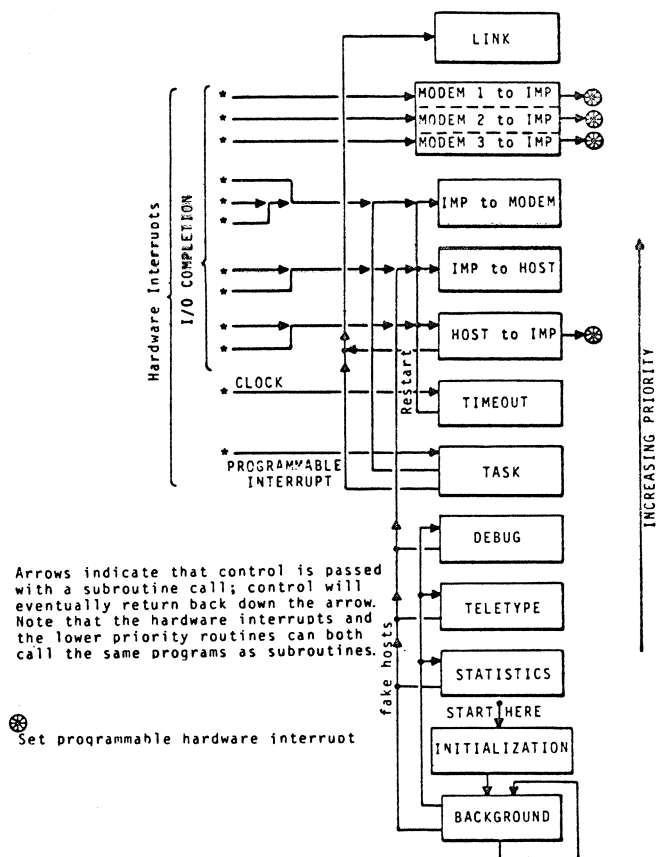


Figure 10—Program control structure

console Teletype. Subsequently, we embedded a version of the stand-alone debugging program into the operational program. This operational debugging program not only provides debugging assistance at a single location but also may be used in *network testing* and *network debugging*.

The initial implementation of the IMP software took place without connecting to a true Host. To permit checkout of the Host-related portions of the operational program, we built a "Host Simulator" that takes input from the console Teletype and feeds the Host routines exactly as though the input had originated in a real Host. Similarly, output messages for a destination Host are received by the simulator and typed out on the console Teletype.

Without recourse to expensive additional peripherals, the assembly facilities on the DDP-516 are inadequate for a large program. (For example, a listing of the IMP program would require approximately 20 hours of Teletype output.) We therefore used other locally available facilities to assist in the assembly process. Specifically, we used a PDP-1 text editor to compose and edit the programs, assembled on the

TABLE II—Transit Times And Message Rates

	<i>Minimum</i>	<i>Maximum</i>
SINGLE WORD MESSAGE		
Transit Time	5 msec	50 msec
Round-trip	10 msec	100 msec
Max. Message Rate/Link	100/sec	10/sec
SINGLE FULL PACKET MESSAGE		
Transit Time	45 msec	140 msec
Round-trip	50 msec	190 msec
Max. Message Rate/Link	20/sec	5/sec
8-PACKET MESSAGE		
Transit Time	265 msec	360 msec
Round-trip	195 msec	320 msec
Max. Message Rate/Link	5/sec	3/sec

DDP-516, and listed the program on the SDS 940 line printer. Use of this assembly process required minor modification of existing PDP-1 and SDS 940 support software

PROJECTED IMP PERFORMANCE

At this writing, the subnet has not yet been subjected to realistic load conditions; consequently, very little experimental data is available. However, we have made some estimates of projected performance of the IMP program and we describe these estimates below.

Host traffic and message delays

In the subnet, the Host-to-Host transit time and the round-trip time (for RFNM receipt) depend upon routing and message length. Since only one message at a time may be present on a given link, the reciprocal of the round-trip delay is the maximum message rate on a link. The primary factors affecting subnet delays are:

- Propagation delay: Electrical propagation time in the Bell system is estimated to be about 10 μ sec per mile. Cross country propagation delay is therefore about 30 msec.
- Modem transmission delay: Because bits enter and leave an IMP at a predetermined modem bit rate, a packet requires a modem transmission time proportional to its length (20 μ sec per bit on a 50-kilobit line).

- Queueing delay: Time spent waiting in the IMP for transmission of previous packets on a queue. Such waiting may occur either at an intermediate IMP or in connection with terminal IMP transmissions into the destination Host.
- IMP processing delay: The time required for the IMP program to process a packet is about 0.35 msec for a store-and-forward packet.

Because the queueing delay depends heavily upon the detailed traffic load in the network, an estimate of queueing delay will not be available until we gain considerable experience with network operation. In Table II, we show an estimate of the one-way and round-trip transit times and the corresponding maximum message rate per link, assuming the negligible queueing delay of a lightly loaded net. In this table, "minimum" delay represents a short hop between two nearby IMPs, and "maximum" delay represents a cross-country path involving five IMPs. In all cases the delays are well within the desired half-second goal.

In a lightly-loaded network with a mixture of nearby and distant destinations, an example of heavy Host traffic into its IMP might be that of 20 links carrying ten single-word messages per second and four more links, each carrying one eight-packet message per second.

Computational load

In general, a line fully loaded with short packets will require more computation than a line with all long packets; therefore the IMP can handle more lines in the latter case. In Figure 11, we show a curve of the computational utilization of the IMP as a function of message length for fully-loaded communication lines. For example, a 50-kilobit line fully loaded in both directions with one-word messages requires slightly over 13 percent of the available IMP time. Since a line will typically carry a variety of different length packets, and each line will be less than fully loaded, the computational load per line will actually be much less.

Throughput is defined to be the maximum number of Host data bits that may traverse an IMP each second. The actual number of bits entering the IMP per second is somewhat larger than the throughput because of such overhead as headers, RFNMs, and acknowledgments. The number of bits on the lines are still larger because of additional line overhead such as framing and error control characters. (Each packet on the phone line contains seventeen characters of

overhead, nine of which are removed before the packet enters an IMP.)

The computational limit on the IMP throughput is approximately 700,000 bits per second. Figure 12 shows maximum throughput as a function of message length. The difference between the throughput curve and the line traffic curve represents overhead.

DISCUSSION

In this section we state some of our conclusions about the design and implementation of the ARPA Network and comment on possible future directions.

We are convinced that use of an IMP-like device is a more sensible way to design networks than is use of direct Host-to-Host connection. First, for the subnet to serve a store-and-forward role, its functions must be independent of Host computers, which may often be down for extended periods. Second, the IMP program is very complex and is highly tailored to the I/O structure of the DDP-516; building such complex functions into special I/O units of each computer that might need network connection is probably economically inadvisable. Third, because of the desirability of having several Host computers at a given site connect to the network, it is both more convenient and more economic to employ IMPs than to provide all the network functions in each of the Host computers. The whole notion of a network node serving a multiplexing function for complexes of local Hosts and terminals lends further support to this conclusion. Finally, because we were led to a design having *some* inter-IMP dependence; we found it advantageous to have *identical* units at each node, rather than computers of different manufacture.

Considering the multiplexing issue directly, it now seems clear that individual network nodes will be connected to a wide variety of computer and terminal complexes. Even the initial ten-node ARPA Network

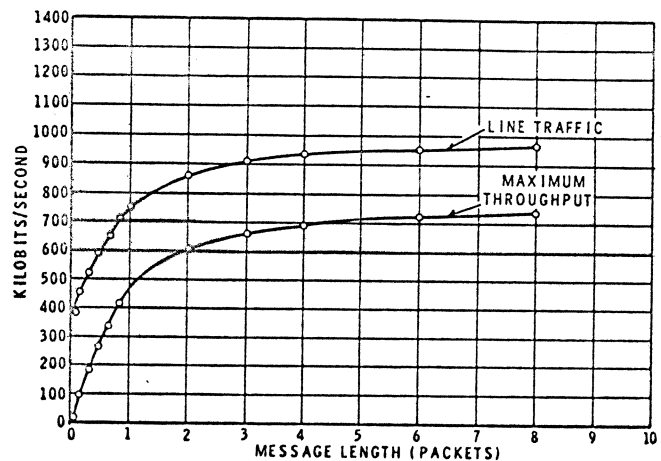


Figure 12—IMP throughput

includes one Host organization that has chosen to submultiplex several computers via a single Host connection to the IMP. We are now studying variants of the IMP design that address this multiplexing issue, and we also expect to cooperate with other groups (such as at the National Physical Laboratory in England) that are studying such multiplexing techniques.

The increasing interest in computer networks will bring with it an expanding interaction between computers and communication circuits. From the outset, we viewed the ARPA Network as a systems engineering problem, including the portion of the system supplied by the common carriers. Although we found the carriers to be properly concerned about circuit performance (the basic circuit performance to date has been quite satisfactory), we found it difficult to work with the carriers *cooperatively* on the technical details, packaging, and implementation of the communication circuit terminal equipment; as a result, the present physical installations of circuit terminal equipment are at best inelegant and inconvenient. In the longer run, for reasons of economy, performance, and reliability, circuit terminal equipment probably should be integrated more closely with computer input/output equipment. If the carriers are unable to participate conveniently in such integrations, we would expect further growth of a competing circuit terminal equipment industry, and more prevalent common carrier provision of bare circuits.

Another aspect of network growth and development is the requirement to connect different rate communication circuits to IMP-like devices as a function of the particular application. In our own IMP design, although there are limitations on total throughput,

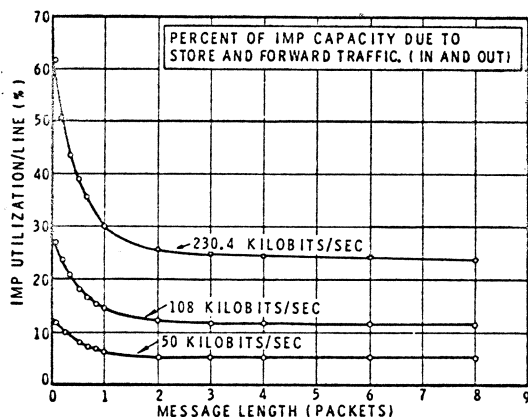


Figure 11—IMP utilization

the IMP can be connected to carrier circuits of any bit rate up to about 250 kilobits; similarly, the interface to a Host computer can operate over a wide range of bit rates. We feel that this flexibility is very important because the economics of carrier offerings, as well as the user requirements, are subject to surprisingly rapid change; even within the time period of the present implementation, we have experienced such changes.

At this point, we would like to discuss certain aspects of the implementation effort. This project required the design, development, and installation of a very complex device in a rather short time scale. The difficulty in producing a complex system is highly dependent upon the number of people who are simultaneously involved. Small groups can achieve complex optimizations of timing, storage, and hardware/software interaction, whereas larger groups can seldom achieve such optimizations on a reasonable time scale. We chose to operate with a very small group of highly talented people. For example, all software, including software tools for assembly, editing, debugging, and equipment testing as well as the main operational program, involved effort by no more than four people at any time. Since so many computer system projects involve much larger groups, we feel it is worth calling attention to this approach.

Turning to the future, we plan to work with the ARPA Network project along several technical directions: (1) the experimental operation of the network and any modifications required to tune its performance; (2) experimental operation of the network with higher bandwidth circuits, e.g., 230.4 kilobits; (3) a review of IMP variants that might perform multiplexing functions; (4) consideration of techniques for designing more economical and/or more powerful IMPs; and (5) participation with the Host organizations in the very sizeable problem of developing techniques and protocols for the effective utilization of the network.

On a more global level, we anticipate an explosive growth of message switched computer networks, not just for the interactive pooling of resources, but for the simple conveniences and economies to be obtained for many classes of digital data communication. We believe that the capabilities inherent in the design of even the present subnet have broad application to other data communication problems of government and private industry.

ACKNOWLEDGMENTS

The ARPA Network has in large measure been the conception of one man, Dr. L. G. Roberts of the

Advanced Research Projects Agency; we gratefully acknowledge his guidance and encouragement. Researchers at many other institutions deserve credit for early interactions with ARPA concerning basic network design; in particular we would like to acknowledge the insight about IMPs provided by W. A. Clark.

At BBN, many persons contributed to the IMP project. We acknowledge the contributions of H. K. Rising, who participated in the subnet design and acted as associate project manager during various phases of the project; B. P. Cosell, who participated significantly in the software implementation; W. B. Barker and M. J. Thrope, who participated significantly in the hardware implementation; and T. Thatch, J. H. Geisman, and R. C. Satterfield, who assisted with various implementation aspects of the project. We also acknowledge the helpful encouragement of J. I. Elkind and D. G. Bobrow.

Finally, we wish to acknowledge the hardware implementation contribution of the Computer Control Division of Honeywell, where many individuals worked cooperatively with us despite the sometimes abrasive pressures of a difficult schedule.

REFERENCES

- 1 P BARAN
On distributed communication networks
IEEE Transactions on Communication Systems Vol CS-12
March 1964
- 2 P BARAN S BOEHM P SMITH
On distributed communications
Series of 11 reports Rand Corporation Santa Monica
California 1964
- 3 B W BOEHM R L MOBLEY
Adaptive routing techniques for distributed communication systems
Rand Corporation Memorandum RM-4781-PR 1966
- 4 *Initial design for interface message processors for the ARPA computer network*
Bolt Beranek and Newman Inc Report No 1763 1969
- 5 *Specifications for the interconnection of a Host and an IMP*
Bolt Beranek and Newman Inc Report No 1822 1969
- 6 G W BROWN J G MILLER T A KEENAN
EDUNET report of the summer study on information networks conducted by the interuniversity communications council
John Wiley and Sons New York 1967
- 7 S CARR S CROCKER V CERF
HOST-HOST communication protocol in the ARPA network
Proceedings of AFIPS SJCC 1970 In this issue
- 8 C A CUADRA
Annual review of information science and technology
Interscience Vol 3 Chapters 7 and 10 1968
- 9 D W DAVIES K A BARTLETT
R A SCANTLEBURY P T WILKINSON
A digital communication network for computers giving rapid response at remote terminals
ACM Symposium on Operating System Principles 1967

- 10 D W DAVIES
The principles of a data communication network for computers and remote peripherals
Proceedings of IFIP Hardware Paper D11 1968
- 11 D W DAVIES
Communications networks to serve rapid-response computers
Proceedings of IFIP Edinburgh 1968
- 12 EIN software catalogue
EDUCOM 100 Charles River Park Boston (Regularly updated)
- 13 R R EVERETT C A ZRAKET H D BENINGTON
Sage—a data processing system for air defense
Proceedings of EJCC 1957
- 14 Policies and regulatory procedures relating to computer and communication services
Notice of Inquiry Docket No 16979 Washington D C 1966
Federal Communications Commission
- 15 L R FORD JR D R FULKERSON
Flows in networks
Princeton University Press 1962
- 16 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
Proceedings of AFIPS SJCC 1970 In this issue
- 17 R T JAMES
The evolution of wideband services
IEEE International Convention Record Part I Wire and Data Communication 1966
- 18 S J KAPLAN
The advancing communication technology and computer communication systems
Proceedings of AFIPS SJCC Vol 32 1968
- 19 L KLEINROCK
Communications nets-stochastic message flow and delay
McGraw-Hill Book Co Inc New York 1964
- 20 L KLEINROCK
Models for computer networks
Proceedings of International Communications Conference June 1969
- 21 L KLEINROCK
Optimization of computer networks for various channel cost functions
Proceedings of AFIPS SJCC 1970 In this issue
- 22 T MARILL
Cooperative networks of time-shared computers
Computer Corporation of America Preliminary Study 1966
- Also Private Report Lincoln Laboratory MIT Cambridge Massachusetts 1966
- 23 T MARILL L G ROBERTS
Toward a cooperative network of time-shared computers
Proceedings of AFIPS FJCC 1966
- 24 Biomedical communications network-technical development Plan
National Library of Medicine June 1968
- 25 Networks of computers symposium NOC-68
Proceedings of Invitational Workshop Ft Meade Maryland National Security Agency September 1969
- 26 Networks of computers symposium NOC-69
Proceedings of Invitational Workshop Ft Meade Maryland (in press) National Security Agency
- 27 M N PERRY W R PLUGGE
American Airlines 'Sabre' electronic reservations system
Proceedings of AFIPS WJCC 1961
- 28 L G ROBERTS
Multiple computer networks and intercomputer communication
ACM Symposium on Operating System Principles 1967
- 29 L G ROBERTS
Access control and file directories in computer networks
IEEE International Convention March 1968
- 30 L G ROBERTS
Resource sharing computer networks
IEEE International Conference March 1969
- 31 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
Proceedings of AFIPS SJCC 1970 In this issue
- 32 R A SCANTLEBURY P T WILKINSON
K A BARTLETT
The design of a message switching centre for a digital communication network
D26 Proceedings of IFIP Hardware Edinburgh 1968
- 33 K STEIGLITZ P WEINER D J KLEITMAN
The design of minimum cost survivable networks
IEEE Transactions on Circuit Theory Vol CT-16 November 1969
- 34 R SUNG J B WOODFORD
Study of communication links for the biomedical communication network
Aerospace Report No ATR-69 (7130-06)-1 1969
- 35 W TEITELMAN R E KAHN
A network simulation and display program
Proceedings of 3rd Annual Princeton Conference on Information Sciences and Systems March 1969

R.N.S.

Improvements in the design and performance of the ARPA network

by J. M. McQUILLAN, W. R. CROWTHER, B. P. COSELL, D. C. WALDEN, and F. E. HEART

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

In late 1968 the Advanced Research Projects Agency of the Department of Defense (ARPA) embarked on the implementation of a new type of computer network which would interconnect, via common-carrier circuits, a number of dissimilar computers at widely separated, ARPA-sponsored research centers. The primary purpose of this interconnection was resource sharing, whereby persons and programs at one research center might access data and interactively use programs that exist and run in other computers of the network. The interconnection was to be realized using wideband leased lines and the technique of message switching, wherein a dedicated path is not set up between computers desiring to communicate, but instead the communication takes place through a sequence of messages each of which carries an address. A message generally traverses several network nodes in going from source to destination, and at each node a copy of the message is stored until it is safely received at the following node.

The ARPA Network has been in operation for over three years and has become a national facility. The network has grown to over thirty sites spread across the United States, and is steadily growing; over forty independent computer systems of varying manufacture are interconnected; provision has been made for terminal access to the network from sites which do not enjoy the ownership of an independent computer system; and there is world-wide excitement and interest in this type of network, with a number of derivative networks in their formative stages. A schematic map of the ARPA Network as of the fall of 1972 is shown in Figure 1.

As can be seen from the map, each site in the ARPA Network consists of up to four independent computer

systems (called Hosts) and one communications processor called an Interface Message Processor, or IMP. All of the Hosts at a site are directly connected to the IMP. Some IMPs also provide the ability to connect terminals directly to the network; these are called Terminal Interface Message Processors, or TIPs. The IMPs are connected together by wideband telephone lines and provide a subnet through which the Hosts communicate. Each IMP may be connected to as many as five other IMPs using telephone lines with bandwidths from 9.6 to 230.4 kilobits per second. The typical bandwidth is 50 kilobits.

During these three years of network growth, the actual user traffic has been light and network performance under such light loads has been excellent. However, experimental traffic, as well as simulation studies, uncovered logical flaws in the IMP software which degraded performance at heavy loads. The software was therefore substantially modified in the spring of 1972. This paper is largely addressed to describing the new approaches which were taken.

The first section of the paper considers some criteria of good network design and then presents our new algorithms in the areas of source-to-destination sequence and flow control, as well as our new IMP-to-IMP acknowledgment strategy. The second section addresses changes in program structure; the third section re-evaluates the IMP's performance in light of these changes. The final section mentions some broader issues.

The initial design of the ARPA Network and the IMP was described at the 1970 Spring Joint Computer Conference,¹ and the TIP development was described at the 1972 Spring Joint Computer Conference.² These papers are important background to a reading of the present paper.

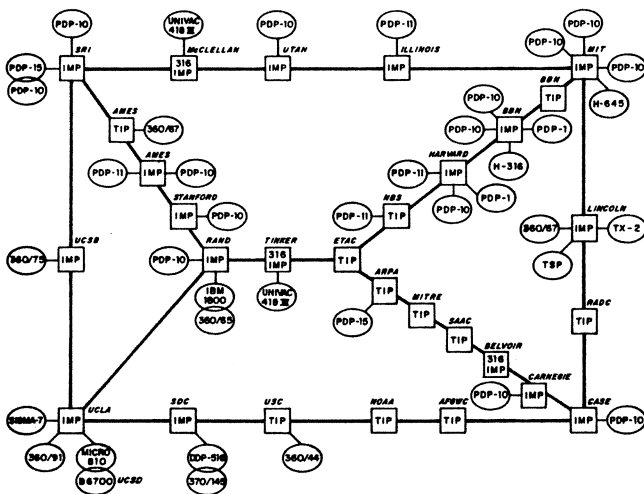


Figure 1—ARPA network, logical map, August 1972

NEW ALGORITHMS

A balanced design for a communication system should provide quick delivery of short interactive messages and high bandwidth for long files of data. The IMP program was designed to perform well under these bimodal traffic conditions. The experience of the first two and one half years of the ARPA Network's operation indicated that the performance goal of low delay had been achieved. The lightly-loaded network delivered short messages over several hops in about one-tenth of a second. Moreover, even under heavy load, the delay was almost always less than one-half second. The network also provided good throughput rates for long messages at light and moderate traffic levels. However, the throughput of the network degraded significantly under heavy loads, so that the goal of high bandwidth had not been completely realized.

We isolated a problem in the initial network design which led to degradation under heavy loads.^{3,4} This problem involves messages arriving at a destination IMP at a rate faster than they can be delivered to the destination Host. We call this *reassembly congestion*. Reassembly congestion leads to a condition we call *reassembly lockup* in which the destination IMP is incapable of passing any traffic to its Hosts. Our algorithm to prevent reassembly congestion and the related sequence control algorithm are described in the following subsections.

We also found that the IMP and line bandwidth requirements for handling IMP-to-IMP traffic could be substantially reduced. Improvements in this area

translate directly into increases in the maximum throughput rate that an IMP can maintain. Our new algorithm in this area is also given below.

Source-to-destination flow control

For efficiency, it is necessary to provide, somewhere in the network, a certain amount of buffering between the source and destination Hosts, preferably an amount equal to the bandwidth of the channel between the Hosts multiplied by the round trip time over the channel. The problem of flow control is to prevent messages from entering the network for which network buffering is not available and which could congest the network and lead to reassembly lockup, as illustrated in Figure 2.

In Figure 2, IMP 1 is sending multi-packet messages to IMP 3; a lockup can occur when all the reassembly buffers in IMP 3 are devoted to partially reassembled messages A and B. Since IMP 3 has reserved all its remaining space for awaited packets of these partially reassembled messages, it can only take in those particular packets from IMP 2. These outstanding packets, however, are two hops away in IMP 1. They cannot get through because IMP 2 is filled with store-and-forward packets of messages C, D, and E (destined for IMP 3) which IMP 3 cannot yet accept. Thus, IMP 3 will never be able to complete the reassembly of messages A and B.

The original network design based source-to-destination sequence and flow control on the link mechanism previously reported in References 1 and 5. Only a single message on a given link was permitted in the subnetwork at one time, and sequence numbers were used to detect duplicate messages on a given link.

We were always aware that Hosts could defeat our flow control mechanism by "spraying" messages over an inordinately large number of links, but we counted on the nonmalicious behavior of the Hosts to keep the

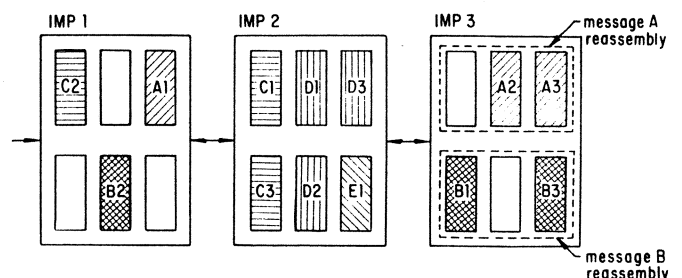


Figure 2—Reassembly lockup

number of links in use below the level at which problems occur. However, simulations and experiments artificially loading the network demonstrated that communication between a pair of Hosts on even a modest number of links could defeat our flow control mechanism; further, it could be defeated by a number of Hosts communicating with a common site even though each Host used only one link. Simulations^{3,4} showed that reassembly lockup may eventually occur when over five links to a particular Host are simultaneously in use. With ten or more links in use with multipacket messages, reassembly lockup occurs almost instantly.

If the buffering is provided in the source IMP, one can optimize for low delay transmissions. If the buffering is provided at the destination IMP, one can optimize for high bandwidth transmissions. To be consistent with our view of a balanced communications system, we have developed an approach to reassembly congestion which utilizes some buffer storage at both the source and destination; our solution also utilizes a request mechanism from source IMP to destination IMP.*

Specifically, no multipacket message is allowed to enter the network until storage for the message has been allocated at the destination IMP. As soon as the source IMP takes in the first packet of a multipacket message, it sends a small control message to the destination IMP requesting that reassembly storage be reserved at the destination for this message. It does not take in further packets from the Host until it receives an allocation message in reply. The destination IMP queues the request and sends the allocation message to the source IMP when enough reassembly storage is free; at this point the source IMP sends the message to the destination.

We maximize the effective bandwidth for sequences of long messages by permitting all but the first message to bypass the request mechanism. When the message itself arrives at the destination, and the destination IMP is about to return the Ready-For-Next-Message (RFNM), the destination IMP waits until it has room for an additional multipacket message. It then piggybacks a storage allocation on the RFNM. If the source Host is prompt in answering the RFNM with its next message, an allocation is ready and the message can be transmitted at once. If the source Host delays too long, or if the data transfer is complete, the source IMP returns the unused allocation to the destination. With this mechanism we have minimized the inter-message delay

and the Hosts can obtain the full bandwidth of the network.

We minimize the delay for a short message by transmitting it to the destination immediately while keeping a copy in the source IMP. If there is space at the destination, it is accepted and passed on to a Host and a RFNM is returned; the source IMP discards the message when it receives the RFNM. If not, the message is discarded, a request for allocation is queued and, when space becomes available, the source IMP is notified that the message may now be retransmitted. Thus, no setup delay is incurred when storage is available at the destination.

The above mechanisms make the IMP network much less sensitive to unresponsive Hosts, since the source Host is effectively held to a transmission rate equal to the reception rate of the destination Host. Further, reassembly lockup is prevented because the destination IMP will never have to turn away a multipacket message destined for one of its Hosts, since reassembly storage has been allocated for each such message in the network.

Source-to-destination sequence control

In addition to its primary function as a flow control mechanism, the link mechanism also originally provided the basis for source-to-destination sequence control. Since only one message was permitted at a time on a link, messages on each link were kept in order; duplicates were detected by the sequence number maintained for each link. In addition, the IMPs marked any message less than 80 bits long as a priority message and gave it special handling to speed it across the network, placing it ahead of long messages on output queues.

The tables associated with the link mechanism in each IMP were large and costly to access. Since the link mechanism was no longer needed for flow control, we felt that a less costly mechanism should be employed for sequence control. We thus decided to eliminate the link mechanism from the IMP subnetwork. RFNMs are still returned to the source Host on a link basis, but link numbers are used only to allow Hosts to identify messages. To replace the per-link sequence control mechanism, we decided upon a sequence control mechanism based on a single logical "pipe" between each source and destination IMP. Each IMP maintains an independent message number sequence for each pipe. A message number is assigned to each message at the source IMP and this message number is checked at the destination IMP. All Hosts at the source and destination IMPs share this message space. Out of an

* This mechanism is similar to that implemented at the level of Host-to-Host protocol,^{6,7,8} indicative of the fact that the same sort of problems occur at every level in a communications system.

eight-bit message number space (large enough to accommodate the settling time of the network), both the source and destination keep a small window of currently valid message numbers, which allows several messages to be in the pipe simultaneously. Messages arriving at a destination IMP with out-of-range message numbers are duplicates to be discarded. The window is presently four numbers wide, which seems about right considering the response time required of the network. The message number serves two purposes: it orders the four messages that can be in the pipe, and it allows detection of duplicates. The message number is internal to the IMP subnetwork and is invisible to the Hosts.

A sequence control system based on a single source/destination pipe, however, does not permit priority traffic to go ahead of other traffic. We solved this problem by permitting two pipes between each source and destination, a priority (or low delay) pipe and a nonpriority (or high bandwidth) pipe. To avoid having each IMP maintain two eight-bit message number sequences for every other IMP in the network, we coupled the low delay and high bandwidth pipe so that duplicate detection can be done in common, thus requiring only one eleven-bit message number sequence for each IMP.

The eleven-bit number consists of a one-bit priority/non-priority flag, two bits to order priority messages, and eight bits to order all messages. For example, if we use the letters A, B, C, and D to denote the two-bit order numbers for priority messages and the absence of a letter to indicate a nonpriority message, we can describe a typical situation as follows: The source IMP sends out nonpriority message 100, then priority messages 101A and 102B, and then nonpriority message 103. Suppose the destination IMP receives these messages in the order 102B, 101A, 103, 100. It passes these messages to the Host in the order 101A, 102B, 100, 103. Message number 100 could have been sent to the destination Host first if it had arrived at the destination first, but the priority messages are allowed to "leapfrog" ahead of message number 100 since it was delayed in the network. The IMP holds 102B until 101A arrives, as the Host must receive priority message A before it receives priority message B. Likewise, message 100 must be passed to the Host before message 103.

Hosts may, if they choose, have several messages outstanding simultaneously to a given destination but, since priority messages can "leapfrog" ahead, and the last message in a sequence of long messages may be short, priority can no longer be assigned strictly on the basis of message length. Therefore, Hosts must explicitly indicate whether a message has priority or not.

With message numbers and reserved storage to be accurately accounted for, cleaning up in the event of a lost message must be done carefully. The source IMP keeps track of all messages for which a RFNM has not yet been received. When the RFNM is not received for too long (presently about 30 seconds), the source IMP sends a control message to the destination inquiring about the possibility of an incomplete transmission. The destination responds to this message by indicating whether the message in question was previously received or not. The source IMP continues inquiring until it receives a response. This technique guarantees that the source and destination IMPs keep their message number sequences synchronized and that any allocated space will be released in the rare case that a message is lost in the subnetwork because of a machine failure.

IMP-to-IMP transmission control

We have adopted a new technique for IMP-to-IMP transmission control which improves efficiency by 10-20 percent over the original separate acknowledge/timeout/retransmission approach described in Reference 1. In the new scheme, which is also used for the Very Distant Host,⁹ and which is similar to Reference 10, each physical network circuit is broken into a number of logical "channels," currently eight in each direction. Acknowledgments are returned "piggy-backed" on normal network traffic in a set of acknowledgment bits, one bit per channel, contained in every packet, thus requiring less bandwidth than our original method of sending each acknowledge in its own packet. The size of this saving is discussed later in the paper. In addition, the period between retransmissions has been made dependent upon the volume of new traffic. Under light loads the network has minimal retransmission delays, and the network automatically adjusts to minimize the interference of retransmissions with new traffic.

Each packet is assigned to an outgoing channel and carries the "odd/even" bit for its channel (which is used to detect duplicate packet transmissions), its channel number, and eight acknowledge bits—one for each channel in the reverse direction.

The transmitting IMP continually cycles through its used channels (those with packets associated with them), transmitting the packets along with the channel number and the associated odd/even bit. At the receiving IMP, if the odd/even bit of the received packet does *not* match the odd/even bit associated with the appropriate receive channel, the packet is accepted and the receive odd/even bit is complemented, otherwise the packet is a duplicate and is discarded.

Every packet arriving over a line contains acknowledgements for all eight channels. This is done by copying the *receive* odd/even bits into the positions reserved for the eight acknowledge bits in the control portion of *every* packet transmitted. In the absence of other traffic, the acknowledgements are returned in "null packets" in which *only* the acknowledge bits contain relevant information (i.e., the channel number and odd/even bit are meaningless; null packets are not acknowledged). When an IMP receives a packet, it compares (bit by bit) the acknowledge bits against the *transmit* odd/even bits. For each *match* found, the corresponding channel is marked unused, the corresponding packet is discarded, and the transmit odd/even bit is complemented.

In view of the large number of channels, and the delay that is encountered on long lines, some packets may have to wait an inordinately long time for transmission. We do not want a one-character packet to wait for several thousand-bit packets to be transmitted, multiplying by 10 or more the effective delay seen by the source. We have, therefore, instituted the following transmission ordering scheme: priority packets which have never been transmitted are sent first; next sent are any regular packets which have never been transmitted; finally, if there are no new packets to send, previously transmitted packets which are unacknowledged are sent. Of course, unacknowledged packets are periodically retransmitted even when there is a continuous stream of new traffic.

In implementing the new IMP-to-IMP acknowledgment system, we encountered a race problem. The strategy of continuously retransmitting a packet in the absence of other traffic introduced difficulties which were not encountered in the original system, which

retransmitted only after a long timeout. If an acknowledgment arrives for a packet which is currently being retransmitted, the output routine must prevent the input routine from freeing the packet. Without these precautions, the header and data in the packet could be changed while the packet was being retransmitted, and all kinds of "impossible" conditions result when this "composite" packet is received at the other end of the line. It took us a long time to find this bug!*

PROGRAM STRUCTURE

Implementation of the IMPs required the development of a sophisticated computer program. This program was previously described in Reference 1. As stated then, the principal function of the IMP program is the processing of packets, including the following: segmentation of Host messages into packets; receiving, routing, and transmitting of store-and-forward packets; retransmitting unacknowledged packets; reassembling packets into messages for transmission into a Host; and generating RFNMs and other control messages. The program also monitors network status, gathers statistics, and performs on-line testing. The program was originally designed, constructed, and debugged over a period of about one year by three programmers.

Recently, after about two and one-half years of operation in up to twenty-five IMPs throughout the network, the operational program was significantly modified. The modification implemented the algorithms described in the previous sections, thereby eliminating causes of network lockup and improving the performance of the IMP. The modification also extended the capabilities of the IMP so it can now interface to Hosts over common carrier circuits (a Very Distant Host⁹), efficiently manage buffers for lines with a wide range of speeds, and perform better network diagnostics. After prolonged study and preliminary design,^{3,4} this program revision was implemented and debugged in about nine man months.

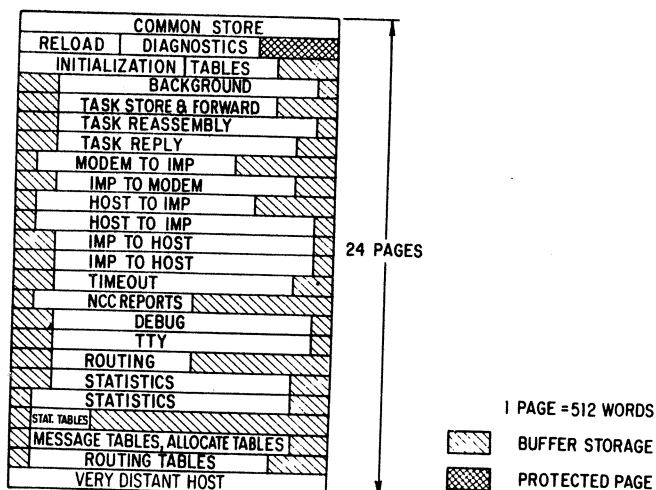


Figure 3—Map of core storage

* Interestingly, a similar problem exists on another level, that of source-destination flow control. If an IMP sends a request for allocation, either single- or multi-packet, to a neighboring IMP, it will periodically retransmit it until it receives an acknowledgment. If it receives an allocation in return, it will immediately begin to transmit the first packet of the message. The implementation in the IMP program sends the request from the same buffer as the first packet, merely marking it with a request bit. If an allocation arrives while the request is in the process of being retransmitted, the program must wait until it has been completely transmitted before it sends the same buffer again as the first packet, since the request bit, the odd/even bit, the acknowledge bits, and the message number (for a multipacket request) will be changed. This was another difficult bug.

We shall emphasize in this section the structural changes the program has recently undergone.

Data structures

Figure 3 shows the layout of core storage. As before, the program is broken into functionally distinct pieces, each of which occupies one or two pages of core. Notice that code is generally centered within a page, and there is code on every page of core. This is in contrast to our previous practice of packing code toward the beginning of pages and pages of code toward the beginning of memory. Although the former method results in a large contiguous buffer area near the end of memory, it has breakage at every page boundary. On the other hand, "centering" code in pages such that there are an integral number of buffers between the last word of code on one page and the first word of code on the next page eliminates almost all breakage.

There are currently about forty buffers in the IMP, and the IMP program uses the following set of rules to allocate the available buffers to the various tasks requiring buffers:

- Each line must be able to get its share of buffers for input and output. In particular, one buffer is always allocated for output on each line, guaranteeing that output is always possible for each line; and double buffering is provided for input on each line, which permits all input traffic to be examined by the program, so that acknowledgments can always be processed, which frees buffers.
- An attempt is made to provide enough store-and-forward buffers so that all lines may operate at full capacity. The number of buffers needed depends directly on line distance and line speed. We currently limit each line to eight or less buffers, and a pool is provided for all lines. Some numerical results on line utilization are presented in a later section. Currently, a maximum of twenty buffers is available in the store-and-forward pool.
- Ten buffers are always allocated to reassembly storage, allowing allocations for one multipacket message and two single-packet messages. Additional buffers may be claimed for reassembly, up to a maximum of twenty-six.

Figure 4 summarizes the IMP table storage. All IMPs have identical tables. The IMP program has twelve words of tables for each of the sixty-four IMPs now possible in the network. The program has ninety-one words of tables for each of the eight Hosts (four

real and four fake) that can be connected; additionally, twelve words of code are replicated for each real Host that can be connected. The program has fifty-five words of tables for each of the five lines that can be connected; additionally, thirty-seven words of code are replicated for each line that can be connected. The program also has tables for initialization, statistics, trace, and so forth.

The size of the initialization code and the associated tables deserves mention. This was originally quite small. However, as the network has grown and the IMP's capabilities have been expanded, the amount of memory dedicated to initialization has steadily grown. This is mainly due to the fact that the IMPs are no longer identical. An IMP may be required to handle a Very Distant Host, or TIP hardware, or five lines and two Hosts, or four Hosts and three lines, or a very high speed line, or, in the near future, a satellite link. As the physical permutations of the IMP have continued to increase, we have clung to the idea that the program should be identical in all IMPs, allowing an IMP to reload its program from a neighboring IMP and providing other considerable advantages. However, maintaining only one version of the program means that the program must rebuild itself during initialization to be the proper program to handle the particular physical configuration of the IMP. Furthermore, it must be able to turn itself back into its nominal form when it is reloaded into a neighbor. All of this takes tables and code. Unfortunately, we did not foresee the proliferation

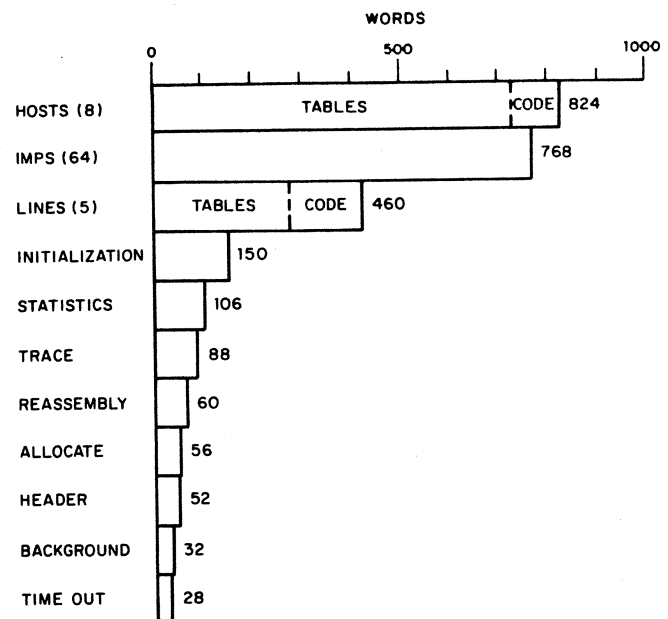


Figure 4—Allocation of IMP table storage

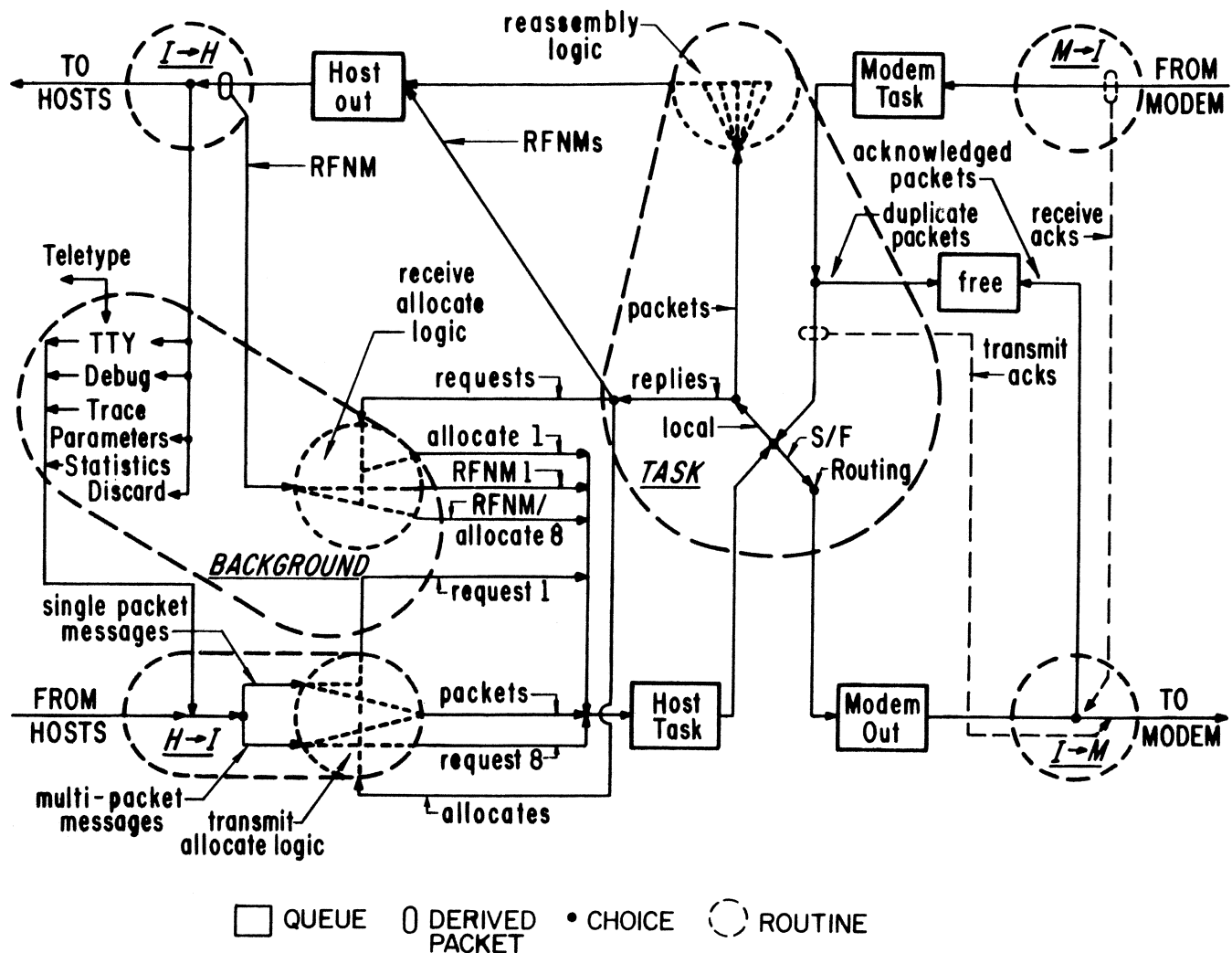


Figure 5—Packet flow and processing

of IMP configurations which has taken place; therefore, we cannot conveniently compute the program differences from a simple configuration key. Instead, we must explicitly table the configuration irregularities.

The packet processing routines

Figure 5 is a schematic drawing of packet flow and packet processing.* We here briefly review the functions of the various packet-processing routines and note important new features.

Host-to-IMP (H→I)

This routine handles messages being transmitted from Hosts at the local site. These Hosts may either be real Hosts or fake Hosts (TTY, Debug, etc.). The routine acquires a message number for each message and passes the message through the transmission allocation logic which requests a reassembly allocation from the destination IMP. Once this allocation is received, the message is broken into packets which are passed to the Task routine via the Host Task queue.

Task

This routine directs packets to their proper destination. Packets for a local Host are passed through the

* Cf. Figure 9 of Reference 1.

reassembly logic. When reassembly is complete, the reassembled message is passed to the IMP-to-Host routine via the Host Out queue. Certain control messages for the local IMP are passed to the transmit or receive allocate logic. Packets to other destinations are placed on a modem output queue as specified by the routing table.

IMP-to-Modem (I→M)

This routine transmits successive packets from the modem output queues and sends piggybacked acknowledgments for packets correctly received by the Modem-to-IMP routine and accepted by the Task routine.

Modem-to-IMP (M→I)

This routine handles inputs from modems and passes correctly received packets to the Task routine via the Modem Task queue. This routine also processes incoming piggybacked acknowledges and causes the buffers for correctly acknowledged packets to be freed.

IMP-to-Host (I→H)

This routine passes messages to local Hosts and informs the background routine when a RFNM should be returned to the source Host.

Background

The function of this routine includes handling the IMP's console Teletype, a debugging program, the statistics programs, the trace program, and several routines which generate control messages. The programs which perform the first four functions run as fake Hosts (as described in Reference 1). These routines simulate the operation of the Host/IMP data channel hardware so the Host-to-IMP and IMP-to-Host routines are unaware they are communicating with anything other than a real Host. This trick saved a large amount of code and we have come to use it more and more. The programs which send incomplete transmission messages, send and return allocations, and send RFNMs also reside in the background program. However, these programs run in a slightly different manner than the fake Hosts in that they do not simulate the Host/IMP channel hardware. In fact, they do not go through the Host/IMP code at all, but rather put their messages directly on the task queue. Nonetheless, the principle is the same.

Timeout

This routine, which is not shown in Figure 5, performs a number of periodic functions. One of these functions is garbage collection. Every table, most queues, and many states of the program are timed out. Thus, if an entry remains in a table abnormally long or if a routine remains in a particular state for abnormally long, this entry or state is garbage-collected and the table or routine is returned to its initial or nominal state. In this way, abnormal conditions are not allowed to hang up the system indefinitely.

The method frequently used by the Timeout routine to scan a table is interesting. Suppose, for example, every entry in a sixty-four entry table must be looked at every now and then. Timeout could wait the proper interval and then look at every entry in the table on one pass. However, this would cause a severe transient in the timing of the IMP program as a whole. Instead, one entry is looked at each time through the Timeout routine. This takes a little more total time but is much less disturbing to the program as a whole. In particular, worst case timing problems (for instance, the processing time between the end of one modem input and the beginning of the next) are significantly reduced by this technique. A particular example of the use of this technique is with the transmission of routing information to the IMP's neighbors. In general, an IMP can have five neighbors. Therefore, it sends routing information to one of its neighbors every 125 msec rather than to all of its neighbors every 625 msec.

In addition to timing out various states of the program, the Timeout routine is used to awaken routines which have put themselves to sleep for a specified period. Typically these routines are waiting for some resource to become available, and are written as co-routines with the Timeout routine. When they are restarted by Timeout the test is made for the availability of the resource, followed by another delay if the resource is not yet available.

PERFORMANCE EVALUATION

In view of the extensive modifications described in the preceding sections, it was appropriate to recalculate the IMP's performance capabilities. The following section presents the results of the reevaluation of the IMP's performance and comparisons with the performance reports of Reference 1.

Throughput vs. message length

In this section we recalculate two measures of IMP performance previously calculated in Reference 1, the

maximum throughput and line traffic. Throughput is the number of Host data bits that traverse an IMP each second. Line traffic is the number of bits that an IMP transmits on its communication circuits per second and includes the overhead of RFNMs, packet headers, acknowledges, framing characters, and checksum characters.

To calculate the IMP's maximum line traffic and throughput, we first calculate the computational load placed on the IMP by the processing of one message. The computational load is the sum of the machine instruction cycles plus the input/output cycles required to process all the packets of a message and their acknowledgments, and the message's RFNM and its acknowledgment. For simplicity in computing the computational load, we ignore the processing required to send and receive the message from a Host since this is only seen by the source and destination IMPs.

A packet has D bits of data, S bits of software overhead, and H bits of hardware overhead. For the original and modified IMP systems, the values of D , S , and H are:

<i>Original</i>	<i>Modified</i>
D 0-1008 bits	0-1008 bits
S 64(packet) + 80(ack) = 144 bits	80 bits(packet+ack)
H 72(packet) + 72(ack) = 144 bits	72 bits(packet+ack)

The input/output processing time for a packet is the time taken to transfer $D+S$ bits from memory to the modem interface at one IMP plus the time to transfer $D+S$ bits into memory at the other IMP. If R is the input/output transfer rate in bits per second,* then the input/output transfer time for a packet is $2(D+S)/R$. Therefore, the total input/output time, I_m , for P packets in a B bit message is $2(B+P \times S)/R$. The input/output transfer time, I_r , for a RFNM is $2S/R$.

To each of these numbers we must add the program processing time, C ; this is about the same for a packet of a message and a RFNM.

* In this calculation we will be making the distinction between the 516 IMP (used originally and reported on in Reference 1) and the 316 IMP (used for all new IMPs). The 516 has a memory cycle time of 0.96 μ sec, and the 316 has a cycle of 1.6 μ sec. The 316 provides a two-cycle data break, in comparison with the four-cycle data break on the 516. Thus, the input/output transfer rates are 16 bits per 3.84 μ sec for the 516 and 16 bits per 3.2 μ sec for the 316.

For the original IMP program, the program processing time per packet consisted of the following:

Modem Output	100 cycles	Send out packet
Modem Input	100 cycles	Receive packet at other IMP
Task	150 cycles	Process it (route onto an output line)
Modem Output	100 cycles	Send back an acknowledgment
Modem Input	100 cycles	Receive acknowledgment at first IMP
Task	150 cycles	Process acknowledgment
	700 cycles	Program processing time per packet

For the modified IMP program, the program processing time consists of:

Modem Output	150 cycles	Send out packet and piggyback acks
Modem Input	150 cycles	Receive packet and process acks
Task	250 cycles	Process packet
	550 cycles	Program processing time per packet

Finally, we add a percentage, V , for overhead for the various periodic processes in the IMP (primarily the routing computation) which take processor bandwidth. V is presently about 5 percent.

We are now in a position to calculate the computational load (in seconds), L , of one P packet message:

$$L = \underbrace{(P \times C + I_m) \times (1 + V)}_{\text{packets}} + \underbrace{(C + I_r) \times (1 + V)}_{\text{RFNM}}$$

The maximum throughput, T , is the number of data bits in a single message divided by the computational loads of the message; that is, $T = B/L$.

The maximum line traffic (in bits per second), R , is the throughput plus the overhead bits for the packets of the message and the RFNM divided by the computational load of the message. That is,

$$R = T + \frac{(P+1) \times (S+H)}{L} = \frac{B + (P+1) \times (S+H)}{L}$$

The maximum throughput and line traffic are plotted for various message lengths in Figure 6 for the original and modified programs and for the 516 IMP and the 316 IMP.

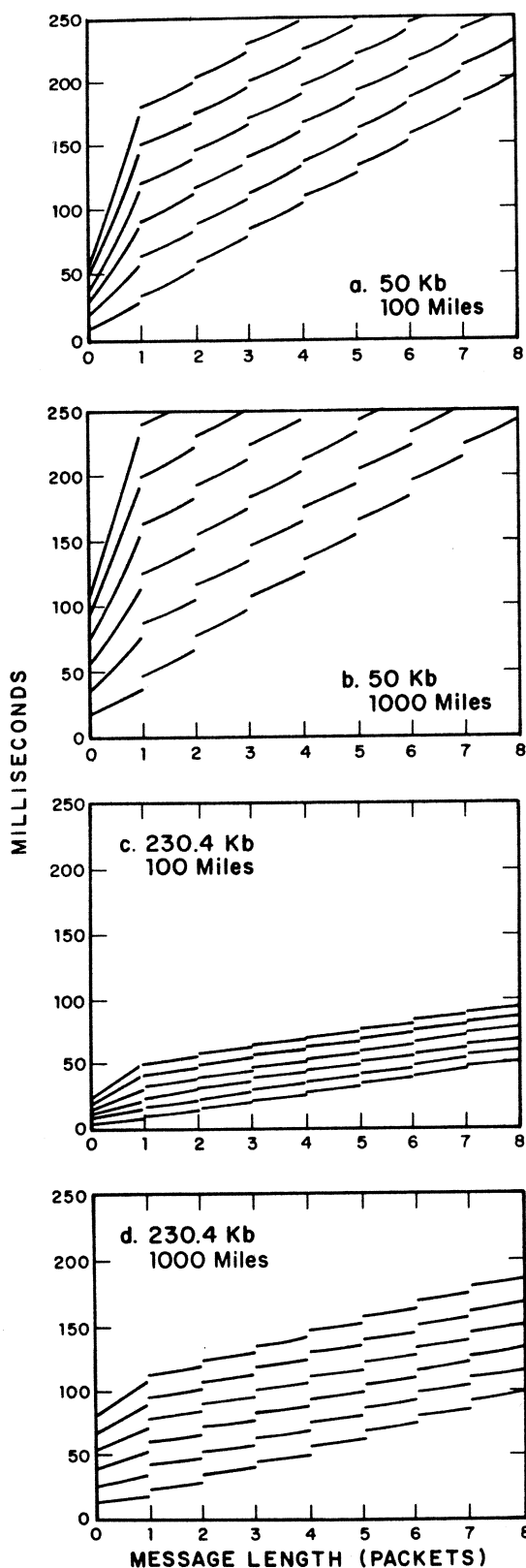


Figure 6—Line traffic and throughput vs. message length. The upper curves plot maximum line traffic, the lower curves plot maximum throughput

The changes to the IMP system can be summarized as follows:

- The program processing time for a store-and-forward packet has been decreased by 20 percent.
- The line throughput has been increased by 4 percent for a 516 IMP and by 7 percent for a 316 IMP.

As a result, the net throughput rate has been increased by 17 percent for a 516 IMP and by 21 percent for a 316 IMP. Thus, a 316 IMP can now process almost as much traffic as a 516 IMP could with the old program. A 516 IMP can now process approximately 850 Kbs.

- The line overhead on a full-length packet has been decreased from 29 percent to 16 percent.

As a result, the effective capacity of the telephone circuits has been increased from thirty-eight full packet messages per second on a 50 Kbs line to forty-three full packet messages per second.

Round trip delay vs. message length

In this section we compute the minimum round trip delay encountered by a message. We define round trip delay as in Reference 1; that is, the delay until the message's RFNM arrives back at the destination IMP. A message has P packets and travels over H hops. The first packet encounters delay due to the packet processing time, C ; the transmission delay, T_P ; and the propagation delay, L . Each successive packet of the message follows $C + T_P$ behind the previous packet. Since the message's RFNM is a single packet message with a transmission delay, T_R , we can write the total delay as

$$\underbrace{H \times (C + T_P + L)}_{\text{first packet}} + \underbrace{(P - 1) \times (C + T_P)}_{\text{successive packets}} + \underbrace{H \times (C + T_R + L)}_{\text{RFNM}}$$

For single packet messages, this reduces to

$$2H(C + L) + H(T_P + T_R)$$

The curves of Figure 7 show minimum round-trip delay through the network for a range of message lengths and hop numbers, and for two sets of line speeds and line lengths. These curves agree with experimental data.^{11,12}

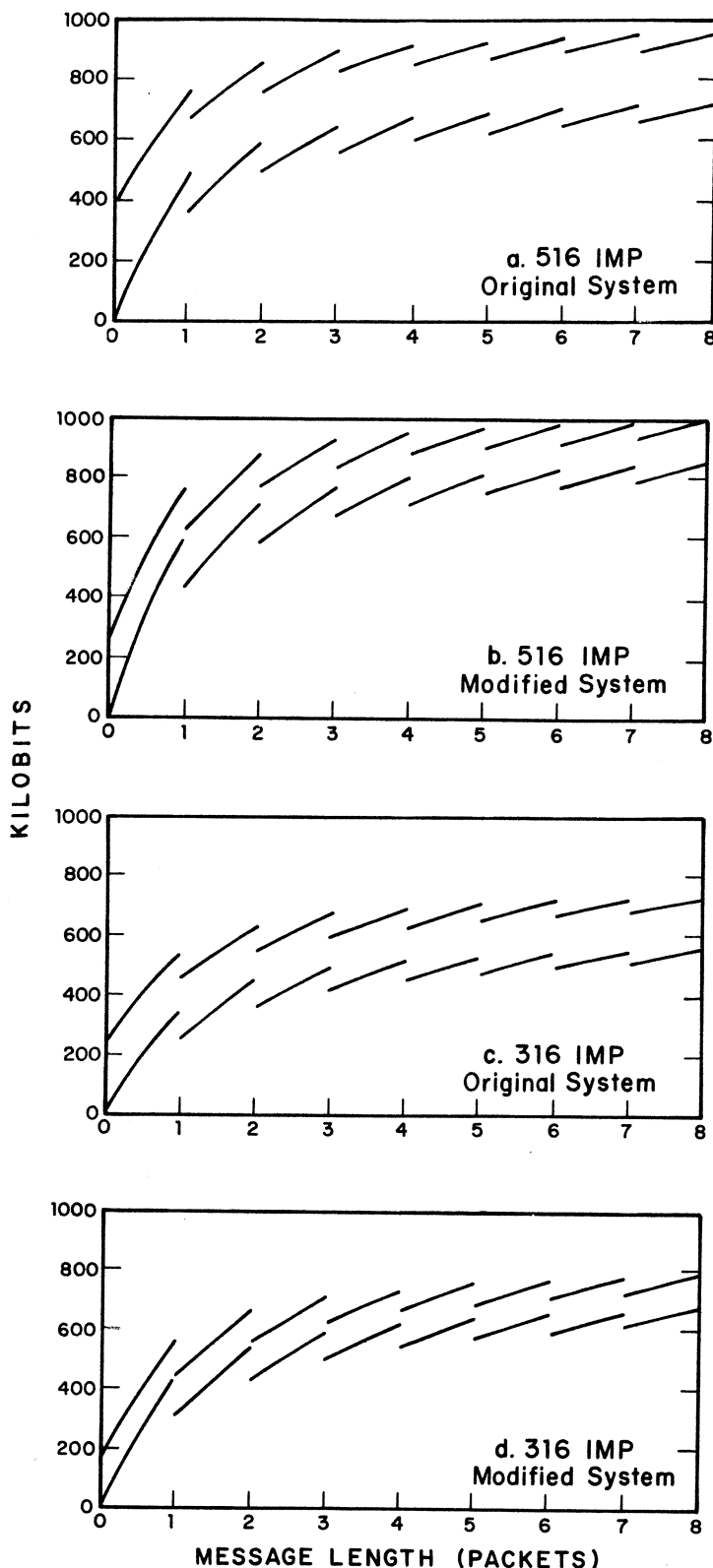


Figure 7—Minimum round trip delay vs. message length.
Curves show delay for 1-6 hops

Line utilization

The number of buffers required to keep a communications circuit fully loaded is a function not only of line bandwidth and distance but also of packet length, IMP delay, and acknowledgment strategy. In order to compute the buffering needed to keep a line busy, we need to know the length of time the sending IMP must wait between sending out a packet and receiving an acknowledgment for it. If we assume no line errors, this time is the sum of: propagation delays for the packet and its acknowledgment, P_P and P_A ; transmission delays for the packet and its acknowledgment, T_P and T_A ; and the IMP processing delay before the acknowledgment is sent. Thus, the number of buffers needed to fully utilize a line is $(P_P + T_P + L + P_A + T_A)/T_P$.

Since $P_P = P_A$, the expression for the number of buffers can be rewritten:

$$\frac{2P}{T_P} + 1 + \frac{L + T_A}{T_P}$$

That is, the number of buffers needed to keep a line full is proportional to the length of the line and its speed, and inversely proportional to the packet size, with the addition of a constant term.

To compute T_P , we must take into account the mix of short and long packets. Thus, we write

$$T_P = \frac{xT_S + yT_L}{x + y}$$

where x to y is the ratio of number of short packets to number of long packets and T_S and T_L are the transmission delays incurred by short and long packets, respectively. The shortest packet permitted is 152 bits long (entirely overhead); the longest packet is 1160 bits long. Computing T_S and T_L for any given line bandwidth is a simple matter; they typically range from 106 μ sec for T_S on a 1.4 Mbs line to 120.5 msec for T_L on a 9.6 Kbs line.

Assuming worst case IMP processing delay (that is, the acknowledge becomes ready for transmission just as the first bit of a maximum length packet is sent), $L = T_L$.

The acknowledge returns in the next outgoing packet at the other IMP, which we assume is of "average" size:*

$$T_A = \frac{T_S + T_L}{2}$$

Propagation delay, P , is essentially just "speed of

* Variations of this assumption have only second order effects on the computation of the number of buffers required.

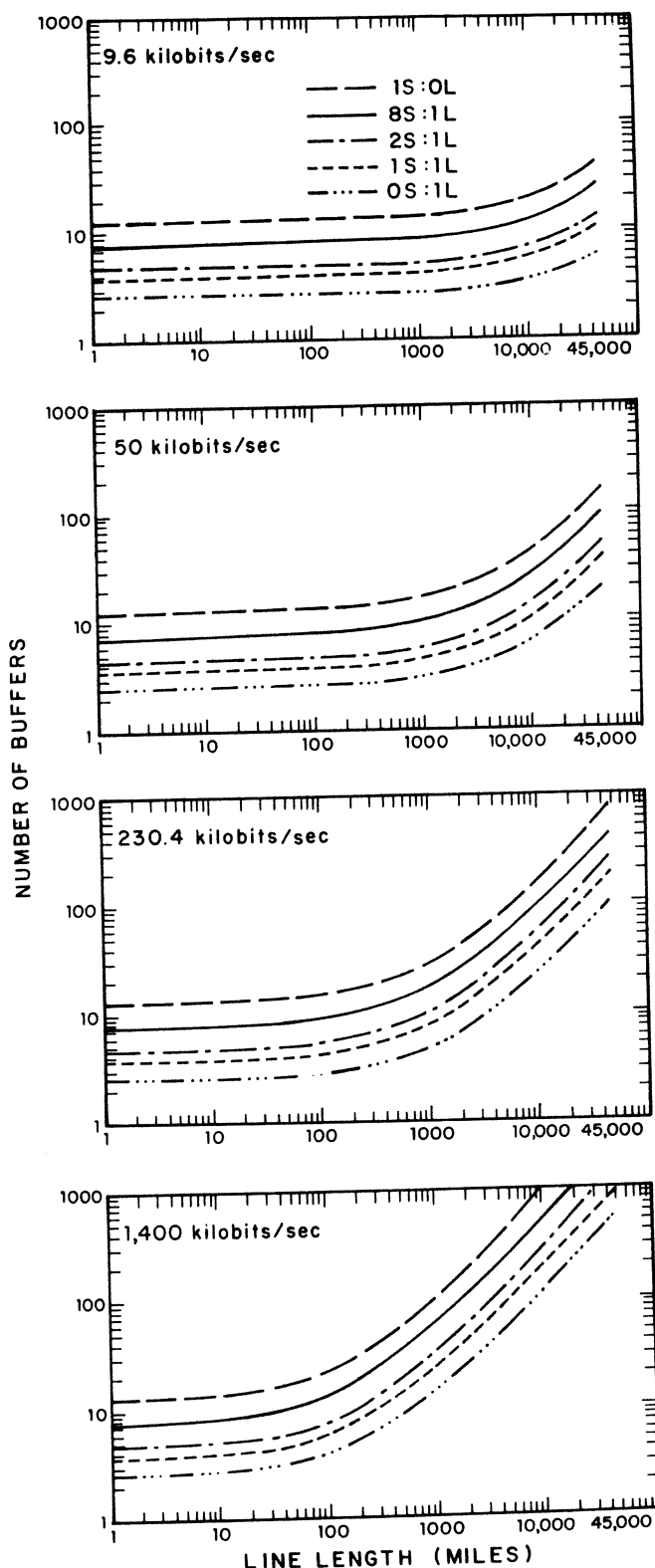


Figure 8—Number of buffers for full line utilization. Traffic mixes are shown as the ratio of number of short packets (S) to number of long packets (L)

light" delay, and ranges from 50 μ sec for short lines, through 20 msec for a cross country line, to 275 msec for a satellite link.

We can now compute the number of buffers required to fully utilize a line for any line speed, line length, and traffic mix. Figure 8 gives the result for typical speeds, lengths, and mixes. Note that the knee of the curves occurs at progressively shorter distances with increasing line speeds. The constant term dominates the 9.6 Kbs case, and it is almost insignificant for the 1.4 Mbs case. Note also that the separation between members of each family of curves remains constant on the log scale, indicating greatly increased variations with distance.

GENERAL COMMENTS

The ARPA Network has represented a fundamental development in the intersection of computers and communications. Many derivative activities are proceeding with considerable energy, and we list here some of the important directions:

- The present network is expanding, adding IMP and TIP nodes at rates approaching two per month. Other government agencies are initiating efforts to use the network, and increasing rates of growth are likely. As befits the growing operational character of the ARPA Network, ARPA is making efforts to transfer the network from under ARPA's research and development auspices to an operational agency or a specialized carrier of some sort.
- Technical improvements in the existing network are continuing. Arrangements have now been made to permit Host-IMP connections at distances over 2000 feet by use of common-carrier circuits. Arrangements are being made to allow the connection of remote-job-entry terminals to a TIP. In the software area, the routing algorithms are still inadequate at heavy load levels, and further changes in these algorithms are in progress. A major effort is under way to develop an IMP which can cope with megabit/second circuits and higher terminal throughput. This new "high speed modular IMP" will be based on a minicomputer, multiprocessor design; a prototype will be completed in 1973.
- The network is being expanded to include satellite links to oversea nodes, and an entirely new approach is being investigated for the "multi-access" use of satellite channels by message switched digital communication systems.¹³ This work could

lead to major changes in world-wide digital communications.

- Many similar networks are being designed by other groups, both in the United States and in other countries. These groups are reviewing the myriad detailed design choices that must be made in the design of message switched systems, and a wide understanding of such networks is growing.
- The existence of the ARPA Network is encouraging a serious review of approaches to obtaining new computer resources. It is now possible to consider investing in major resources, because a national, or even international, network clientele is available over which to amortize the cost of such major resources.
- Perhaps most important, the network has catalyzed important computer research into how programs and operating systems should communicate with each other, and this research will hopefully lead to improved use of all computers.

The ARPA Network has been an exciting development, and there is much yet left to learn.

ACKNOWLEDGMENTS

Dr. Lawrence G. Roberts and others in the ARPA office have been a continuing source of encouragement and support. The entire "IMP group" at Bolt Beranek and Newman Inc. has participated in the development, installation, test, and maintenance of the IMP sub-network. In addition, Dr. Robert E. Kahn of Bolt Beranek and Newman Inc. was deeply involved in the isolation of certain network weaknesses and in the formative stages of the corrective algorithms. Alex McKenzie made many useful suggestions during the writing of this paper. Linda Ebersole helped with the production of the manuscript.

REFERENCES

- 1 F E HEART R E KAHN S M ORNSTEIN
W R CROWTHER D C WALDEN
The interface message processor for the ARPA computer network
Proceedings of AFIPS 1970 Spring Joint Computer Conference Vol 36 pp 551-567
- 2 S M ORNSTEIN F E HEART W R CROWTHER
H K RISING S B RUSSELL A MICHEL
The terminal IMP for the ARPA computer network
Proceedings of AFIPS 1972 Spring Joint Computer Conference Vol 40 pp 243-254
- 3 R E KAHN W R CROWTHER
A study of the ARPA network design and performance
Report No 2161 Bolt Beranek and Newman Inc August 1971

- 4 R E KAHN W R CROWTHER
Flow control in a resource sharing computer network
Proceedings of the Second ACM IEEE Symposium on Problems in the Optimization of Data Communications Systems Palo Alto California October 1971 pp 108-116
- 5 F HEART S M ORNSTEIN
Software and logic design interaction in computer networks
Infotech Computer State of the Art Report No 6 Computer Networks 1971
- 6 S CARR S CROCKER V CERF
Host/host protocol in the ARPA network
Proceedings of AFIPS 1970 Spring Joint Computer Conference Vol 36 pp 589-597
- 7 S CROCKER J HEAFNER R METCALFE
J POSTEL
Function-oriented protocols for the ARPA network
Proceedings of AFIPS 1972 Spring Joint Computer Conference Vol 40 pp 271-280
- 8 A MCKENZIE
Host/host protocol for the ARPA network
Available from the Network Information Center as NIC 8246 at Stanford Research Institute Menlo Park California 94025
- 9 *Specifications for the interconnection of a host and an IMP*
Bolt Beranek and Newman Inc Report No 1822 revised April 1972
- 10 K BARTLETT R SCANTLEBURY
P WILKINSON
A note on reliable full-duplex transmission over half duplex links
Communications of the ACM 12 5 May 1969 pp 260-261
- 11 G D COLE
Computer networks measurements techniques and experiments
UCLA-ENG-7165 Computer Science Department School of Engineering and Applied Science University of California at Los Angeles October 1971
- 12 G D COLE
Performance measurements on the ARPA computer network
Proceedings of the Second ACM IEEE Symposium on Problems in the Optimization of Data Communications Systems Palo Alto California October 1971 pp 39-45
- 13 N ABRAMSON
The ALOHA system—Another alternative for computer communications
Proceedings of AFIPS 1970 Fall Joint Computer Conference Vol 37 pp 281-285

SUPPLEMENTARY BIBLIOGRAPHY

(The following describe issues related to, but not directly concerned with, those discussed in the text.)

- H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
Proceedings of AFIPS 1970 Spring Joint Computer Conference Vol 36 pp 581-587
- H FRANK R E KAHN L KLEINROCK
Computer communication network design—Experience with theory and practice
Proceedings of AFIPS 1972 Spring Joint Computer Conference Vol 40 pp 255-270

R.N. §

Issues in packet switching network design*

by W. R. CROWTHER, F. E. HEART, A. A. MCKENZIE, J. M. McQUILLAN, and
D. C. WALDEN

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

The goals of this paper are to identify several of the key design choices that must be made in specifying a packet-switching network and to provide some insight in each area. Through our involvement in the design, evolution, and operation of the ARPA Network over the last five years (and our consulting in the design of several other networks), we have learned to appreciate both the opportunities and the hazards of this new technical domain.

The last year or so has seen a sudden increase in the number of packet-switching networks under consideration worldwide. It is natural that these networks try to improve on the example of the ARPA Network, and therefore that they contain many features different from those of the ARPA Network. We recognize that networks must be designed differently to meet different requirements; nevertheless, we think that it is easy to overlook important aspects of performance, reliability, or cost. It is vital that these issues be adequately understood in the development of very large practical networks—common user systems for hundreds or thousands of Hosts—since the penalties for error are correspondingly great.

Some brief definitions are needed to isolate the kind of computer network under consideration here:

Nodes. The nodes of the network are real-time computers, with limited storage and processing resources, which perform the basic packet-switching functions.

Hosts. The Hosts of the network are the computers, connected to nodes, which are the providers and users of the network services.

Lines. The lines of the network are some type of communications circuit of relatively high bandwidth and reasonably low error rate.

Connectivity. We assume a general, distributed topology in which each node can have multiple paths to other nodes, but not necessarily to all other nodes. Simple networks such as stars or rings are degenerate cases of the general topology we consider.

Message. The unit of data exchanged between source Host and destination Host.

Packet. The unit of data exchanged between adjacent nodes.

Acknowledgment. A piece of control information returned to a source to indicate successful receipt of a packet or message. A packet acknowledgment may be returned from an adjacent node to indicate successful receipt of a packet; a message acknowledgment may be returned from the destination to the source to indicate successful receipt of a message.

Store and Forward Subnetwork. The node stores a copy of a packet when it receives one, forwards it to an adjacent node, and discards its copy only on receipt of an acknowledgment from the adjacent node, a total storage interval of much less than a second.

Packet Switching. The nodes forward packets from many sources to many destinations along the same line, multiplexing the use of the line at a high rate.

Routing Algorithm. The procedure which the nodes use to determine which of the several possible paths through the network will be taken by a packet.

Node-Node Transmission Procedures. The set of procedures governing the flow of packets between adjacent nodes.

Source-Destination Transmission Procedures. The set of procedures governing the flow of messages between source node and destination node.

Host-Node Transmission Procedures. The set of procedures governing the flow of information between a Host and the node to which that Host is directly connected.

Host-Host Transmission Procedures. The set of procedures governing the flow of information between the source Host and the destination Host.

Within the class of network under consideration, there are already several operational networks and many network designs. The ARPA Network¹ is made up of over fifty node computers called IMPs and over seventy Hosts. The Cyclades Network² is a French network consisting of about six nodes and about two Hosts per node. The Societe Internationale de Telecommunication Aeronautique (SITA) Network³ connects centers in eight or so cities mostly in Europe. The European Informatics Network (EIN),⁴ also known as Cost-11, is currently in a design stage and will be a network interconnecting about six computers in several Common Market countries. Some other packet-switching network designs include: Autodin II,⁵ NPL,⁶ PCI,⁷ RCP,⁸ and Telenet.⁷

* This work was supported under Advanced Research Projects Agency Contracts DAHC15-69-C-0179 and F08606-73-C-0027.

Some of the more obvious differences among these networks can be cited briefly. The ARPA Network splits messages into packets up to 1000 bits long; the other networks have 2000-bit packets and no multipacket messages. Hosts connect to a single node in the ARPA Network and SITA; multiple connections are possible in Cyclades and EIN. Dynamic routing is used in the ARPA Network and EIN; a different adaptive method is used in SITA; fixed routing is presently used in Cyclades. The ARPA Network delivers messages to the destination Host in the same sequence as it accepts them from the source Host; Cyclades does not; in EIN it is optional. Clearly, many of the design choices made in these networks are in conflict with each other. The resolution of these conflicts is essential if balanced, high-performance networks are to be planned and built, particularly since many future designs will be intended for larger, less experimental, and more complex networks.

FUNDAMENTAL ISSUES

In this section we define what we believe are fundamental properties and requirements of packet-switching networks and what we believe are the fundamental criteria for measuring network performance.

Network properties and requirements

We begin by giving the properties central to packet-switching 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 between the Hosts with the following characteristics:

- a. Finite, fluctuating delay—A result of the basic line bandwidth, speed of light delays, queueing in the nodes, line errors, etc.
- b. Finite, fluctuating bandwidth—A result of network overhead, line errors, use of the network by many sources, etc.
- c. Finite packet error rate (duplicate or lost packets)—A result of the acknowledgment system in any store-and-forward discipline (this is a different use of the term "error rate" than in traditional telephony). Duplicate packets are caused when a node goes down after receiving a packet and forwarding it without having sent the acknowledgment. The previous node then generates a duplicate with its retransmission of the packet. Packets are lost when a node goes down after receiving a packet and acknowledging it before the successful transmission of the packet to the next node. An attempt to prevent lost and duplicate packets must fail as there is a tradeoff between minimizing duplicate packets and minimizing lost packets. If the nodes avoid duplication of packets whenever possible, more packets are

lost. Conversely, if the nodes retransmit whenever packets may be lost, more packets are duplicated.

- d. Disordering of packets—A property of the acknowledgment and routing algorithms.

These four properties describe what we term the *store-and-forward subnetwork*.

There are also two basic problems to be solved by the source and destination* in the virtual path described above:

- e. Finite storage—A property of the nodes.
- f. Differing source and destination bandwidths—Largely a property of the Hosts.

A slightly different treatment of this subject can be found in Reference 9.

The fundamental requirements for packet-switching networks are dictated by the six properties enumerated above. These requirements include:

- a. Buffering—Buffering is required because it is generally necessary to send multiple data units on a communications path before receiving an acknowledgment. Because of the *finite delay* of the network, it may be desirable to have buffering for multiple packets in flight between source and destination in order to increase throughput. That is, a system without adequate buffering may have unacceptably low throughput due to long delays waiting for acknowledgment between transmissions.

- b. Pipelining—The *finite bandwidth* of the network may necessitate the pipelining of each message flowing through the network by breaking it 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 node in the path results in excessive delay. By breaking the message into packets, the nodes are able to forward the first packet 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 $P * H$, where the proportionality constant is the packet length divided by the transmission rate.**

- c. Error Control—The node-to-node packet processing algorithm must exercise error control; with an acknowledgment system in order to deal with the *finite packet error rate* of the circuits. It must also detect when a circuit becomes unusable, and when to begin to use it again. In the source-to-destination message processing algorithm, the destination may need to exercise some controls to detect missing and duplicated messages or portions of messages, which would appear as incorrect data to the end user. Further, acknowledgments of message delivery or non-delivery may be useful, possibly to trigger retransmission. This mechanism in turn requires error control and retransmission itself, since the delivery reports can be lost

* The question of whether the source and destination nodes or the source and destination Hosts should solve these problems is addressed in a later section.

** See page 90 of Reference 9 for a derivation and more exact result.

or duplicated. The usual technique is to assign some unique number to identify each data unit and to time out unanswered units. The error correction mechanism is invoked infrequently, as it is needed only to recover from node or line failures.

d. Sequencing—Since *packet sequences can be 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 may take place while several messages are in transit. The sequencing mechanism is frequently invoked since it is needed to recover from line errors.

e. Storage allocation—The fact that *storage* in the nodes is *finite* means that both the packet processing and message processing algorithms must exercise control over its use. The storage may be allocated at either the sender or the receiver.

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, with no more messages (packets) accepted after a certain number, or tied to the storage allocation technique, with no more messages (packets) accepted until a certain amount of storage is free, or the rules can be independent of these features.

In satisfying the above six requirements, the algorithm often exercises contention resolution rules to allocate 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 unlock “impossible” deadlocks and other conditions that may result from hardware or software failures.

Network performance goals

Packet-switching communications systems have two fundamental goals in the processing of data—low delay and high throughput. Each message should be handled with a minimum of waiting time, and the total flow of data should be as large as possible. The difference between low delay and high throughput is important. What the network user wants is the completion of his data transmission in the shortest possible time. The time between transmission of the first bit and delivery of the first bit is a function of network delay, while the time between delivery of the first bit and delivery of the last bit is a function of network throughput. For interactive users with short messages, low delay is more important, since there are few bits per message. For the transfer of long data files, high throughput is more important.

There is a fundamental tradeoff between low delay and high throughput, as is readily apparent in considering some of the mechanisms used to accomplish each goal. For low delay, a small packet size is necessary to cut transmission time, to improve the pipelining characteristics, and to shorten queueing latency at each node; furthermore, short queues are desirable. For high throughput, a large packet size is necessary to decrease the circuit overhead in bits per second and the processing overhead per bit. That is, long packets increase the effective circuit bandwidth and nodal processing bandwidth. Also, long queues may be necessary to provide sufficient buffering for full circuit utilization. Therefore, the network may need to employ separate mechanisms if it is to provide low delay for some users and high throughput for others.

To these two goals one must add two other equally important goals, which apply to message processing and to the operation of the network as a whole. First, the network should be cost-effective. Individual message service should have a reasonable cost as measured in terms of utilization of network resources; further, the network facilities, primarily the node computers and the circuits, should be utilized in a cost-effective way. Secondly, the network should be reliable. Messages accepted by the network should be delivered to the destination with a high probability of success. And the network as a whole should be a robust computer communications service, fault-tolerant, and able to function in the face of node or circuit failures.

In summary, we believe that delay, throughput, reliability, and cost are the four criteria upon which packet-switching network designs should be evaluated and compared. Further, it is the combined performance in all four areas which counts. For instance, poor delay and throughput characteristics may be too big a price to pay for “perfect” reliability.

Key design choices

We believe there are three major areas in which the key choices must be made in designing a packet-switching network. First, there is network hardware design, including the node computer, the network circuits, the Host-to-node connections, and overall connectivity. Second, there is store-and-forward subnetwork software design, primarily the routing algorithm and the node-to-node transmission procedures. Third, there is source-to-destination software design, which encompasses end-to-end transmission procedures and the division of responsibility between Hosts and nodes.* These topics are covered in the following sections.

* There are strong interactions between the topics discussed in the second and third areas. The end-to-end traffic requirements of a specific user can only be met if the store-and-forward subnetwork has mechanisms which act in concert with the source-to-destination mechanisms to provide the required performance. Discussion of this interaction, an important consideration in packet-switching network design, is beyond the scope of this paper.

NETWORK HARDWARE DESIGN

In this section we outline some of the design issues associated with the choice of the node computer, the network circuits, the Host-to-node connections, and overall connectivity. Since the factors affecting these choices change rapidly with the introduction of new technology, we discuss only general observations and design questions.

The node computer

The architecture of the node computer is related to several other network design parameters, as detailed below.

Processor

The speed of the processor is important in determining the throughput rates possible in the network. The store-and-forward processing bandwidth of the processor can be computed by counting instructions in the inner loop (see Reference 10 for an example). The source-to-destination processing bandwidth can be calculated in a similar fashion. These rates should be high enough so that the entire bandwidth of the network lines can be used, i.e., so that the node is not a bottleneck. It has been our experience that the speed of the processor and memory is the main factor in this bandwidth calculation; complex or specialized instruction sets are not valuable because simple instructions make up most of the node program.

A different aspect of the node computer which can also affect throughput is its responsiveness. Because circuits are synchronous devices, they require service with very tight time requirements. If the node does not notice that input has completed on a given circuit, and does not prepare for a new input within a given time, the next input arriving on that circuit will be lost. Likewise on output, the node must be responsive in order to keep the circuits fully loaded. This requirement suggests that some form of interrupt system¹ or high-speed polling device¹¹ is necessary to keep response latency low, and that the overhead of an operating system and task scheduler and dispatcher may be prohibitive. Finally, we note that the amount of time required by the node to process input and output is most critical in determining the *minimum* packet size, since it is with packets of this size that the highest packet arrival and departure rates (and thus processing requirements) can be observed. Of course, data buffering in the device interfaces can partially alleviate these problems.

Memory

The speed of memory may be a major determinant of processor speed, thus affecting the node bandwidth. An equally important consideration is memory speed for I/O transfers, since the node's overall bandwidth results from

a division of total memory bandwidth based on some processing time for a given amount of I/O time. First, there is the question of whether the I/O transfers act in a cycle-stealing fashion to slow the processor or whether memory is effectively multi-ported to allow concurrent use. Then there is the issue of contention for memory among the various synchronous I/O devices. In a worst-case scenario, it is possible for all the I/O devices to request a memory transfer at the same instant, which keeps memory continuously busy for some time interval. A key design parameter is the ratio of this time to the available data buffering time of the least tolerant I/O device. This ratio should be less than one, and may therefore determine how much I/O can be connected to the node.

The size of the memory, naturally, is another key parameter. It has been our experience^{1,10} that the program and associated data structures take up the majority of storage in the node. The remainder of memory is devoted to buffering of two kinds: packet buffering between adjacent nodes, and message buffering between source and destination nodes. These requirements can be calculated quite simply in each case as the product of the maximum data rate to be supported times the round trip time (for a returning acknowledgment). In large networks it may be necessary to rely on sophisticated compression techniques to ensure that tables for the routing algorithm, the source-to-destination transmission procedures, and so on, do not require excessive storage.

I/O

The speed of the I/O system has been touched upon above in relation to processor and memory bandwidth. Other factors worth noting are the internal constraints imposed by the I/O system itself—its delay and bandwidth. A different dimension, and one that we have found to be inadequately designed by most manufacturers, is the flexibility and extensibility of the I/O system. Most manufacturers supply only a limited range of I/O options (some of which may be too slow or too expensive to use). Further, only a limited number of each type can be connected. A packet-switching network node requires high performance from the I/O system, both in the number of connections and in their data rates.

General architecture

There are other factors to consider in evaluating or designing a node processor apart from performance in terms of bandwidth and delay. As we mentioned, extensibility in I/O is very important and comparatively rare; it is more common to find memory systems which can be expanded. Processor systems which can be expanded are not at all common, and yet processor bandwidth may be the limiting factor in some node configurations. Without a modular approach allowing processing, memory and I/O

growth, the cost of the node computer can be quite high due to large step functions in component cost.

Another aspect of node computer architecture is its reliability, particularly for large systems with many lines and Hosts. A failure of such a system has a large impact on network performance. We have studied these issues of performance, cost, and reliability of node computers in a packet-switching network, and have developed, under ARPA sponsorship, a new approach to this problem. Our computer, called the Pluribus, is a multiprocessor made up of minicomputers with modular processors, memory, and I/O components, and a distributed bus architecture to interconnect them.¹¹ Because of its specially designed hardware and software features,¹² it promises to be a highly reliable system. We point out that many of these issues of performance, cost, and reliability could become critically important in very large networks serving thousands of Hosts and terminals.

We also note that there are so many stringent technical constraints on the computer that a choice made on other grounds (e.g., expediency, politics), as is common, is particularly unfortunate.

The network circuits

We next consider some of the important characteristics of the circuits used in the network.

Bandwidth

The bandwidth of the network circuits is likely to be their most important characteristic. It defines the traffic-carrying capacity of the network, both in the aggregate and between any given source and destination. What is less obvious is that the bandwidth (and hence the time to clock a packet out onto the line) may be the main factor determining the transit delays in the network. The minimum delay through the network depends mainly on circuit rates and lengths, and additional delays are largely accounted for by queueing delay, which is directly proportional to circuit bandwidth. These two factors lead to the general observation that the faster the network lines, the longer the packet should be, since long packets have less overhead and permit higher throughput, while the added delay due to length is less important at high circuit rates. In addition, more packet and message buffering is required when higher speed circuits are used.

Delay

The major effect of circuits with appreciable delay is that they require more buffering in the nodes to keep them fully loaded. That is, the node must maintain more packets in flight at once over a circuit with longer delay. This effect may be so large (a circuit using a satellite has a delay of a quarter of a second) as to require significantly

more memory in the nodes.¹⁰ This memory is needed at the nodes connected to the circuit to permit sufficient packet buffering for node-to-node transmission using the circuit. The subtle point is that additional buffering is also required at all nodes in the network that may need to maintain high source-to-destination rates over network paths which include this circuit. If they are to provide maximum throughput, they need sufficient message buffering to keep the entire network path fully loaded.

Reliability

Traditionally, the telephone carriers have quoted error rates in the following manner: "No more than an average of 1 bit in 10^6 bits in error." This definition is not entirely adequate for packet switching, though it may be for continuous transmission. For packet switching, the average bit error rate is less interesting than the average packet error rate (packets with one or more bits in error). For example, ten bits in error in every tenth packet is a 10 percent packet error rate, while one bit in error in every packet is a 100 percent packet error rate, yet the two cases have the same bit error rate.

An example of an acceptable statement of error performance would be as follows:

The circuit operates in two modes. Mode 1: no continuous sequence of packet errors longer than two seconds, with the average packet error rate less than one in a thousand. Mode 2: a continuous sequence of errors longer than two seconds with the following frequency distribution:

> 2 seconds	no more often than once per day
> 1 minute	no more often than once per week
> 15 minutes	no more often than once per month
> 1 hour	no more often than once per 3 months
> 6 hours	no more often than once per year
> 1 day	never

While the figures above may seem too stringent in practice, the mode 1 bit error rate is actually quite lax compared to conventional standards. In any case, these are the kinds of behavior descriptions needed for intelligent design of packet-switching network error control procedures. Therefore, it is important that the carriers begin to provide such descriptions.

The packet error rate of a circuit has two main effects. First, if the rate is high enough, it can degrade the effective circuit bandwidth by forcing the retransmission of many packets. While this is basically a problem for the carrier to repair, the network nodes must recognize this condition and decide whether or not to continue to use the circuit. This is a tradeoff between reduced throughput with the circuit and increased delay and less network connectivity without it. Before the circuit can be used, it must be working in both directions for packets and for control information like routing and acknowledgments, and with a sufficiently low packet error rate.

The second effect of the error rate is present even for relatively low error rates. It is necessary to build a very good error-detection system so that the users of the network do not see errors more often than some specified extremely low frequency. That is, the network should detect enough errors so that the effective network error rate is at least an order of magnitude less than the Host error or failure rate. A usual technique here is a cyclic redundancy check on each packet. This checksum should be chosen carefully; to first order, its size does not depend on packet length* and it should be quite large, for example 24 bits for 50-Kbs lines and 32 bits for multi-megabit lines or lines with high error rates.

The Host-to-node connections

We examine the bandwidth and reliability of the Host connection to the network in the next two sections.

Bandwidth

The issues in choosing the bandwidth of the Host connections are similar to those for the network circuits. In addition to establishing an upper bound on the Hosts' throughput, the rate is also an important factor in delay. The delay to send or receive a long message over a relatively slow Host connection may be comparable in magnitude to the network round trip time. To eliminate this problem, and also to allow high peak throughput rates, the Host connection bandwidth should be as high as possible (within the limits of cost-effectiveness), even higher than the average Host throughput would indicate. By the same argument given above for packet size, a higher speed Host connection allows the use of a longer message with less overhead and Host processing per bit and therefore greater efficiency.

Reliability

The reliability of the Host connection is an important aspect of the network design; several points are worth noting. First, the connection should have a packet error rate which is at least as low as the network circuits. This can be accomplished by a highly reliable direct connection locally or by error-detection and retransmission. The use of error control procedures implies that the Host-node transmission procedures resemble the node-node transmission procedures which are discussed in a later section. Second, if the Host application requires extremely high reliability, a Host-to-Host data checksum and message sequence check are both useful for detecting infrequent net-

work failures. Third, if the Host requires uninterrupted network service, and the Host is reliable enough itself to justify such service, multiple connections of the Host to various nodes can improve the availability of the network. This option complicates matters for the source-to-destination transmission procedures in the nodes (e.g., sequencing) since there may be more than one possible destination node serving the Host.

Overall connectivity

The subject of network topology is a complex one,¹³ and we limit ourselves here to a few general observations. In practice, it seems that the connectivity of the nodes in the network should be relatively uniform. It is obvious that nodes with only a single line are to be avoided for reliability considerations but nodes with many circuits also present a reliability problem since they remove so much network connectivity when they are down. We also feel that the direction for future evolution of network geometries will be toward a "central office" kind of layout with relatively fewer nodes and with a high fan-in of nearby Hosts and terminals. This tendency will become more pronounced as higher reliability in the node computer becomes possible, even for large systems. One reason that we favor this approach is that a large node computer presents an increased opportunity for shared use of the node resources (processor and memory) among many different devices leading to a much more efficient and cost-effective implementation. This trend will mean that in the future, even more than now, a key cost of network topology will be the ultimate data connection to the user (Host or terminal), who may be far from the central office. Concentrators and multiplexors have been the traditional solution; in packet-switching networks, a small node computer could fill this function. In conclusion, we see flexibility and extensibility as two key requirements for the node computer. These factors together with increasing performance and fan-in requirements imply a very high reliability standard as well.

STORE-AND-FORWARD SUBNETWORK SOFTWARE DESIGN

We cover two major areas in our discussion of store-and-forward subnetwork software design, the routing algorithm and the node-to-node transmission procedures, both of which are packet-oriented and require no information about messages.

The routing algorithm

The fundamental step in designing a routing algorithm is the choice of the control regime to be used in the operation of the algorithm. *Non-adaptive* algorithms make no real attempt to adjust to changing network conditions; no

* Assuming that the probability of packet error is proportional to the product of packet length and bit error rate, the checksum length should be proportional to the log of the product of the desired time between undetected errors, the bit error rate, and the total bandwidth of all network circuits.

routing information is exchanged by the nodes, and no observations or measurements are made at individual nodes. *Centralized adaptive* algorithms utilize a central authority which dictates the routing decisions to the individual nodes in response to network changes. *Isolated adaptive* algorithms operate independently with each node making exclusive use of local data to adapt to changing conditions. *Distributed adaptive* algorithms utilize internode cooperation and the exchange of information to arrive at routing decisions.*

Non-adaptive algorithms

Under this heading come such techniques as fixed routing, fixed alternate routing, and random routing (also known as flooding or selective flooding).

Simple fixed routing is too unreliable to be considered in practice for networks of more than trivial size and complexity. Any time a single line or node fails, some nodes become unable to communicate with other nodes. In fact, networks utilizing fixed routing always assume manual updates (as necessary) to another fixed routing pattern. However, in practice this means that every routine network component failure becomes a catastrophe for operational personnel, every site spending frantic hours manually reconstructing routing tables.¹⁵

At their best, in the absence of network component failure, fixed routing algorithms are inefficient. While the routing tables can be fixed to be optimal for some traffic flow, fixed routing is inevitably inefficient to the extent that network traffic flows vary from the optimal traffic flow. Unreliability and inefficiency are also characteristic of two alternative techniques to fixed routing which fall under the heading of non-adaptive algorithms: fixed routing with fixed alternate routes and random routing.¹⁴

Non-adaptive algorithms are all extremely simple and can therefore be implemented at low cost. They are thus possibly suitable for hardware implementation, for theoretical analysis and for studying the effects of varying other network parameters and algorithms.

In conclusion, we do not recommend non-adaptive routing for most networks because it is unreliable and inefficient. Despite these drawbacks, many networks have been proposed or begun with non-adaptive routing, generally because it is simpler to implement and to understand. Perhaps this tendency will be reversed as more information about other routing techniques is published and as network technology generally grows more sophisticated.

Centralized adaptive algorithms

In a centralized adaptive algorithm, the nodes send the information needed to make a routing decision to a Routing Control Center (RCC) which dictates its decision back to the nodes for actual use. The advantages claimed for a

centralized algorithm are: (a) the routing computation is simpler to understand than a non-centralized algorithm, and the computation itself can follow one of several well known algorithms, e.g.;¹⁶ (b) the nodes are relieved of the burden and overhead of the routing computation; (c) more nearly optimal routing is possible because of the sophistication that is possible in a centralized algorithm; and (d) routing "loops" (a possible temporary property of distributed algorithms) can be avoided.

Unfortunately, the processor bandwidth utilization at the center is likely to be very heavy. The classical algorithms that a centralized approach might use generally run in time proportional to N^3 (where N is the number of nodes in the network), while their distributed counterparts can run (through parallel execution) in time proportional to N^2 . While it may be a saving to remove computation from the nodes, it may not be possible to perform a cubic computation on a large network in real time on a single computer, no matter how powerful.¹⁴

The claim that more optimal routing is possible with a centralized approach is not true in practice. To have optimal routing, the input information must be completely accurate and up-to-date. Of course, with any realistic centralized algorithm, the input data will no longer be completely accurate when it arrives at the center. Similarly, the output data—the routing decisions—will not go into effect at the nodes until some time after they have been determined at the center.

Distributed routing algorithms, whether fixed random, fixed alternate, or adaptive, may contain temporary loops, that is, a packet may traverse a complete circle while the algorithm adapts (or simulates adaptation in fixed strategies) to network change. Proponents of centralized routing often argue that such loops can best be avoided by centralization of the computation. However, because of the time lags cited above, there may indeed be loops during the time of propagation of a routing update when some nodes have adopted the new routes and other nodes have not.

A centralized routing algorithm has several inherent weaknesses in the updating procedure, the first being unreliability. If the RCC should fail, or the node to which it is connected goes down, or the lines around that node fail, or a set of lines and nodes in the network fail so as to partition the network into isolated components, then some or all of the nodes in the network are without any routing information. Of course, several steps can be taken to improve on the simple centralized policy. First, the RCC can have a backup computer, either doing another task until a RCC failure, or else on hot standby. This is not sufficient to meet the problem of network failures, only local outages, but it is necessary if the RCC computer has any appreciable failure rate. Second, there can be multiple RCCs in different locations throughout the network, and again the extra computers can be in passive or active standby. Here there is the problem of identifying which center is in control of which nodes, since the nodes must know to which center to send their routing input data.

* A much more detailed discussion is given in Reference 14.

A related difficulty with centralized algorithms lies in the fact that when a node or line fails in the network, the failed component may have been on the previously best path between the RCC and the nodes trying to report the failure. In this case, just at the time the RCC needs routes over which to receive and transmit routing information, no routes are available; the availability of new routes requires the very change the RCC is unsuccessfully attempting to distribute. Solutions which have been proposed to solve this "deadlock" are slow, complicated, awkward, and frequently rely on the temporary use of distributed algorithms.¹⁷

Finally, centralized algorithms can place heavy and uneven demands on network line bandwidth; near the RCC there is a concentration of routing information going to and from the RCC. This heavy line utilization near the center means that centralized algorithms do not grow gracefully with the size of the network and, indeed, this may place an upper limit on the size of the network.

Isolated adaptive algorithms

One of the primary characteristics of an isolated algorithm which attempts to adapt to changing conditions is that it takes on the character of a heuristic process: it must "learn" and "forget" various facts about the network environment. While such an approach may have an intuitive appeal, it can be shown rather simply that heuristic routing procedures are unstable and are therefore not of interest for most practical network applications. The fundamental problem with isolated adaptive algorithms is that they must rely on indirect information about network conditions, since each node operates independently and without direct knowledge of or communication with the other nodes.

There are two basic approaches to be employed, separately or in tandem, to the process of learning and forgetting. We call these approaches positive feedback and negative feedback. One way to implement positive feedback was suggested by Baran as part of his hot-potato routing doctrine.¹⁸ Each node increments the handover number in a packet as it forwards the packet. Then the handover number is used in a "backwards learning" technique to estimate the transit time from the current node to the source of the packet. Clearly, this scheme has drawbacks because it lacks any *direct* way of adapting to changes. If no packets from a given source are routed through a node by the rest of the network, the node has no information about which route to choose in sending a message to that source. In general, as part of a positive feedback loop, the routing algorithm must periodically try routes other than the current best ones, since it has no direct way of knowing if better routes exist. Thus, there must always be some level of traffic traveling on any route that the nodes are to learn about, since it is only by feedback from traffic that they can learn.

The other half of an adaptive isolated algorithm is the negative feedback cycle. One technique to use here is to

penalize the choice of a given path when a packet is detected to have returned over the same path without being delivered to its destination. The relation of this technique to the exploratory nature of positive feedback is evident.

An adaptive isolated algorithm, therefore, has this fundamental weakness: in the attempt to adapt heuristically, it must oscillate, trying first one path and then another, even under stable network conditions. This oscillation violates one of the important goals of any routing algorithm, stability, and it leads to poor utilization of network resources and slow response to changing conditions. Incorrect routing of the packets during oscillation increases delay and reduces effective throughput correspondingly. There is no solution to the problem of oscillation in such algorithms. If the oscillation is damped to be slow, then the routing will not adapt quickly to improvements and will therefore declare nodes unreachable when they are not, with the result that suboptimal paths will be used for extended periods. If the oscillation is fast, then suboptimal paths will also be used much of the time, since the network will be chronically full of traffic going the wrong way.

Distributed adaptive algorithms

In our experience, distributed adaptive algorithms have none of the inherent limitations of the above algorithms; e.g., not the inherent unreliability and inefficiency of non-adaptive algorithms, nor the unreliability and size limitations of centralized algorithms, nor the inherent inefficiency and instability of isolated algorithms. For example, the distributed adaptive routing algorithm in the ARPA Network has operated for five years with little difficulty and good performance. However, distributed algorithms do have some practical difficulties which must be overcome in order to obtain good performance.

Consider the following example of a distributed adaptive algorithm. Each node estimates the "distance" it expects a packet to have to traverse to reach each possible destination over each of its output lines. Periodically, it selects the minimum distance estimate for each destination and passes these estimates to its immediate neighbors. Each node then constructs its own routing table by combining its neighbors' estimates with its own estimates of distance to each neighbor. For each destination, the table is then made to specify that selected output line for which the sum of the estimated distance to the neighbor plus the neighbor's distance estimate to the destination is smallest.

Such an algorithm can be made to measure distance in hops (i.e., lines which must be traversed), delay, or any of a number of other metrics including excess bandwidth and reliability (of course, for the latter two, one must maximize rather than minimize). The above algorithm is representative of a class of distributed adaptive algorithms which we consider briefly in the remainder of this section. For simplicity of discussion we will assume that distance is measured in hops.

The first point is that distributed algorithms are slow in adapting to some kinds of change; in particular, the algorithm reacts quickly to good news, and slowly to bad news. If the number of hops to a given node decreases, the nodes soon all agree on the new, lower, number. If the hop count increases, the nodes will not believe the reports of higher counts while they still have neighbors with the old, lower values. This is demonstrated in Reference 14. Another point is that there is no way for a node to know ahead of time what the next-best or fall-back path will be in the event of a failure, or indeed if one exists. In fact, there must be some finite time, the network response time, between when a change in the network occurs and when the routing algorithm adapts to the change. This time depends on the size and shape of the network.

We have come to conclude that the routing algorithm should continue to use the best route to a given destination, both for updating and forwarding, for some time period after it gets worse. That is, the algorithm should report to the adjacent nodes the current value of the previous best route and use it for routing packets for a given time interval. We call this *hold down*.¹⁴ One way to look at this is to distinguish between changes in the network topology and traffic that necessitate changing the choice of the best route, and those changes which merely affect the characteristics of the route, like hop count, delay, and throughput. In the case when the identity of the path remains the same, the mechanism of hold down provides an instantaneous adaptation to the changes in the characteristics of the path; certainly, this is optimal. When the identity of the path must change, the time to adapt is equal to the absolute minimum of one network response time, while the other nodes have a chance to react to the worsening of the best path and to decide on the next best path. This is optimal for any algorithm within the practical limits of propagation times.*

The routing algorithm is extremely important to network reliability, since if it malfunctions the network is useless. Further, a distributed routing algorithm has the property that all the nodes must be performing the routing computation correctly for the algorithm to be reliable. A local failure can have global consequences; e.g., one node announcing that it is the best path to all nodes. Routing messages between nodes must have checksums and must be discarded if a checksum error is detected. All routing *programs* must be checksummed before every execution to verify that the code about to be run is correct. The checksum of the program should include the preliminary checksum computation itself, the routing program, any constants referenced, and anything else which could affect its successful execution. Any time a checksum error is detected in a node, the node should immediately be stopped from participating in the routing computation until it is restored to correct operation again.

* This is a very simplified description of hold down. A more complete description states in detail when hold down should be invoked and for what duration. Such a description may be found in Reference 14 and more is being learned.¹⁹

Node-to-node transmission procedures

In this section we discuss some of the issues in designing node-to-node transmission procedures, that is, the packet processing algorithms. We touch on these points only briefly since many of them are simple or have been discussed previously. Note that many of these issues occur again in the discussion of source-to-destination transmission procedures.

Buffering and pipelining

As we noted in discussing memory requirements, the amount of node-to-node packet buffering needs to equal the product of the circuit rate times the expected acknowledgment delay in order to get full line utilization. It may also be efficient to provide a small amount of additional buffering to deal with statistical fluctuations in the arrival rates, i.e., to provide queueing. These requirements imply that the nodes must do bookkeeping about multiple packets, which raises the several issues discussed next.

Error control

We have discussed many of the aspects of node-to-node error control above: the need for a packet checksum, its size, the basis of the acknowledgment/retransmission system, the decision on whether the line is usable, and so on. These procedures are critical for network reliability, and they should therefore run smoothly in the face of any kind of node or circuit failure. Where possible, the procedures should be self-synchronizing; at least they should be free from deadlock and easy to resynchronize.¹⁰

Storage allocation and flow control

Storage allocation can be fairly simple for the packet processing algorithms. The sender must hold a copy of the packet until it receives an acknowledgment; the receiver can accept the packet if it is without error and there is an available buffer. The receiver should not use the last free buffer in memory, since that would cut off the flow of control information such as routing and acknowledgments. In accepting too many packets, there is also the chance of a storage-based deadlock in which two nodes are trying to send to each other and have no more room to accept packets. This is explained fully in Reference 20.

The above implies that the flow control procedures can also be fairly simple. The need to buffer a circuit can be expressed in a quantitative limit of a certain number of packets. Therefore, the node can apply a cut-off test per line as its flow control throttle. More stringent rules can be used, but may be unnecessary.

Priority

The issue of priority in packet processing is quite important for network performance. First of all, the concept

of two or more priority levels for packets is useful in decreasing queueing delay for important traffic. Beyond this, however, careful attention must be paid to other kinds of transmissions. Routing messages should go with the highest priority, followed by acknowledgments (which can also be piggybacked in packets). Packet retransmissions must be sent with the next highest priority, higher than that for first transmission of packets. If this priority is not observed, retransmissions can be locked out indefinitely. The question of preemptive priority (i.e., stopping a packet in mid-transmission to start a higher priority one) is one of a direct tradeoff of bandwidth against delay since circuit bandwidth is wasted by each preemption.

Packet size

There has been much thought given in the packet-switching community to the proper size for packets. Large packets have a lower probability of successful transmission over an error-prone telephone line (and this drives the packet size down), while overhead considerations (longer packets have a lower percentage overhead) drive packet size up. The delay-lowering effects of pipelining become more pronounced as packet size decreases, generally improving store-and-forward delay characteristics; further, decreasing packet size reduces the delay that priority packets see because they are waiting behind full length packets. However, as the packet size goes down, effective throughput also goes down due to overhead. Metcalfe has previously commented on some of these points.²¹

Kleinrock and Naylor²² recently suggested that the ARPA Network packet size was suboptimal and should perhaps be reduced from about 1000 bits to 250 bits. This was based on optimization of node buffer utilization for the observed traffic mix in the network. However, in Reference 23, we point out that the relative cost of node buffer storage vs. circuits is possibly such that one should not try to optimize node buffer storage. The true trade-off which governs packet size might well be efficient use of phone line bandwidth (driving packet size larger) vs. delay characteristics (driving packet size smaller). If buffer storage is limiting, perhaps one should just buy more. Further, it is probably true that if one is trying for high bandwidth utilization, buffer size must be large. That is, high bandwidth utilization probably implies the use of large packets, which implies full buffers; when idle, the buffer size does not matter.

As noted above, the choice of packet is influenced by many factors. Since some of the factors are inherently in conflict, an optimum is difficult to define, much less find. The current ARPA Network packet size of about 1000 bits is a good compromise. Other packet sizes (e.g., the 2000 bits used in several other networks) may also be acceptable compromises. However, note that a 2000-bit packet size generally means a factor of two increase in delay over a 1000-bit packet size, because even high priority short packets will be delayed behind normal long

packets which are in transmission at each node. The use of preemptive priority might make longer packet sizes efficient.

Davies and Barber²⁴ are often quoted as recommending a minimum length "packet" of about 2000 bits because they have concluded that most of the messages currently exchanged within banks and airlines fit nicely in one packet of this size. To clarify this point, we note that they use the term "packet" for the unit of information we call a "message" and thus are not actually addressing the issue of packet size. We discuss message size below.

SOURCE-TO-DESTINATION SOFTWARE DESIGN

In this section we discuss the end-to-end transmission procedures and the division of responsibility between the Hosts and nodes.

End-to-end transmission procedures

There is a considerable controversy at the present time over whether or not a store-and-forward subnetwork of nodes should concern itself with end-to-end transmission procedures. Many workers² feel that the subnetwork should be close to a pure packet carrier with little concern for maintaining message order, for high levels of correct message delivery, for message buffering in the subnetwork, etc. Other workers, including ourselves,²³ feel that the subnetwork should take responsibility for many of the end-to-end message processing procedures. Of course, there are some workers who hold to positions in between.³ However, many design issues remain constant whether these functions are performed at Host level or subnetwork level, and we discuss these constants in this section.

Buffering and pipelining

As noted earlier in this paper, any practical network must allow multiple messages simultaneously in transit between the source and the destination, to achieve high throughput. If, for example, one message of 2000 bits is allowed to be outstanding between the source and destination at a time, and the normal network transit for the message including destination-to-source acknowledgment is 100 milliseconds, then the throughput rate that can be sustained is 20,000 bits per second. If slow lines, slow responsiveness of the destination Host, great distance, etc., cause the normal network transit time to be half a second, then the throughput rate is reduced to only 4,000 bits per second. Likewise, we think that pipelining is essential for most networks to improve delay characteristics; data should travel in reasonably short packets.

To summarize, low delay requirements drive packet size smaller, network and Host lines faster, and network paths shorter (i.e., fewer node-to-node hops). High throughput requirements drive the number of packets in flight up, packet overhead down, and the number of alternative paths up.

Error control

We consider source-to-destination error control to comprise three tasks: detecting bit errors in the delivered messages, detecting missing messages or pieces of messages, and detecting duplicate messages or pieces of messages.

The former task is done in a straightforward manner through the use of checksums. A checksum is appended to the message at the source and the checksum is checked at the destination; when the checksum does not check at the destination, the incorrect message is discarded, requiring it to be retransmitted from the source. Several points about the manner in which checksumming should be done are worthy of note: (a) If possible, the checksum should check the correctness of the resequencing of the messages which possibly got out of order in their traversal of the network. (b) A powerful checksum is more efficient than alternative methods such as replication of a critical control field; it is better to extend the checksum by the number of bits that would have been used in the redundant field. (c) Unless encryption is desirable for some other reason it is simpler (and just as safe) to prevent delivery of a message to an incorrect Host through the use of a powerful checksum than it is to use an encryption mechanism. (d) Node-to-node checksums do not fulfill the same function as end-to-end checksums because they check only the lines, not the nodes.

An inherent characteristic of packet-switching networks is that some messages or portions of messages (i.e., packets) will fail to be delivered, and there will be some duplicate delivery of messages or portions of messages, as described in the section on network properties.*

Missing messages can be detected at the destination through the use of one state bit for each unit of information which can be simultaneously traversing the network. An interesting detail is that for the purposes of missing message detection, the state bits used must precisely cycle through all possible states. For example, stamping messages with a time stamp does nothing for the process of missing message detection because, unless a message is sent for every "tick" of the time stamp, there is no way to distinguish the case of a missing message from the case where no messages were sent for a time.

Duplicate messages can be detected with an identifying sequence number such that messages which arrive from a prior point in the sequence are recognized as duplicates. What should be noted carefully here is that duplicate messages can arrive at the destination up to some time, possibly quite long, after the original copy, and the sequence number must not complete a full cycle during this period. For example, if a network goal is to be able to transmit 200 minimum length messages per second from the source to the destination and each needs a unique sequence number, and if it is possible for messages to ar-

rive at the destination up to 15 seconds after initial transmission from the source, then the sequence number must be able to uniquely identify at least 3000 packets. It is usually no trouble to calculate the maximum number of messages that can be sent during some time interval. What is more difficult is to limit the maximum time after which duplicate messages will no longer arrive at the destination. One method is to put a timer in each message which is counted down as the message traverses the network; if the timer ever counts out, the message is discarded as too old, thus guaranteeing that no messages older than the initial setting of the timer will be delivered to the destination. Alternatively, one can calculate approximately the maximum arrival time through study of all the worst case paths through the network and all the worst case combinations of events which might cause messages to loop around in the network for excessive lengths of time; this seems to work reasonably well in practice.

In either case, there certainly must be mechanisms to resynchronize the sequence numbers between the source and the destination at node start-up time, to recover from a node failure, etc. A good practice is to resynchronize the sequence numbers occasionally even though they are not known to be out of step. A good frequency with which to do redundant resynchronization would be every time a message has not been sent for longer than the maximum delivery time. In fact, this is the maximum frequency with which the resynchronization can be done (without additional mechanisms); if duplicates are to be detected reliably, the sequence number at the destination *must* function without disruption for the maximum delivery time after the "last message" has been sent. If it is desirable or necessary to resynchronize the sequence numbers more often than the maximum time, an additional "use" number must be attached to the sequence number to uniquely identify which "instance" of this set of sequence numbers is in effect; and, of course, the packets must also carry the use number. This point is addressed in greater detail in References 25 and 26.

The next point to make about end-to-end error control is that any message going from source to destination can potentially be missing or duplicated; i.e., not only data messages but control messages. In fact, the very messages used in error control (e.g., sequence number resynchronization messages) can themselves be missing or duplicated, and a proper end-to-end protocol must handle these cases.

Finally, there must be some inquiry-response system from the source to the destination to complete the process of detecting lost messages. When the proper reply or acknowledgment has not been received for too long, the source may inquire whether the destination has received the message in question. Alternatively, the source may simply retransmit the message in question. In any case, this source inquiry and retransmission system must also function in the face of duplicated or lost inquiries and inquiry response control messages. As with the inter-node acknowledgment and retransmission system, the end-to-end acknowledgment and retransmission system must depend on positive acknowledgments from the destination to the

* Throughout the remainder of this subsection we use the word "message" to mean either messages or portions of messages (i.e., packets).

source and on explicit inquiries or retransmissions from the source. Negative acknowledgments from the destination to the source are never sufficient (because they might get lost) and are only useful for increased efficiency.

Storage allocation and flow control

One of the fundamental rules of communications systems is that the source cannot simply send data to the destination without some mechanism for guaranteeing storage for that data. In very primitive systems one can guarantee a rate of disposal of data, as to a line printer, and not exceed that rate at the data source. In more sophisticated systems there seem to be only two alternatives. Either one can explicitly reserve space at the destination for a known amount of data in advance of its transmission, or one can declare the transmitted copy of the data expendable, sending additional copies from the source until there is an acknowledgment from the destination. The first alternative is the high bandwidth solution: when there is no space, only tiny messages travel back and forth between the source and destination for the purpose of reserving destination storage. The second alternative is the low delay solution: the text of the message propagates as fast as possible. See Reference 10 for a more lengthy discussion.

In either case storage is tied up for an amount of time equal to at least the round trip time. This is a fundamental result—the minimum amount of buffering required by a communications system, either at the source or at the destination, equals the product of round trip time and the channel bandwidth. The only way to circumvent this result is to count on the destination behaving in some predictable fashion (an unrealistic assumption in the general case of autonomous communicating entities).

As we stated earlier, our experience and analysis convince us that if both low delay and high throughput are desired, then there must be mechanisms to handle each, since high throughput and low delay are conflicting goals. This is true, in particular, for the storage allocation mechanism. In several networks, e.g.,² mainly for the sake of simplicity, only the low delay solution has been proposed or implemented; that is, messages are transmitted from the source without reservation of space at the destination. Those people making the choice never to reserve space at the destination frequently assert that high bandwidth will still be possible through use of a mechanism whereby the source sends messages toward the destination, notes the arrival of acknowledgments from the destination, uses these acknowledgments to estimate the destination reception rate, and adjusts its transmissions to match that rate. We feel that such schemes may be quite difficult to parameterize for efficient control and therefore may result in reduced effective bandwidth and increased effective delay. If, in addition to possible discards at the destination, the network solves its internal problems by discarding packets, or if the destination Host too often solves its internal problems by discarding packets, perfor-

mance will suffer further. As reported in Reference 20, contention for destination storage, which must be resolved through the discard of packets in the absence of a storage allocation mechanism, happens practically continuously under even modest traffic loads, and in a way uncoordinated with the rates and strategies of the various sources. As a result, well-behaved Hosts may unavoidably be penalized for the actions of poorly-behaved Hosts.

In addition to space to hold all data, there must also be space to hold all control messages. In particular, there must be space to record what needs to be sent and what has been sent. If a message will result in a response, there must be space to hold the response; and once a response has been sent, the information about what kind of answer was sent must be kept for as long as retransmission of that response may be necessary.

Precedence and preemption

The first point to note about precedence and preemption is that the total transit time being specified for most packet-switching networks of which we are aware is on the order of less than a few seconds (often only a fraction of a second). Thus, the traditional specifications (for example, low priority traffic must be able to preempt all other traffic so that it can traverse the network in under two minutes) no longer make much sense. When all messages traverse the network in less than a few seconds, there is generally no need to specify that top priority traffic must preempt other traffic, nor to specify the relative precedences between the other types of traffic.

Though priority is not strictly necessary for speed, it may be useful for contention resolution. It appears to us that there are three precedence and preemption strategies that are reasonable to consider for a packet-switching network. Strategy 1 is to permanently assign the resources necessary to handle high priority traffic; this guarantees the delivery time for the high priority traffic but is expensive and should only be done for limited high priority traffic. Strategy 2 is to preempt resources as necessary for high priority traffic. This can have two effects. Preempting packet buffers results in data loss; preempting internal node tables (e.g., the tables associated with packet sequence numbering) results in state information loss. State information loss means that data errors are possible which may go unreported. Strategy 3 is not to preempt resources, and to rely on the standard mechanisms with a priority ordering. This is simple for the nodes, but it does not itself guarantee delivery within a certain time.

We think the correct strategy is probably a mixture of the strategies above. Possibly some resources, on a very limited basis, should be reserved for the tiny amount of flash traffic. This guarantees minimum delay without any queueing latency. For the rest of the traffic, the normal delivery times are probably acceptable. The presence of higher priority traffic can cause gradual throttling of lower priority traffic, without loss of state information. As the time to do this graceful throttling is normally only a frac-

tion of a second, the higher priority traffic has no real reason to demand instantaneous, information-losing preemption of the lower priority traffic.

Message size

The question is often asked: "If one increases packet size, and decreases message size until the two become the same, will not the difficult message reassembly problem be removed?" The answer is that, perhaps unfortunately, message size and packet size are almost unrelated to reassembly.

We have already noted the relationship between delay and packet size. 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. Larger packets become desirable only when lines become so long or fast that propagation delay is larger than transmission time.

Message size needs to be large because the overhead on messages is significant. It is inefficient for the nodes to have to address too many messages and it may be inefficient for Hosts to have too many message interrupts. The upper limit on message size is what can conveniently be reassembled, given node storage and networks delays.

When a channel has an appreciable delay, it is necessary to buffer several pieces of data in the channel at one time in order to obtain 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.

We do not feel that the choice between single- and multi-packet messages is as important as all the controversy on the subject would lead one to believe. There is agreement that buffering many data units in transit through the network simultaneously is a necessity. Having multi-packet messages is probably more efficient (as the extra level of hierarchy allows overhead functions to be applied at the correct, i.e., most efficient, level); having single-packet messages probably offers the opportunity for finer grained storage allocation and flow control mechanisms.

Division of responsibility between subnetwork and Host

In the previous section we discussed a number of issues of end-to-end procedure design which must be considered wherever the procedures are implemented, whether in the subnetwork or in the Hosts. In this section we discuss the proper division of responsibility between the subnetwork and the Hosts.

Extent of 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 Net-

work 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 such functions should reside at Host level rather than subnetwork level.^{2,27,28}

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 nodes should not; (b) Host/Host performance may be degraded by the nodes doing these jobs; (c) network interconnections may be impeded by the nodes doing message processing; (d) lockups can happen in subnetwork message processing; (e) the node would become simpler and have more buffering capacity if it did not have to do message processing.

The last point is true although the extent of simplification and the additional buffering is probably not significant, but we believe the other statements are subject to some question. We have previously^{23,25} given our detailed reasons for this belief. Here we simply 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, nodes must control subnetwork resources, and Hosts must control Host resources. For reliability, the basic subnetwork environment must be under the effective control of the node 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 node 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 node resources such as buffering, CPU bandwidth, and line bandwidth.

d. It is basically cheaper to do message processing in the nodes than in the Hosts and it has very few detrimental effects.

Peripheral processor connections

In a number of cases, an organization has desired to connect a large Host to a network by inserting an additional minicomputer between the main Host and the node. The general notion has been to locate the Host-Host transmission procedures in this additional machine, thus reliev-

ing the main Host from coping with these tasks. Stated reasons for this notion include:

- It is difficult to change the monitor in the main Host, and new monitor releases by the Host manufacturer pose continuing compatibility problems.
- Core or timing limitations exist in the main Host.
- It is desirable to use I/O arrangements that may already exist or be available between the main Host and the additional mini (and between the mini and the node) to avoid design or procurement of new I/O gear for the main Host.

While this approach may sound good in principle, and, in fact, may be the only possible approach in some instances, it often leads to problems.

First, the I/O arrangements between the main Host and any preexisting peripheral processor were not designed for network connection and usually present timing and bandwidth constraints that greatly degrade performance. More seriously, the logical protocols that may have preexisted will almost certainly preclude the main Host from acting as a general purpose Host on the network. For instance, while initial requirements may only indicate a need for simple file transfers to a single distant point, requirements tend to change in the face of new facilities, and the network cannot then be used to full advantage.²⁹

Second, the peripheral processor and its software are often provided by an outside group, and the Host organization may know even less about their innards than they know about the main Host. The node is centrally maintained, improved, modified, and controlled by the Network Manager, but the peripheral processor, while an equally foreign body, is not so fortunate. This issue alone is crucial; functions that do not belong in the main Hosts belong in centrally monitored network equipment. Note that it is exactly those Host groups who are unwilling to touch the main Host's monitor who will be unlikely to be able to make subtle improvements in the protocols, error message handling, and timing of the peripheral processor. From a broader economic view, common functions belong in the network and should be designed once; the peripheral processor approach is a succession of costly special cases and the total cost is greatly escalated.

The long term solution to the dilemma is to have the various manufacturers support hardware and software interfaces that connect to widely used networks. This is not likely to occur until commercial networks exist and are widely available. In the meantime, potential Host organizations that wish to use early networks (like the ARPA Network) should try to find ways to put the network connection directly into the main Host. An anthropomorphic illustration may be helpful: the network is, among other things, a set of standardized protocols or languages. A potential network Host is in the position of a person who needs to have dealings with people who speak a language he does not know. If he does not want to learn the language, he can indeed opt for using an interpreter, but

performance is poor, the process is very inconvenient, expensive, and unpleasant, and subtle meaning is always lost. The situation is quite similar when a Host tries to work through a peripheral processor. If a Host wishes to interact with a network, it is usually unrealistic to try to make the Host think that the network is a card reader or some other familiar peripheral. As usual, you get what you pay for.

Other message services

One commonly suggested design requirement is for storage in the communications subnetwork, usually for messages which are currently undeliverable because a Host or a line is down. This requirement should have no effect whatsoever on the design of the communications part of the network; it is an orthogonal requirement which should be implemented by providing special storage Hosts at strategic locations in the network. These can be at every node, at a few nodes, or at a single node, depending on the relative importance of reliability, efficient line utilization, and cost.

Another commonly suggested design requirement is for the communications subnetwork to provide a message broadcast capability; i.e., a Host gives a message to its node along with a list of Host addresses and the nodes somehow send copies to all the Hosts in the list. Again we believe that such a requirement should have no effect on the design of the communications part of the network and that messages to be broadcast should be sent to a special Host (perhaps one of the ones in the previous paragraph) for such broadcast.

CONCLUSION

There has now been considerable experience with the design of packet-switching networks and several groups (ours included) believe that they have come to understand many of the fundamental design issues. On the other hand, packet switching is still in its youth, and there are many new issues to be explored. Such new issues include, among others: (a) the techniques for transferring packet-switching technology from its initial limited R&D implementations to widespread production implementations; (b) the methods whereby the newly available packet-by-satellite technology can be utilized in packet-switching networks; (c) transmission of speech through packet-switching networks; (d) packet transmission by radio; (e) interconnection of packet-switching networks; and (f) effects of packet-switching networks on Host operating system design. Several other papers in these same proceedings cover in detail some of the new design issues just mentioned^{30,31,32,33} and we plan to address some of these new issues ourselves in the near future.

ACKNOWLEDGMENTS

Since 1969 our research on packet-switching network design has been encouraged and supported by the Information Processing Techniques office of the Advanced Research Projects Agency. Many of our colleagues at Bolt Beranek and Newman Inc. have participated in our research. Through the ARPA Network and through the International Network Working Group we have been fortunate to be able to receive frequently incisive critiques of our work and to have the opportunity to study the work of others. In particular, over the past year our interactions with Holger Opderbeck and his colleagues at UCLA have been enlightening. Finally, we acknowledge the substantial assistance of Robert Brooks and Barbara Erwin with the preparation of the manuscript, and we acknowledge the very useful comments of Drs. J. Burchfiel, W. Hawyrko, R. Metcalfe, and J. Postel, who reviewed the presentation and content of this paper.

REFERENCES

- Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *AFIPS Conference Proceedings*, Vol. 36, June 1970, pp. 551-567; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 300-316.
- Pouzin, L., "Presentation and Major Design Aspects of the Cyclades Computer Network," *Proceedings of the Third ACM Data Communications Symposium*, November 1973, pp. 80-88.
- Brant, G. J. and G. J. Chretien, "Methods to Control and Operate a Message-Switching Network," *Computer Communications Network and Teletraffic*, Polytechnic Press of the Polytechnical Institute of Brooklyn, Brooklyn, N.Y., 1972.
- Barber, D. L. A. (ed.), *A Specification for a European Informatics Network*, Co-Opération Européenne dans le Domaine de la Recherche Scientifique et Technique, January 4, 1974.
- Rosner, R. D., "A Digital Data Network Concept for the Defense Communications System," *Proceedings of the National Telecommunications Conference*, Atlanta, November 1973, pp. 22C1-6.
- Davies, D. W., K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals," *Proceedings of the ACM Symposium on Operating Systems Principles*, October 1967.
- Auerbach Publishers Inc., *Public Packet Switching Networks*, Data Processing Manual No. 3-08-04, 1974.
- Despres, R. F., "A Packet Switching Network with Graceful Saturated Operation," *Proceedings of the First International Conference on Computer Communication*, October 1972, pp. 345-351.
- Pouzin, L., *Basic Elements of a Network Data Link Control Procedure (NDLC)*, INWG 54, NIC 30375, January 1974, a limited number of copies available for the cost of reproduction and handling from INWG, c/o Prof. V. Cerf, Digital Systems Laboratory, Stanford, CA 94305.
- McQuillan, J. M., W. R. Crowther, B. P. Cosell, D. C. Walden, and F. E. Heart, "Improvements in the Design and Performance of the ARPA Network," *AFIPS Conference Proceedings*, Vol. 41, December 1972, pp. 741-754.
- Heart, F. E., S. M. Ornstein, W. R. Crowther, and W. B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," *AFIPS Conference Proceedings*, Vol. 42, June 1973, pp. 592-537; also in *Selected Papers: International Advanced Study Institute, Computer Communication Networks*, R. L. Grimsdale and F. F. Kuo (eds.), University of Sussex, Brighton, England, September 1973; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 329-337.
- Ornstein, S. M., W. R. Crowther, M. F. Krale, R. D. Bressler, A. Michel, and F. E. Heart, "Pluribus—A Reliable Multiprocessor," these proceedings.
- Frank, H., R. E. Kahn, and L. Kleinrock, "Computer Communications Network Design—Experience with Theory and Practice," *AFIPS Conference Proceedings*, Vol. 40, June 1972, pp. 255-270; also in *Networks*, Vol. 2, No. 2, 1972, pp. 135-166; also in *Advances in Computer Communication*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 254-269.
- McQuillan, J. M., *Adaptive Routing Algorithms for Distributed Computer Networks*, BBN Report No. 2831, May 1974, available from the National Technical Information Service, AD781467.
- Grange, J. L., *Cyclades Network*, personal communication.
- Floyd, R. W., "Algorithm 97, Shortest Path," *CACM* 5 (6), June 1962, p. 345.
- Gerla, M., "Deterministic and Adaptive Routing Policies in Packet-Switched Computer Networks," *Proceedings of the Third ACM Data Communications Symposium*, November 1973, pp. 23-28.
- Baran, P., *On Distributed Communications: I. Introduction to Distributed Communications Networks*, Rand Corp. Memo RM-3420-PR, August 1964, p. 37.
- Opderbeck, H. and W. Naylor, *ARPA Network Measurement Center*, personal communication.
- Kahn, R. E. and W. R. Crowther, "Flow Control in a Resource-Sharing Computer Network," *Proceedings of the Second ACM/IEEE Symposium on Problems in the Optimization of Data Communications Systems*, Palo Alto, California, October 1971, pp. 108-116; also in *IEEE Transactions on Communications*, Vol. COM-20, No. 3, Part II, June 1972, pp. 539-546.
- Metcalfe, R. M., *Packet Communication*, Massachusetts Institute of Technology Project MAC Report MAC TR-114, December 1973.
- Kleinrock, L. and W. Naylor, "On Measured Behavior of the ARPA Network," *AFIPS Conference Proceedings*, Vol. 43, May 1974, pp. 767-780.
- Crowther, W. R., F. E. Heart, A. A. McKenzie, J. M. McQuillan, and D. C. Walden, *Network Design Issues*, BBN Report No. 2918, November 1974, to be available from National Technical Information Service.
- Davies, D. W., and D. L. A. Barber, *Communication Networks for Computers*, London: John Wiley and Sons, 1973.
- McQuillan, J. M., "The Evolution of Message Processing Techniques in the ARPA Network," to appear in *International Computer State of the Art Report No. 24: Network Systems and Software*, Infotech, Maidenhead, England.
- Tomlinson, R. S., *Selecting Sequence Numbers*, INWG—Protocol Note #2, August 1974, available as with Reference 9.
- Cerf, V. and R. Kahn, "A Protocol for Packet Network Intercommunications," *IEEE Transactions on Communications*, Vol. COM-22, No. 5, May 1974, pp. 637-648.
- Cerf, V., *An Assessment of ARPANET Protocols*, RFC 635, NIC 30489, April 1974, available as with Reference 9.
- Metcalfe, R. M., "Strategies for Operating Systems in Computer Networks," *Proceedings of the ACM National Conference*, August 1972, pp. 278-281.
- Retz, D. L., "Operating System Design Considerations for the Packet Switching Environment," these proceedings.
- Forgie, J. W., "Speech Transmission in Packet-Switched Store-and-Forward Networks," these proceedings.
- Lam, S. S., and L. Kleinrock, "Dynamic Control Schemes for a Packet Switched Multi-Access Broadcast Channel," these proceedings.
- Kahn, R. E., "The Organization of Computer Resources into a Packet Radio Network," these proceedings.

HOST-HOST communication protocol in the ARPA network*

by C. STEPHEN CARR

University of Utah
Salt Lake City, Utah

and

STEPHEN D. CROCKER and VINTON G. CERF

University of California
Los Angeles, California

INTRODUCTION

The Advanced Research Projects Agency (ARPA) Computer Network (hereafter referred to as the "ARPA network") is one of the most ambitious computer networks attempted to date.¹ The types of machines and operating systems involved in the network vary widely. For example, the computers at the first four sites are an XDS 940 (Stanford Research Institute), an IBM 360/75 (University of California, Santa Barbara), an XDS SIGMA-7 (University of California, Los Angeles), and a DEC PDP-10 (University of Utah). The only commonality among the network membership is the use of highly interactive time-sharing systems; but, of course, these are all different in external appearance and implementation. Furthermore, no one node is in control of the network. This has insured generality and reliability but complicates the software.

Of the networks which have reached the operational phase and been reported in the literature, none have involved the variety of computers and operating systems found in the ARPA network. For example, the Carnegie-Mellon, Princeton, IBM network consists of 360/67's with identical software.² Load sharing among identical batch machines was commonplace at North American Rockwell Corporation in the early 1960's. Therefore, the implementers of the present network have been only slightly influenced by earlier network attempts.

However, early time-sharing studies at the University of California at Berkeley, MIT, Lincoln Laboratory, and System Development Corporation (all ARPA sponsored) have had considerable influence on the design of the network. In some sense, the ARPA network of time-shared computers is a natural extension of earlier time-sharing concepts.

The network is seen as a set of data entry and exit points into which individual computers insert messages destined for another (or the same) computer, and from which such messages emerge. The format of such messages and the operation of the network was specified by the network contractor (BB&N) and it became the responsibility of representatives of the various computer sites to impose such additional constraints and provide such protocol as necessary for users at one site to use resources at foreign sites. This paper details the decisions that have been made and the considerations behind these decisions.

Several people deserve acknowledgment in this effort. J. Rulifson and W. Duvall of SRI participated in the early design effort of the protocol and in the discussions of NIL. G. Deloche of Thomson-CSF participated in the design effort while he was at UCLA and provided considerable documentation. J. Curry of Utah and P. Rovner of Lincoln Laboratory reviewed the early design and NIL. W. Crowther of Bolt, Beranek and Newman contributed the idea of a virtual net. The BB&N staff provided substantial assistance and guidance while delivering the network.

We have found that, in the process of connecting machines and operating systems together, a great deal of rapport has been established between personnel at

*This research was sponsored by the Advanced Research Projects Agency, Department of Defense, under contracts AF30(602)-4277 and DAHC15-69-C-0285.

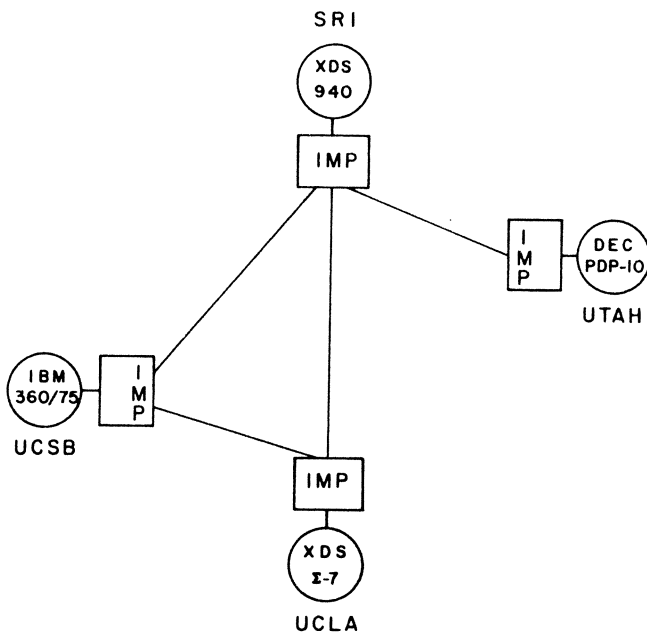


Figure 1—Initial network configuration

the various network node sites. The resulting mixture of ideas, discussions, disagreements, and resolutions has been highly refreshing and beneficial to all involved, and we regard the human interaction as a valuable by-product of the main effort.

THE NETWORK AS SEEN BY THE HOSTS

Before going on to discuss operating system communication protocol, some definitions are needed.

A *HOST* is a computer system which is part of the network.

An *IMP* (Interface Message Processor) is a Honeywell DDP-516 computer which interfaces with up to four HOSTs at a particular site, and allows HOSTs access into the network. The configuration of the initial four-HOST network is given in Figure 1. The IMPs form a store-and-forward communications network. A companion paper in these proceedings covers the IMPs in some detail.³

A *message* is a bit stream less than 8096 bits long which is given to an IMP by a HOST for transmission to another HOST. The first 32 bits of the message are the *leader*. The leader contains the following information:

- (a) HOST
- (b) Message type
- (c) Flags
- (d) Link number

When a message is transmitted from a HOST to its IMP, the HOST field of the leader names the receiving HOST. When the message arrives at the receiving HOST, the HOST field names the sending HOST.

Only two message types are of concern in this paper. Regular messages are generated by a HOST and sent to its IMP for transmission to a foreign HOST. The other message type of interest is a RFNM (Request-for-Next-Message). RFNMs are explained in conjunction with links.

The flag field of the leader controls special cases not of concern here.

The link number identifies over which of 256 logical paths (links) between the sending HOST and the receiving HOST the message will be sent. Each link is unidirectional and is controlled by the network so that no more than one message at a time may be sent over it. This control is implemented using RFNM messages. After a sending HOST has sent a message to a receiving HOST over a particular link, the sending HOST is prohibited from sending another message over that same link until the sending HOST receives a RFNM. The RFNM is generated by the IMP connected to the receiving HOST, and the RFNM is sent back to the sending HOST after the message has entered the receiving HOST. It is important to remember that there are 256 links in each direction and that no relationship among these is imposed by the network.

The purpose of the link and RFNM mechanism is to prohibit individual users from overloading an IMP or a HOST. Implicit in this purpose is the assumption that a user does not use multiple links to achieve a wide band, and to a large extent the HOST-HOST protocol cooperates with this assumption. An even more basic assumption, of course, is that the network's load comes from some users transmitting sequences of messages rather than many users transmitting single messages coincidentally.

In order to delimit the length of the message, and to make it easier for HOSTs of differing word lengths to communicate, the following formatting procedure is used. When a HOST prepares a message for output, it creates a 32-bit leader. Following the leader is a binary string, called *marking*, consisting of an arbitrary number of zeroes, followed by a one. Marking makes it possible for the sending HOST to synchronize the beginning of the text of a message with its word boundaries. When the last bit of a message has entered an IMP, the hardware interface between the IMP and HOST appends a one followed by enough zeroes to make the message length a multiple of 16 bits. These appended bits are called *padding*. Except for the marking and padding, no limitations are placed on the text of a message. Figure 2 shows a typical message sent by a 24-bit machine.

DESIGN CONCEPTS

The computers participating in the network are alike in two important respects: each supports research inde-

pendent of the network, and each is under the discipline of a time-sharing system. These facts contributed to the following design philosophy.

First, because the computers in the network have independent purposes, it is necessary to preserve decentralized administrative control of the various computers. Since all of the time-sharing supervisors possess elaborate and definite accounting and resource allocation mechanisms, we arranged matters so that these mechanisms would control the load due to the network in the same way they control locally generated load.

Second, because the computers are all operated under time-sharing disciplines, it seemed desirable to facilitate basic interactive mechanisms.

Third, because this network is used by experienced programmers it was imperative to provide the widest latitude in using the network. Restrictions concerning character sets, programming languages, etc., would not be tolerated and we avoided such restrictions.

Fourth, again because the network is used by experienced programmers, it was felt necessary to leave the design open-ended. We expect that conventions will arise from time to time as experience is gained, but we felt constrained not to impose them arbitrarily.

Fifth, in order to make network participation comfortable, or in some cases, feasible, the software interface to the network should require minimal surgery on the HOST operating system.

Finally, we accepted the assumption stated above that network use consists of prolonged conversations instead of one-shot requests.

Those considerations led to the notions of connections, a Network Control Program, a control link, control commands, sockets, and virtual nets.

A *connection* is an extension of a link. A connection connects two processes so that output from one process

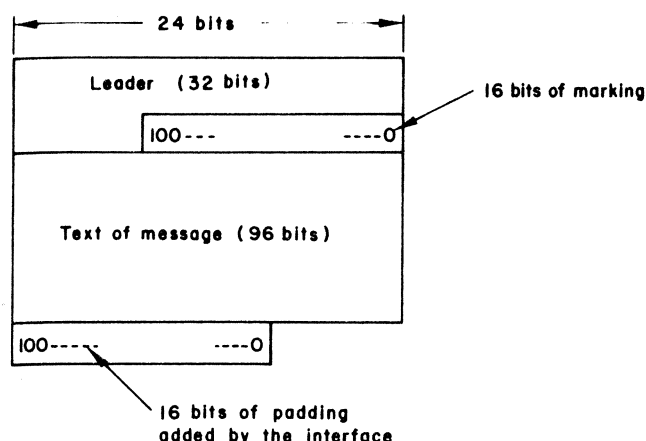


Figure 2—A typical message from a 24-bit machine

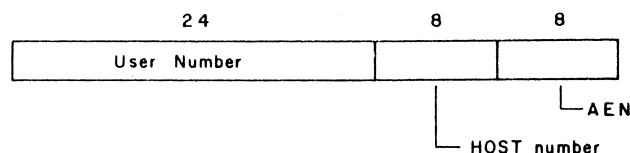


Figure 3—A typical socket

is input to the other. Connections are simplex, so two connections are needed if two processes are to converse in both directions.

Processes within a HOST communicate with the network through a *Network Control Program* (NCP). In most HOSTs, the NCP will be part of the executive, so that processes will use system calls to communicate with it. The primary function of the NCP is to establish connections, break connections, switch connections, and control flow.

In order to accomplish its tasks, a NCP in one HOST must communicate with a NCP in another HOST. To this end, a particular link between each pair of HOSTs has been designated as the *control link*. Messages received over the control link are always interpreted by the NCP as a sequence of one or more *control commands*. As an example, one of the kinds of control commands is used to assign a link and initiate a connection, while another kind carries notification that a connection has been terminated. A partial sketch of the syntax and semantics of control commands is given in the next section.

A major issue is how to refer to processes in a foreign HOST. Each HOST has some internal naming scheme, but these various schemes often are incompatible. Since it is not practical to impose a common internal process naming scheme, an intermediate name space was created with a separate portion of the name space given to each HOST. It is left to each HOST to map internal process identifiers into its name space.

The elements of the name space are called *sockets*. A socket forms one end of a connection, and a connection is fully specified by a pair of sockets. A socket is specified by the concatenation of three numbers:

- (a) a user number (24 bits)
- (b) a HOST number (8 bits)
- (c) AEN (8 bits)

A typical socket is illustrated in Figure 3.

Each HOST is assigned all sockets in the name space which have field (b) equal to the HOST's own identification.

A socket is either a *receive socket* or a *send socket*, and is so marked by the low-order bit of the AEN (0 = receive, 1 = send). The other seven bits of the

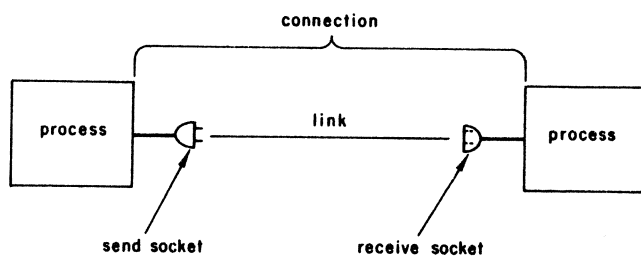


Figure 4—The relationship between sockets and processes

AEN simply provide a sizable population of sockets for each user number at each HOST. (AEN stands for "another eight-bit number".)

Each user is assigned a 24-bit user number which uniquely identifies him throughout the network. Generally this will be the 8-bit HOST number of his home HOST, followed by 16 bits which uniquely identify him at that HOST. Provision can also be made for a user to have a user number not keyed to a particular HOST, an arrangement desirable for mobile users who might have no home HOST or more than one home HOST. This 24-bit user number is then used in the following manner. When a user signs onto a HOST, his user number is looked up. Thereafter, each process the user creates is tagged with his user number. When the user signs onto a foreign HOST via the network, his same user number is used to tag processes he creates in that HOST. The foreign HOST obtains the user number either by consulting a table at login time, as the home HOST does, or by noticing the identification of the caller. The effect of propagating the user's number is that each user creates his own *virtual net* consisting of processes he has created. This virtual net may span an arbitrary number of HOSTs. It will thus be easy for a user to connect his processes in arbitrary ways, while still permitting him to connect his processes with those in other virtual nets.

The relationship between sockets and processes is now describable (see Figure 4). For each user number at each HOST, there are 128 send sockets and 128 receive sockets. A process may request from the local NCP the use of any one of the sockets with the same user number; the request is granted if the socket is not otherwise in use. The key observation here is that a socket requested by a process cannot already be in use unless it is by some other process within the same virtual net, and such a process is controlled by the same user.

An unusual aspect of the HOST-HOST protocol is that a process may switch its end of a connection from one socket to another. The new socket may be in any virtual net and at any HOST, and the process may

initiate a switch either at the time the connection is being established, or later. The most general forms of switching entail quite complex implementation, and are not germane to the rest of this paper, so only a limited form will be explained. This limited form of switching provides only that a process may substitute one socket for another while establishing a connection. The new socket must have the same user number and HOST number, and the connection is still established to the same process. This form of switching is thus only a way of relabelling a socket, for no change in the routing of messages takes place. In the next section we document the system calls and control commands; in the section after next, we consider how login might be implemented.

SYSTEM CALLS AND CONTROL COMMANDS

Here we sketch the mechanics of establishing, switching and breaking a connection. As noted above, the NCP interacts with user processes via system calls and with other NCPs via control commands. We therefore begin with a partial description of system calls and control commands.

System calls will vary from one operating system to another, so the following description is only suggestive. We assume here that a process has several input-output paths which we will call *ports*. Each port may be connected to a sequential I/O device, and while connected, transmits information in only one direction. We further assume that the process is blocked (dismissed, slept) while transmission proceeds. The following is the list of system calls:

Init	$\langle \text{port} \rangle$, $\langle \text{AEN 1} \rangle$, $\langle \text{AEN 2} \rangle$, $\langle \text{foreign socket} \rangle$
where $\langle \text{port} \rangle$	is part of the process issuing the Init
and	$\left. \begin{array}{l} \langle \text{AEN 1} \rangle \\ \langle \text{AEN 2} \rangle \end{array} \right\}$ are 8-bit AEN's (see Figure 3)

$\langle \text{foreign socket} \rangle$ is the 40-bit socket name of the distant end of the connection.

The first AEN is used to initiate the connection; the second is used while the connection exists.

The low-order bits of $\langle \text{AEN 1} \rangle$ and $\langle \text{AEN 2} \rangle$ must agree, and these must be the complement of the low-order bit of $\langle \text{foreign socket} \rangle$.

The NCP concatenates $\langle \text{AEN 1} \rangle$ and $\langle \text{AEN 2} \rangle$ each with the user number of the process and the HOST number to form 40-bit sockets.

It then sends a Request for Connection (RFC) control command to the distant NCP. When the distant NCP responds positively, the connection is established and

the process is unblocked. If the distant NCP responds negatively, the local NCP unblocks the requesting process, but informs it that the system call has failed.

Listen $\langle \text{port} \rangle$, $\langle \text{AEN } 1 \rangle$

where $\langle \text{port} \rangle$ and $\langle \text{AEN } 1 \rangle$ are as above.

The NCP retains the ports and $\langle \text{AEN } 1 \rangle$ and blocks the process. When an RFC control command arrives naming the local socket, the process is unblocked and notified that a foreign process is calling.

Accept $\langle \text{AEN } 2 \rangle$

After a listen has been satisfied, the process may either refuse the call or accept it and switch it to another socket. To accept the call, the process issues the Accept system call. The NCP then sends back an RFC control command.

Close $\langle \text{port} \rangle$

After establishing a connection, a process issues a Close to break the connection. The Close is also issued after a Listen to refuse a call.

Transmit $\langle \text{port} \rangle$, $\langle \text{addr} \rangle$

If $\langle \text{port} \rangle$ is attached to a send socket, $\langle \text{addr} \rangle$ points to a message to be sent. This message is preceded by its length in bits.

If $\langle \text{port} \rangle$ is attached to a receive socket, a message is stored at $\langle \text{addr} \rangle$. The length of the message is stored first.

Control commands

A vocabulary of control commands has been defined for communication between Network Control Programs. Each control command consists of an 8-bit operation code to indicate its function, followed by some parameters. The number and format of parameters is fixed for each operation code. A sequence of control commands destined for a particular HOST can be packed into a single control message.

RFC $\langle \text{my socket } 1 \rangle$, $\langle \text{my socket } 2 \rangle$,
 $\langle \text{your socket} \rangle$, $\langle \text{link} \rangle$

This command is sent because a process has executed either an Init system call or an Accept system call. A link is assigned by the prospective receiver, so it is omitted if $\langle \text{my socket } 1 \rangle$ is a send socket.

There is distinct advantage in using the same commands both to initiate a connection (Init) and to accept a call (Accept). If the responding command were different from the initiating command, then two processes could call each other and become blocked waiting for each other to respond. With this scheme no deadlock

occurs and it provides a more compact way to connect a set of processes.

CLS $\langle \text{my socket} \rangle$, $\langle \text{your socket} \rangle$

The specified connection is terminated

CEASE $\langle \text{link} \rangle$

When the receiving process does not consume its input as fast as it arrives, the buffer space in the receiving HOST is used to queue the waiting messages. Since only limited space is generally available, the receiving HOST may need to inhibit the sending HOST from sending any more messages over the offending connection. When the sending HOST receives this command, it may block the process generating the messages.

RESUME $\langle \text{link} \rangle$

This command is also sent from the receiving HOST to the sending HOST and negates a previous CEASE.

LOGGING IN

We assume that within each HOST there is always a process in execution which listens to login requests. We call this process the *logger*, and it is part of a special virtual net whose user number is zero. The logger is programmed to listen to calls on socket number 0. Upon receiving a call, the logger switches it to a higher (even) numbered socket, and returns a call to the socket numbered one less than the send socket originally calling. In this fashion, the logger can initiate 127 conversations.

To illustrate, assume a user whose identification is X'010005' (user number 5 at UCLA) signs into UCLA, starts up one of his programs, and this program wants to start a process at SRI. No process at SRI except the logger is currently willing to listen to our user, so he executes

Init, $\langle \text{port} \rangle = 1$, $\langle \text{AEN } 1 \rangle = 7$,
 $\langle \text{AEN } 2 \rangle = 7$,
 $\langle \text{foreign socket} \rangle = 0$.

His process is blocked, and the NCP at UCLA sends

RFC $\langle \text{my socket } 1 \rangle = \text{X}'0100050107'$,
 $\langle \text{my socket } 2 \rangle = \text{X}'0100050107'$,
 $\langle \text{your socket} \rangle = \text{X}'0000000200'$

The logger at SRI is notified when this message is received, because it has previously executed

Listen $\langle \text{port} \rangle = 9$, $\langle \text{AEN } 1 \rangle = 0$.

- (i) .LOGIN^{cr}
- (ii) .R TELNET^{cr}
- (iii) ESCAPE CHARACTER IS ^{cr}
- (iv) CONNECT TO SRI^{cr}
- (v) @ENTER CARR.^{cr}
- (vi) @CAL.^{cr}
- (vii) CAL AT YOUR SERVICE^{cr}
- (viii) >READ FILE FROM NETWRK.^{cr}
- (ix) #NETWRK:←DSK:MYFILE.CAL^{cr}

Figure 5—A typical TELNET dialog
Underlined characters are those typed by the user

The logger then executes

Accept ⟨AEN 2⟩ = 88.

In response to the Accept, the SRI NCP sends

RFC ⟨my socket 1⟩ = X'0000000200'
 ⟨my socket 2⟩ = X'0000000258'
 ⟨your socket⟩ = X'0100050107'
 ⟨link⟩ = 37

where the link has been chosen from the set of available links. The SRI logger then executes

Init ⟨port⟩ = 10
 ⟨AEN 1⟩ = 89, ⟨AEN 2⟩ = 89,
 ⟨foreign socket⟩ = X'0100050106'

which causes the NCP to send

RFC ⟨my socket 1⟩ = X'0000000259'
 ⟨my socket 2⟩ = X'0000000259'
 ⟨your socket⟩ = X'0100050106'

The process at UCLA is unblocked and notified of the successful Init. Because the SRI logger always initiates a connection to the AEN one less than it has just been connected to, the UCLA process then executes

Listen ⟨port⟩ = 11
 ⟨AEN 1⟩ = 6

and when unblocked,

Accept ⟨AEN 2⟩ = 6.

When these transactions are complete, the UCLA process is doubly connected to the logger at SRI. The logger will then interrogate the UCLA process, and if satisfied, create a new process at SRI. This new process will be tagged with the user number X'010005', and both connections will be switched to the new process. In this case, switching the connections to the new process corresponds to "passing the console down" in many time-sharing systems.

USER LEVEL SOFTWARE

At the user level, subroutines which manage data buffers and format input destined for other HOSTs are provided. It is not mandatory that the user use such subroutines, since the user has access to the network system calls in his monitor.

In addition to user programming access, it is desirable to have a subsystem program at each HOST which makes the network immediately accessible from a teletype-like device without special programming. Subsystems are commonly used system components such as text editors, compilers and interpreters. An example of a network-related subsystem is *TELNET*, which will allow users at the University of Utah to connect to Stanford Research Institute and appear as regular terminal users. It is expected that more sophisticated subsystems will be developed in time, but this basic one will render the early network immediately useful.

A user at the University of Utah (UTAH) is sitting at a teletype dialed into the University's PDP-10/50 time-sharing system. He wishes to operate the Conversational Algebraic Language (CAL) subsystem on the XDS-940 at Stanford Research Institute (SRI) in Menlo Park, California. A typical TELNET dialog is illustrated in Figure 5. The meaning of each line of dialog is discussed here.

- (i) The user signs in at UTAH.
- (ii) The PDP-10 run command starts up the TELNET subsystem at the user's HOST.
- (iii) The user identifies a break character which causes any message following the break to be

interpreted locally rather than being sent on to the foreign HOST.

- (iv) The TELNET subsystem will make the appropriate system calls to establish a pair of connections to the SRI logger. The connections will be established only if SRI accepts another foreign user.

The UTAH user is now in the pre-logged-in state at SRI. This is analogous to the standard teletype user's state after dialing into a computer and making a connection but before typing anything.

- (v) The user signs in to SRI with a standard login command.

Characters typed on the user's teletype are transmitted unaltered through the PDP-10 (user HOST) and on to the 940 (serving HOST). The PDP-10 TELNET subsystem will have automatically switched to full-duplex, character-by-character transmission, since this is required by SRI's 940. Full duplex operation is allowed for by the PDP-10, though not used by most Digital Equipment Corporation subsystems.

(vi) and (vii) The 940 subsystem, CAL, is started. At this point, the user wishes to load a CAL file into the 940 CAL subsystem from the file system on his local PDP-10.

- (viii) CAL is instructed to establish a connection to UTAH in order to receive the file. "NETWRK" is a predefined 940 name similar in nature to "PAPER TAPE" or "TELETYPE".

- (ix) Finally, the user types the break character (#) followed by a command to his PDP-10 TELNET program, which sends the desired file to SRI from Utah on the connection just established for this purpose. The user's next statement is in CAL again.

The TELNET subsystem coding should be minimal for it is essentially a shell program built over the network system calls. It effectively establishes a shunt in the user HOST between the remote user and a distant serving HOST.

Given the basic system primitives, the TELNET subsystem at the user HOST and a manual for the serving HOST, the network can be profitably employed by remote users today.

HIGHER LEVEL PROTOCOL

The network poses special problems where a high degree of interaction is required between the user and a particular subsystem on a foreign HOST. These problems arise due to heterogeneous consoles, local operating system overhead, and network transmission delays. Unless we use special strategies it may be

difficult or even impossible for a distant user to make use of the more sophisticated subsystems offered. While these difficulties are especially severe in the area of graphics, problems may arise even for teletype interaction. For example, suppose that a foreign subsystem is designed for teletype consoles connected by telephone, and then this subsystem becomes available to network users. This subsystem might have the following characteristics.

1. Except for echoing and correction of mistyping, no action is taken until a carriage return is typed.
2. All characters except "↑", "←" and carriage return are echoed as the character typed.
3. ← causes deletion of the immediately preceding accepted character, and is echoed as that character.
4. ↑ causes all previously typed characters to be ignored. A carriage return and line feed are echoed.
5. A carriage return is echoed as a carriage return followed by a line feed.

If each character typed is sent in its own message, then the characters

H E L L O ← ← P c.r.

cause nine messages in each direction. Furthermore, each character is handled by a user level program in the local HOST before being sent to the foreign HOST.

Now it is clear that if this particular example were important, we would quickly implement rules 1 to 5 in a local HOST program and send only complete lines to the foreign HOST. If the foreign HOST program could not be modified so as to not generate echoes, then the local program could not only echo properly, it could also throw away the later echoes from the foreign HOST. However, the problem is not any particular interaction scheme; the problem is that we expect many of these kinds of schemes to occur. We have not found any general solutions to these problems, but some observations and conjectures may lead the way.

With respect to heterogeneous consoles, we note that although consoles are rarely compatible, many are equivalent. It is probably reasonable to treat a model 37 teletype as the equivalent of an IBM 2741. Similarly, most storage scopes will form an equivalence class, and most refresh display scopes will form another. Furthermore, a hierarchy might emerge with members of one class usable in place of those in another, but not vice versa. We can imagine that any scope might be an adequate substitute for a teletype, but hardly the reverse. This observation leads us to wonder if a network-wide language for consoles might be possible. Such a language would provide for distinct treatment of different classes of consoles, with semantics ap-

propriate to each class. Each site could then write interface programs for its consoles to make them look like network standard devices.

Another observation is that a user evaluates an interactive system by comparing the speed of the system's responses with his own expectations. Sometimes a user feels that he has made only a minor request, so the response should be immediate; at other times he feels he has made a substantial request, and is therefore willing to wait for the response. Some interactive subsystems are especially pleasant to use because a great deal of work has gone into tailoring the responses to the user's expectations. In the network, however, a local user level process intervenes between a local console and a foreign subsystem, and we may expect the response time for minor requests to degrade. Now it may happen that all of this tailoring of the interaction is fairly independent of the portion of the subsystem which does the heavy computing or I/O. In such a case, it may be possible to separate a subsystem into two sections. One section would be the "substantive" portion; the other would be a "front end" which formats output to the user, accepts his inputs, and controls computationally simple responses such as echoes. In the example above, the program to accumulate a line and generate echoes would be the front end of some subsystem. We now take notice of the fact that the local HOSTs have substantial computational power, but our current designs make use of the local HOST only as a data concentrator. This is somewhat ironic, for the local HOST is not only poorly utilized as a data concentrator, it also degrades performance because of the delays it introduces.

These arguments have led us to consider the possibility of a Network Interface Language (NIL) which would be a network-wide language for writing the front end of interactive subsystems. This language would have the feature that subprograms communicate through network-like connections. The strategy is then to transport the source code for the front end of a subsystem to the local HOST, where it would be compiled and executed.

During preliminary discussions we have agreed that NIL should have at least the following semantic properties not generally found in languages.

1. Concurrency. Because messages arrive asynchronously on different connections, and because user input is not synchronized with subsystem output, NIL must include semantics to accurately model the possible concurrencies.
2. Program Concatenation. It is very useful to be able to insert a program in between two other programs. To achieve this, the interconnection of programs

would be specified at run time and would not be implicit in the source code.

3. Device substitutability. It is usual to define languages so that one device may be substituted for another. The requirement here is that any device can be modeled by a NIL program. For example, if a network standard display controller manipulates tree-structures according to messages sent to it then these structures must be easily implementable in NIL.

NIL has not been fully specified, and reservations have been expressed about its usefulness. These reservations hinge upon our conjecture that it is possible to divide an interactive subsystem into a transportable front end which satisfies a user's expectations at low cost and a more substantial stay-at-home section. If our conjecture is false, then NIL will not be useful; otherwise it seems worth pursuing. Testing of this conjecture and further development of NIL will take priority after low level HOST-HOST protocol has stabilized.

HOST/IMP INTERFACING

The hardware and software interfaces between HOST and IMP is an area of particular concern to the HOST organizations. Considering the diversity of HOST computers to which a standard IMP must connect, the hardware interface was made bit serial and full-duplex. Each HOST organization implements its half of this very simple interface.

The software interface is equally simple and consists of messages passed back and forth between the IMP and HOST programs. Special error and signal messages are defined as well as messages containing normal data. Messages waiting in queues in either machine are sent at the pleasure of the machine in which they reside with no concern for the needs of the other computer.

The effect of the present software interface is the needless rebuffering of all messages in the HOST in addition to the buffering in the IMP. The messages have no particular order other than arrival times at the IMP. The Network Control Program at one HOST (e.g., Utah) needs waiting RFRM's before all other messages. At another site (e.g., SRI), the NCP could benefit by receiving messages for the user who is next to be run.

What is needed is coding representing the specific needs of the HOST on both sides of the interface to make intelligent decisions about what to transmit next over the channel. With the present software interface, the channel in one direction once committed to a particular message is then locked up for up to 80 milli-

seconds. This approaches one teletype character time and needlessly limits full-duplex, character by character, interaction over the net. At the very least, the IMP/HOST protocol should be expanded to permit each side to assist the other in scheduling messages over the channels.

CONCLUSIONS

At this time (February 1970) the initial network for four sites is just beginning to be utilized. The communications system of four IMPs and wide band telephone lines have been operational for two months. Programmers at UCLA have signed on as users of the SRI 940. More significantly, one of the authors (S. Carr) living in Palo Alto uses the Salt Lake PDP-10 on a daily basis by first connecting to SRI. We thus have first hand experience that remote interaction is possible and is highly effective.

Work on the ARPA network has generated new

areas of interest. NIL is one example, and interprocess communication is another. Interprocess communication over the network is a subcase of general interprocess communication in a multiprogrammed environment. The mechanism of connections seems to be new, and we believe this mechanism is useful even when the processes are within the same computer.

REFERENCES

- 1 L ROBERTS
The ARPA network
Invitational Workshop on Networks of Computers
Proceedings National Security Agency p 115 ff 1968
- 2 R M RUTLEDGE et al
An interactive network of time-sharing computers
Proceedings of the 24th National Conference Association for
Computing Machinery p 431 ff 1969
- 3 F E HEART R E KAHN S M ORNSTEIN
W R CROWTHER D C WALDEN
The interface message processor for the ARPA computer network
These Proceedings

Although this document was written for people familiar with the File Transfer Protocol, lay users have had little trouble using FTP. A couple words on the protocol will improve matters further. File transfer is done by two programs. The first is this program (I will call it FTP) which is run by the user and acts as an interface between the user and network. The second is the server end (I will call it the server) which runs at the foreign site and acts as the interface between the foreign file system and FTP. These two programs communicate through a software entity called a 'socket'. FTP sends ASCII messages to the server, which replies with ASCII messages preceded with a decimal message code. The decimal code is meant for FTP to look at, and the rest is meant for the user. FTP will print both on the user's TTY. Normally only the more important messages will be printed, as FTP will filter out most of the uninteresting ones.

FTP still does not handle all network errors correctly, so if you have trouble making connections, use the monitor command 'IMP CLOSE/SELF' and try again.

* * * COMMANDS * * *

The FTP commands are broken down into five groups: network control, server control, file access, file transfer, and miscellaneous. Many of the commands will copy data directly from the teletype to the server, so one must be careful to end commands with a carriage return. Each command is a string, which may be abbreviated. However, if new commands are added later that prevent certain abbreviations from working, then I will resist all efforts by anyone to implement those abbreviations as real commands. I.e., if you dislike having to learn new abbreviations, then don't use abbreviations at all.

* * NETWORK CONTROL COMMANDS * *

The network control commands control the network connection to various hosts. While there is no connection, commands are prompted by a *. While the connection is open, the prompt character is !. While in ! mode, the interface never goes into TTY IOWait, but alternately checks for TTY or TELNET activity. If a message comes over the network and you have not started typing a command, that message will be printed and you will be prompted with another !.

FTP USER PROGRAM DOCUMENTATION 2
Version 3E

* HOST - setup a TELNET connection. *

The HOST command takes a host specification - either a host name (it uses the same algorithm as IMPCOM, so abbreviations, nicknames, etc. are permissible) or it's host number (in decimal) - and opens a TELNET connection to it. On a CMU-CMU connection, a USER command (see below) for your PPN may be sent. If you plan to use FTP solely to send mail to someone, a carriage return will be sufficient. Successful completion puts FTP into I mode. While in I mode, any messages the server sends will be printed on the teletype until you start typing a new command. At this point, FTP will wait for a line, and keep waiting - even if you rubout what has been typed - until you type a carriage return. If the foreign site closes the TELNET connections, then FTP will automatically return to * mode. It is a condition that will not normally occur. Should you control-C FTP to do something else, and wish to reuse the old connection (for more transfers or to close the connection properly), the HOST command with no arguments may be used, as the host name in this case is simply redundant data.

Example:

```
*HOST SAIL
300 SU-AI FTP SERVER
I
```

* LOCAL - set local socket number. *

The LOCAL command (with a decimal argument) sets the local socket number to be used in future connections. This command is necessary if you want more than 1 job running FTP under the same man number.

Example:

```
*LOCAL 20 ;The initial value is 20.
```

* QUIT - break TELNET connection. *

The QUIT command does different functions depending upon the current prompt character. In I mode, it will fake a BYE command (see below) if you haven't done one, close the telnet connections and return to * mode. In * mode, it will return to monitor mode (. mode if you wish).

Examples:

```
IQUIT
231 SEE YOU AGAIN LATER ;Only if you didn't do a BYE
*QUIT
```

* * SERVER CONTROL COMMANDS * *

All server control commands must be executed in I mode, since they all need to be able to output to the server.

FTP USER PROGRAM DOCUMENTATION 3
Version 3E

* USER - tell the server who you are. *

The USER command should be the first server command used after making the TELNET connection. It takes a string argument which it passes to the server. This command identifies yourself to the server for billing and access checking purposes. Note: On a CMU-CMU connection, the HOST command optionally simulate a USER command for your PPN. If the server requires a password, then the USER command will simulate a PASSWORD command (see below).

Examples:

```
*HOST CMU-10A           ;CMU-CMU users can free user
Want to Login? (Y or CR): Y ; cmd with their PPN
IUSER NET,GUE           ;for places like Stanford
IUSER GUEST              ;for XEROX
```

* PASSWORD - for security. *

The PASSWORD command may not be necessary on all systems. It takes a string argument which it passes to the server. Note: Under most circumstances, the USER command will catch the password requirement, and will prompt for one.

Example:

```
IPASSWORD ARPA
```

* ACCOUNT - for TENEX only. *

TENEX systems require an account number before they will let you write files on their systems. The ACCOUNT command will make TENEX happy, it takes a decimal argument.

Example:

```
IHOST TENEX           ;connect to TENEX
300 BBN-TENEX FTP Server 1.31.02.....
Iuser network-user     ;an invalid number most places
Password:              ;this will not echo.
331 Password OK, Account please.
IACCOUNT 123           ;give it the account number
I                       ;it wants
```

* MAIL - send mail from TTY input. *

There are two ways to send mail via FTP - the MAIL and MLFL commands (MLFL will be described later). MAIL takes a user specification as an argument, which it passes to the server. If the server decides that user exists at its site, it will tell FTP to go ahead, and FTP will tell you. Thereafter, everything typed on your terminal will be sent to that user. Mail may be terminated by an altmode character, control-Z, or by a line containing only a period. Note that FTP doesn't tell you all, so keep these in mind.

Example:

```
MAIL WERME
```


FTP USER PROGRAM DOCUMENTATION
Version 3E

4

Enter mail, end with altmode.
EXAMPLE OF FTP MAIL.

\$
I

* SEARCH - change structure search list. *

The SEARCH command allows you to modify the server's search list. It takes a string as an argument, and the server treats it exactly as though it were destined for SETSRC's C command. In fact, the CMU server sends it to SETSRC (all file related functions are performed on a PTY job). This is an example of a locally defined function - SEARCH will work only at DEC 10/50 sites.

* CPATH - change default path. *

This command will modify your default path specification, and will work not only at CMU but also at TENEXes. Its argument is a user specification, and successful completion of it will mean that all file system references will go to that person's DSK area instead of your own. At DEC 10/50 sites, it may also be used to specify the SFD path.

* * File transfer commands. * *

The transfer commands are the heart of the FTP system. these are the ones which effect most of the data flow through the net. While they are running, a second connection, called the data connection is opened, and all data flows through it. All the transfer commands have the same possible formats. for an arbitrary command XFER, legal formats are as follows.

Basic: XFER DEV:FILE.EXT[PPN];PATHNAME

(PATHNAME. is the foreign file name) DEV:FILE.EXT[PPN] is the local PDP-10 file name in all cases. Note that this is unlike PIP, so that even when you send a file through the net, the local name comes first. It must be done this way or else FTP would not be able to find the local file name since it makes no assumption on what the pathname format is.

Copy file: DEV:FILE.EXT/STRING

This form is intended primarily for use with stanford. It will copy the FILE and EXT and append everything after the /, use that as the pathname. E.G.:
XFER APE.ME/[UP,DOC]
is equivalent to: XFER APE.ME;APE.ME[UP,DOC]

Special: XFER DEV:FILE.EXT[PPN]

FTP USER PROGRAM DOCUMENTATION
Version 3E

5

This form is meant primarily for systems much like DEC's. It will act in two different ways, depending upon the direction of transfer. For commands that read from the net, it will act as:

XFER FILE.EXT;DEV:FILE.EXT[PPN]

and for commands that write out to the net, it will act as:

XFER DEV:FILE.EXT[PPN];FILE.EXT.

This command should be sufficient to handle most transfers involving CMU, Harvard, and some to Stanford and TENEXes.

In all cases, the local device will default to DSK:, and PPN may be a CMUPPN only at CMU.

After you type the CR, FTP and the foreign server will trade a couple commands before the actual transfer starts. The 250 message from the server is an acknowledgement that the transfer is set to go. (FTP may have figured that out itself and started the transfer before this message prints.) When FTP finishes the transfer, it will print the information about the number of bytes sent and the baud rate, then accept another command when the server says it is finished with its work.

These are the currently defined file transfer commands:
RETRIEVE: This will get a file from the server.

STORE: This will send a file to the server.

LIST: This will get a directory for the area specified in the pathname. Thus, to get the directory of the system area at a TENEX site, one would use this command:
LIST LPT:;<SYSTEM>

MLFL: This is the other way to send mail via FTP. MLFL will send the local file specified to the mail box system at the foreign site, to the user specified in PATHNAME.

* * File access control commands. * *

The ability to transfer files alone is not enough. Some files have special formats, some transfer modes are more efficient, the capability to rename and delete files is necessary, and one may wish 'watch' the file be transferred. To fill these needs, there are several file access commands.

* BYTE - set transfer byte size. *

The BYTE command will set the desired byte size of the transfer mode. Currently only byte sizes of 8 bits and 36 bits are allowed, with 36 being the more efficient. The server must be told of the byte

FTP USER PROGRAM DOCUMENTATION 6
Version 3E

size, therefore, it will send a message to the server if the TELNET connection is open. FTP will remember what the server believes are the current transfer parameters, and the transfer commands will ensure that the server is kept upto date. Therefore, BYTE and TYPE (below) commands done in * mode will be in effect after a FTP connection is opened.

* TYPE - set transfer mode. *

The TYPE command sets another transfer characteristic, that of the file type. Only two types are currently implemented: ASCII and IMAGE. Only the first character of the type need be specified. FTP initializes itself to image type, 36 bit byte size ('IMAGE 36'), which is not the network default, but is the most efficient mode. A 'TYPE A' command will temporarily assume a byte size of 8 bits, since FTP definition requires it. Only in image type will the byte size specified in the BYTE command be used.

* HASH - watch the transfer. *

The HASH command complements the value of the hash flag. Initially off, the HASH command will cause the file transfer commands to print a # character after every buffer is transferred.

* STATUS - return status of server or files. *

The STATUS command will return the current status of the server if no argument is given, or return a directory if a PATHNAME is given. Further, it will give the current states of the HASH and VERBOSE (described later) flags.

* DELETE - delete a file from the foreign site. *

The delete command will accept a pathname as an argument, which it passes to the server to delete.

* RENAME - rename a file at the foreign site. *

The rename command will accept two pathnames, on different lines. The first is the current name of the file, the second is the desired new name.

* BYE - terminate server operation. *

When you are done transferring files back and forth, the BYE command will tell the server that you are going away. Some servers (Stanford) will not say goodbye to you, since they expect you won't be listening. Others (MII-AI) will send a friendly see you later message. Since the QUIT command will now do a BYE if you haven't, the BYE command might as well be ignored. (NOW he tells us!)

FTP USER PROGRAM DOCUMENTATION
Version 3E

7

* * DEBUGGING AND MISCELLANEOUS COMMANDS. * *

* HELP - see who knows what. *

If you should need a list of the current FTP commands, typing help will give them to you. In addition, in ! mode, FTP will give you all the server's help information too.

* SLEEP - good for slow systems. *

To avoid the many possible deadlock situations inherent in network communications, FTP will normally wait a finite period of time for any server replies it expects. Sometimes FTP doesn't wait long enough, decides that something is messed up, and gets itself a little confused in the process. For example, the MLFL function to load TENEXES has often timed out, and FTP says 'transfer aborted' although the server eventually finished. To increase the timeout period, the SLEEP command is used to set a sleep factor which is used to multiply all future timeouts. For example, if FTP would normally wait 10 seconds for a reply, if a 'SLEEP 2' command were done, it would now wait $2 \times 10 = 20$ seconds.

* VERBOSE - to see all server messages. *

Much of what the server sends to FTP is very uninteresting to a person who is more concerned about getting a file instead of seeing how it's done. Therefore, FTP attempts to determine the usefulness of server messages and doesn't print most of them on the user's terminal. If you are interested in seeing what's being said behind your back, the VERBOSE command will toggle you in and out of verbose mode.

Example:

!TYPE A

!VERBOSE

Verbose mode

!TYPE A

200 TYPE A accepted ;all commands have replies

* AUTO - process a command file. *

The AUTO command takes a filename argument which it will treat as a series of FTP commands. If the file has line numbers, both line numbers and SOS-type page marks will be ignored. Unlike most command file processors, all commands will be echoed so you can watch it bomb. A minor bug: the extension will be defaulted to CMD, even if you specify a null extension (e.g. 'AUTO ORK.') - this may be fixed sometime in the future.

Example:

To produce the sample run at the end of this listing, one could use this command file:

HOST CMUB

FTP USER PROGRAM DOCUMENTATION 8
Version 3E

```
Y
PASSWORD
RETRIEVE FTP.DOC;SYS:FTP.DOC
MAIL WERME
AN EXAMPLE OF FTP-MAIL.
$
QUIT
QUIT
To use it:
.R FTP
FTP V3E
*AUTO FTP.EXM
*HOST CMUB....
```

* COPY - Copy string to server verbatim. *

The COPY command copies the rest of the line directly to the server, thus providing a convenient way to implement any FTP command not previously covered.

Example:

```
ICOPY XCMD          ;send an experimental XCMD
                   ;CMD to the server.
```

!

* DDT - enter DDT. *

The DDT command will put you into DDT, if it's loaded. What you'll do there I'm not really sure, but I find it handy for debugging.

* ICP - Set ICP socket number. *

The ICP command redefines which socket is to be used for the ICP (Initial Connection Protocol). It takes a decimal argument, like the LOCAL command and others. It is meant primarily to debug new servers running as normal programs.

* XPATCH - crosspatch to the server. *

The XPATCH command has even less use. It crosspatches your teletype to the server, so anything you type goes directly to the server. It is useful only for very low level server debugging.

AN EXAMPLE OF USING FTP.

```
.R FTP
FTP V3
*HOST CMUB
Want to Login? (Y or CR): Y ;Free USER cmd
Password:                  ;FTP will turn off echoing
```

FTP USER PROGRAM DOCUMENTATION 9
Version 3E

000 2250 23-Mar-73 Fri
TH 1500 SYS:NEWS(3-21)
!RETRIEVE FTP.DOC;SYS:FTP.DOC ;Note: RETRIEVE SYS:FTP.DOC
;will have the same effect

250 RETR started.
2269 bytes, 22.0 KBD
!MAIL WERME
AN EXAMPLE OF FTP-MAIL.

\$

!QUIT
*QUIT

. ;back in monitor mode.