# Chapter 24

# The Interface Message Processor for the ARPA Computer Network[1]

*F. E. Heart / R. E. Kahn / S. M. Ornstein / W. R. Crowther / D. C. Walden*

## 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.[2] 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 Fig. 1.) This approach divides the genesis of the ARPA Network into two parts: (1) design and implementation of the IMP subnet, and (2) design

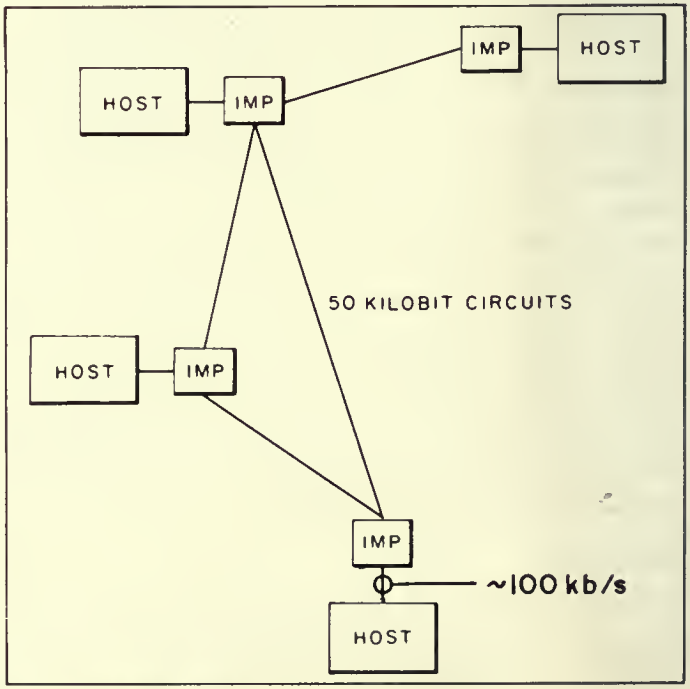


**Fig. 1. Hosts and IMPs.**

[1] *Proc. AFIPS SJCC*, 1970, pp. 551–567.

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

and implementation of protocols and techniques for the sensible utilization of the network by the Hosts.

Implementation of the subnet involves two major 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 connecting 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 Fig. 2. The connections between the first four sites are indicated by solid lines. Dotted lines indicate planned connections.

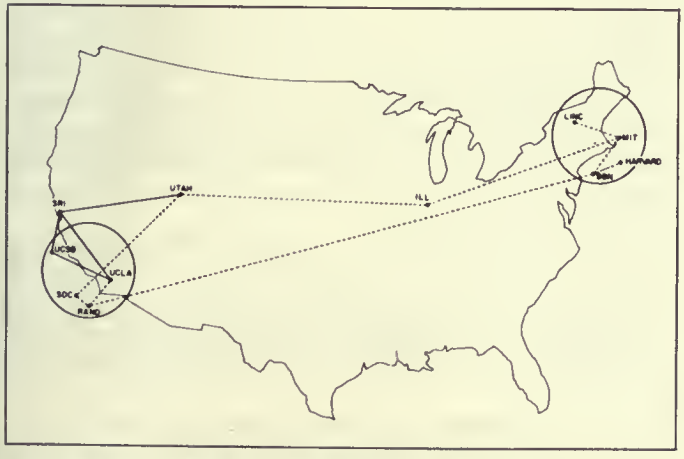


**Fig. 2. Network map (from the ARPANET Directory).**

## 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 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 μsec. 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 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 invisible 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

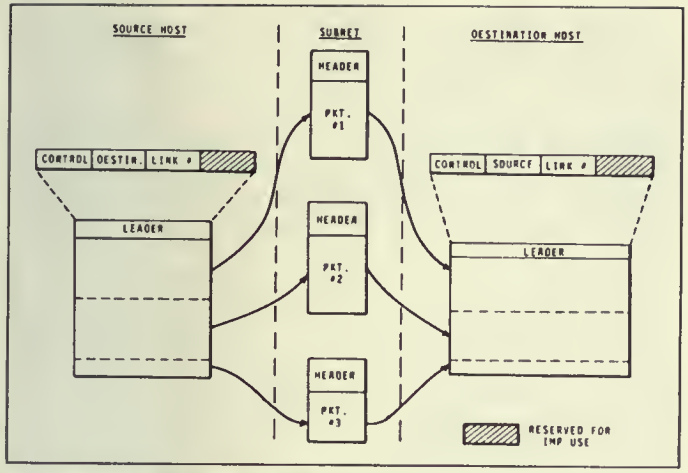


**Fig. 3. Messages and packets.**

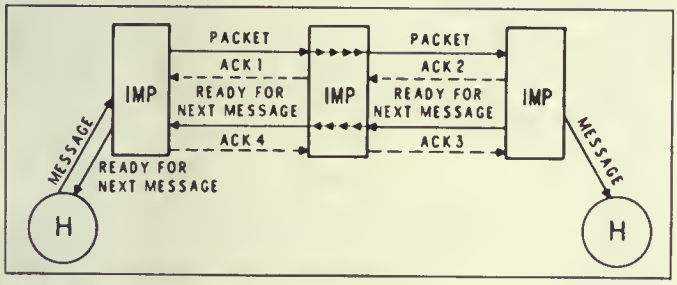


**Fig. 4. RFNMs and acknowledgments.**

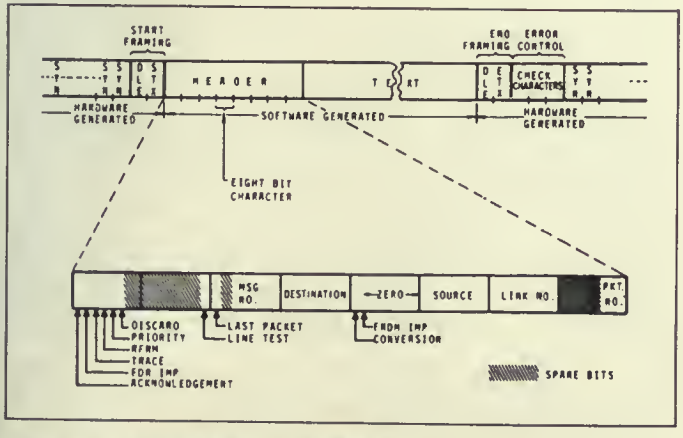


**Fig. 5. Format of packet on phone line.**

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 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 resassembled. 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 Fig. 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 half second 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 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 into 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 miliseconds, 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 approximately 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 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-μsec 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-μsec 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 maintenance, 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.

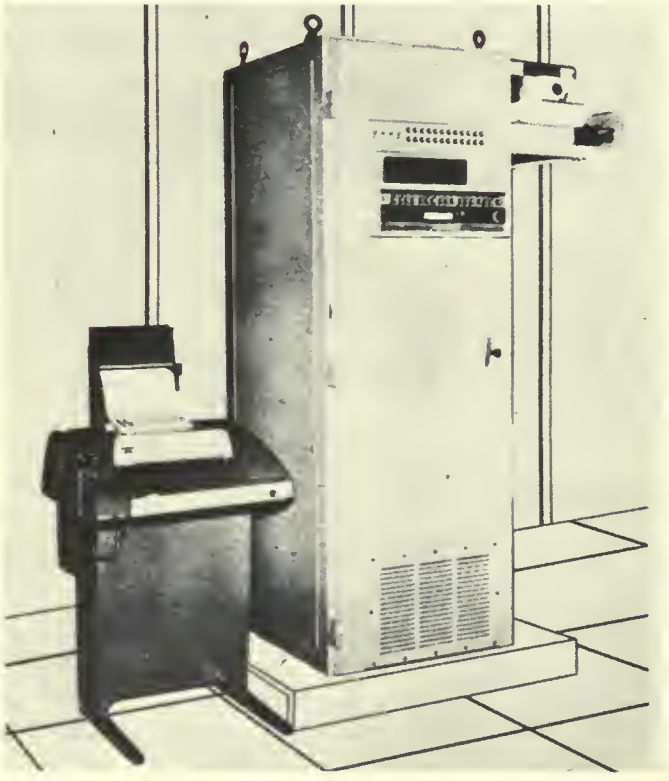

Fig. 6. The IMP.

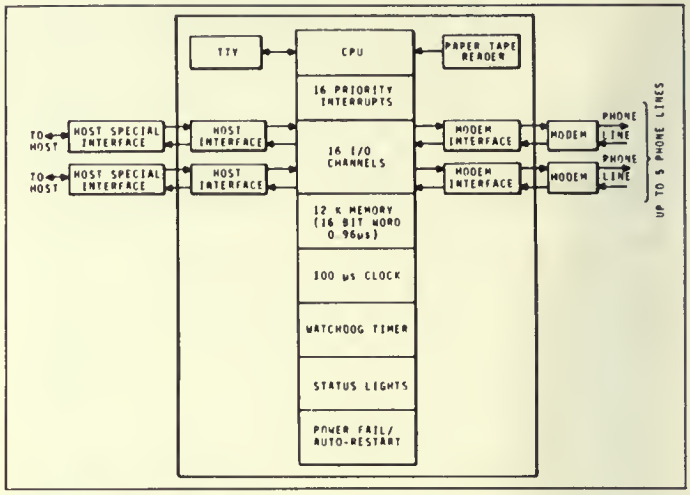


Fig. 7. IMP configuration.

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, reconfiguration 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 10-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 modern 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 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 functionally 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 Fig. 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 contains the remaining five programs and deals with initialization, debugging,

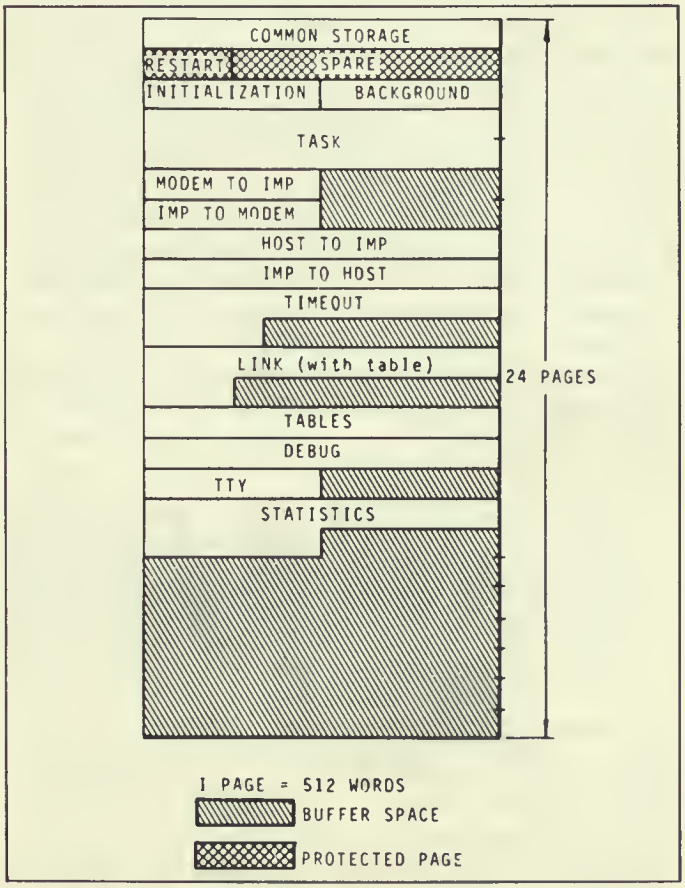


**Fig. 8. Map of core storage.**

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 1) 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.

**Table 1 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 |

*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 → 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,

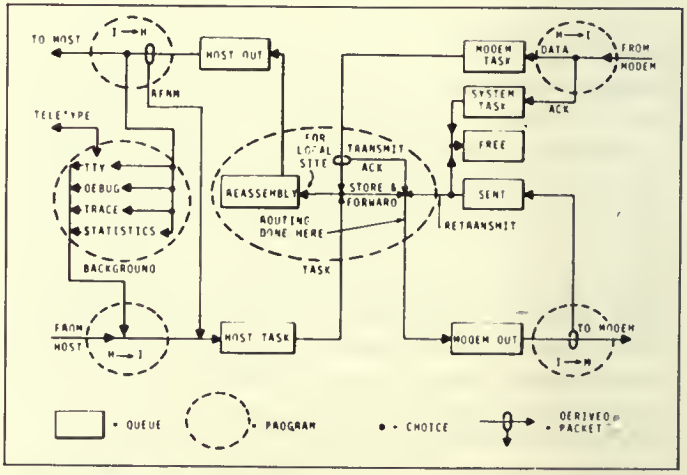


**Fig. 9. Internal packet flow.**

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 → 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. Whenver 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 → M) transmits successful 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 → 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 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          S     M
```

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 outgoing 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.

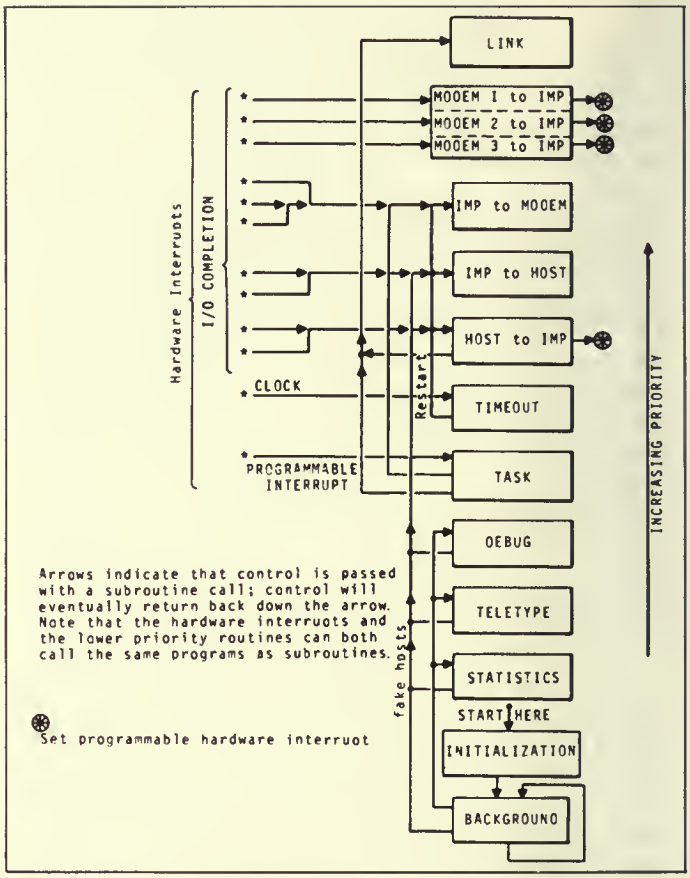


**Fig. 10. Program control structure.**

**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 Fig. 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 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 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 μsec 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 μsec 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 2, we show an estimate of the one-way and round-trip transit times and the corresponding maximum message

**Table 2 Transit Times and Message Rates**

| | *Minimum* | *Maximum* |
|---|---|---|
| | *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 |

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 Fig. 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 is 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

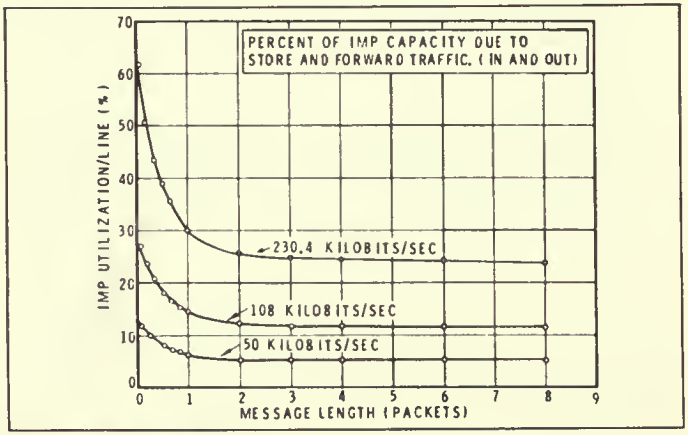


Fig. 11. IMP utilization.

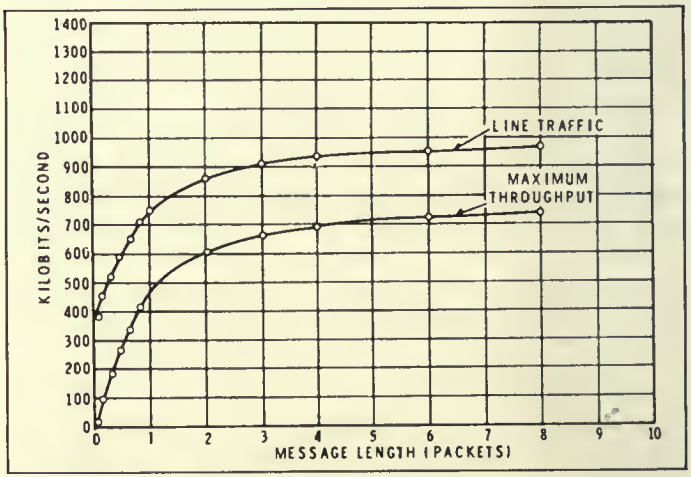


Fig. 12. IMP throughput.

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 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, 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.

## References

Baran [1964]; Baran, Boehm, and Smith [1964]; Boehm and Mobley [1966]; BBN Report No. 1763 [1969]; BBN Report No. 1822 [1969]; Brown, Miller, and Keenan [1967]; Carr, Crocker, and Cerf [1970]; Cuadra [1968]; Davies [1968a]; Davies [1968b]; Davies, Bartlett, Scantlebury, and Wilkinson [1967]; EDUCOM *EIN Catalog*; Everett, Zraket, and Benington [1957]; FCC [1966a]; Ford and Fulkerson [1962]; Frank, Frisch, and Chóu [1970]; James [1966]; Kaplan [1968]; Kleinrock [1964]; Kleinrock [1969]; Kleinrock [1970]; Marill [1966]; Marill and Roberts [1966]; National Library of Medicine [1968]; *NOC Symp.* [1968]; *NOC Symp.* [1969]; Perry and Plugge [1961]; Roberts [1967]; Roberts [1968]; Roberts [1969]; Roberts and Wessler [1970]; Scantlebury, Wilkinson, and Bartlett [1968]; Steiglitz, Weiner, and Kleitman [1969]; Sung and Woodford [1969]; Teitelman and Kahn [1969].