# 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.

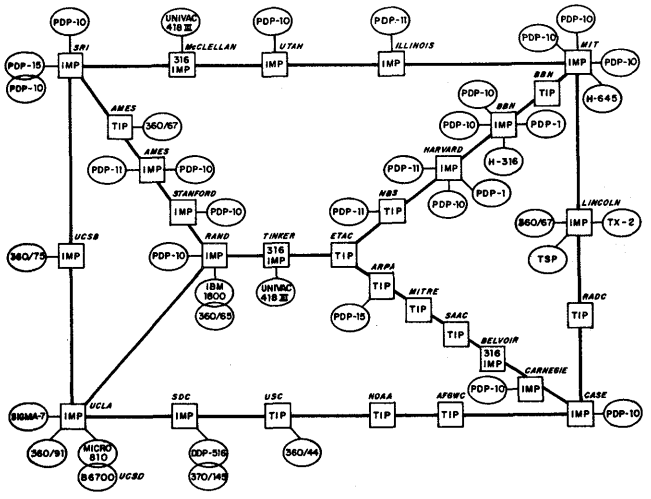


Figure 1—ARPA network, logical map, August 1972

## 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

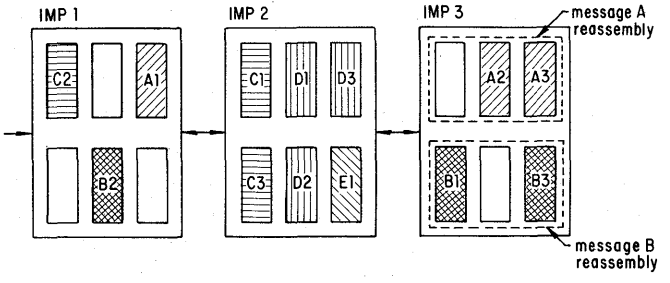


Figure 2—Reassembly lockup

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 requring 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 "piggybacked" 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 acknowledges 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 acknowledges 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!*

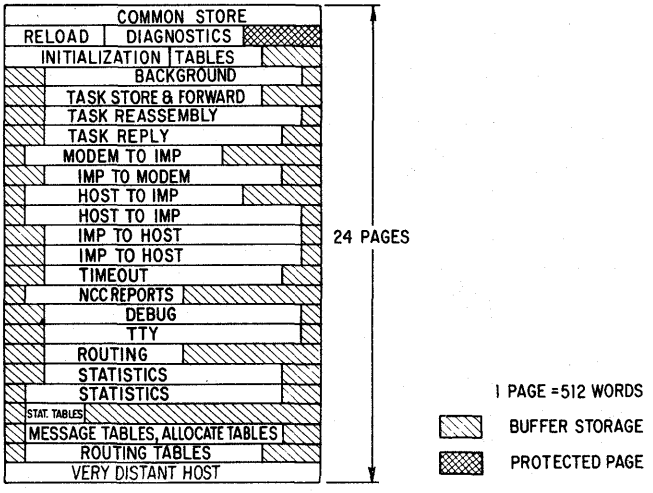


Figure 3—Map of core storage

## 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 Host[9]), 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.

* 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

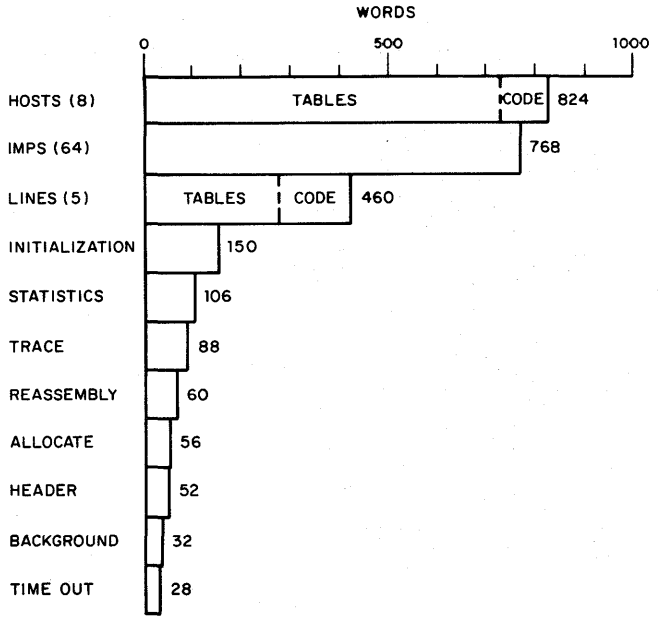


Figure 4—Allocation of IMP table storage

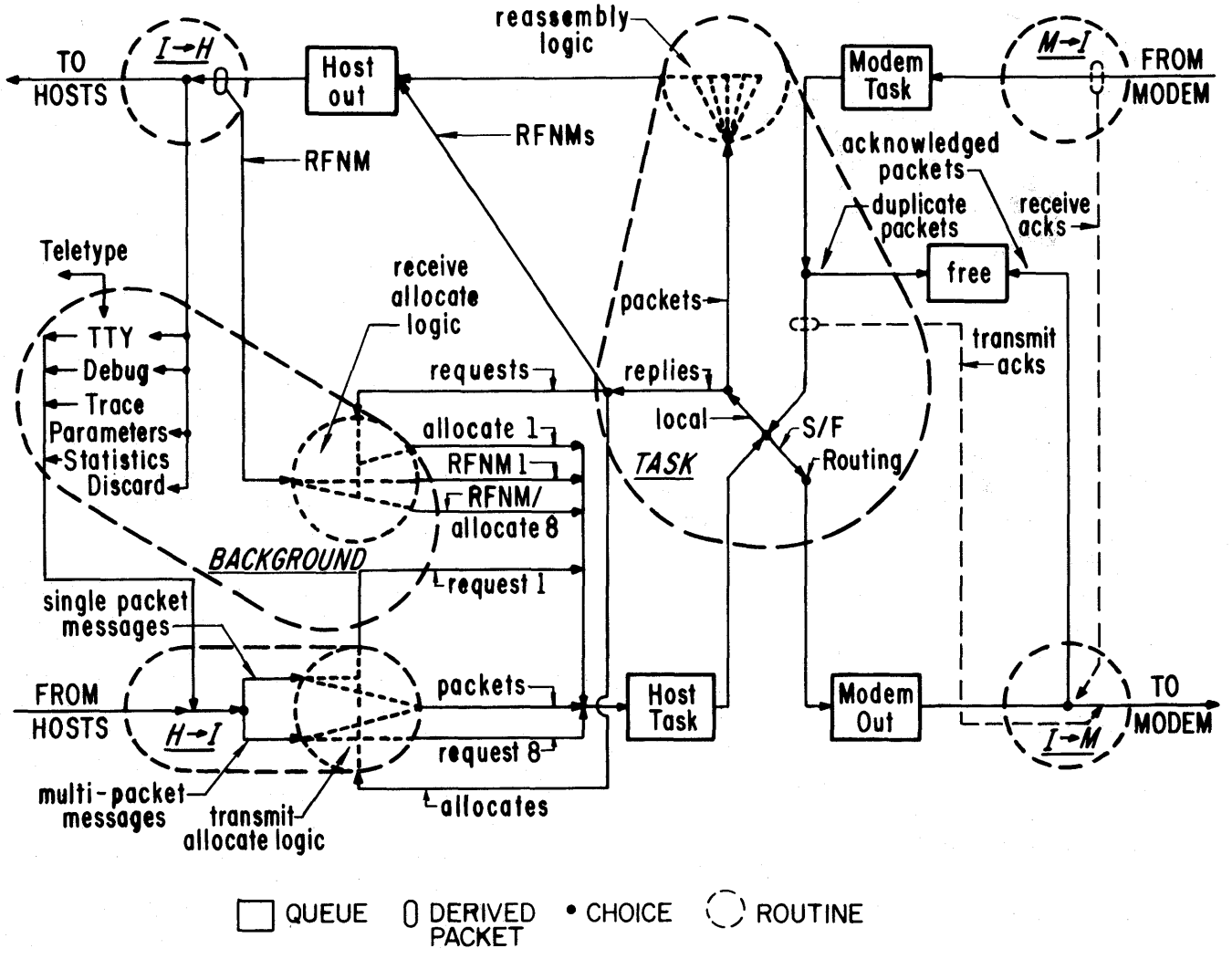


Figure 5—Packet flow and processing

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 transmi 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 or some resource to become available, and are written as coroutines 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:

| | *Original* | *Modified* |
|---|---|---|
| $D$ | 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.

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.

* 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 μsec, and the 316 has a cycle of 1.6 μsec. 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 μsec for the 516 and 16 bits per 3.2 μsec for the 316.

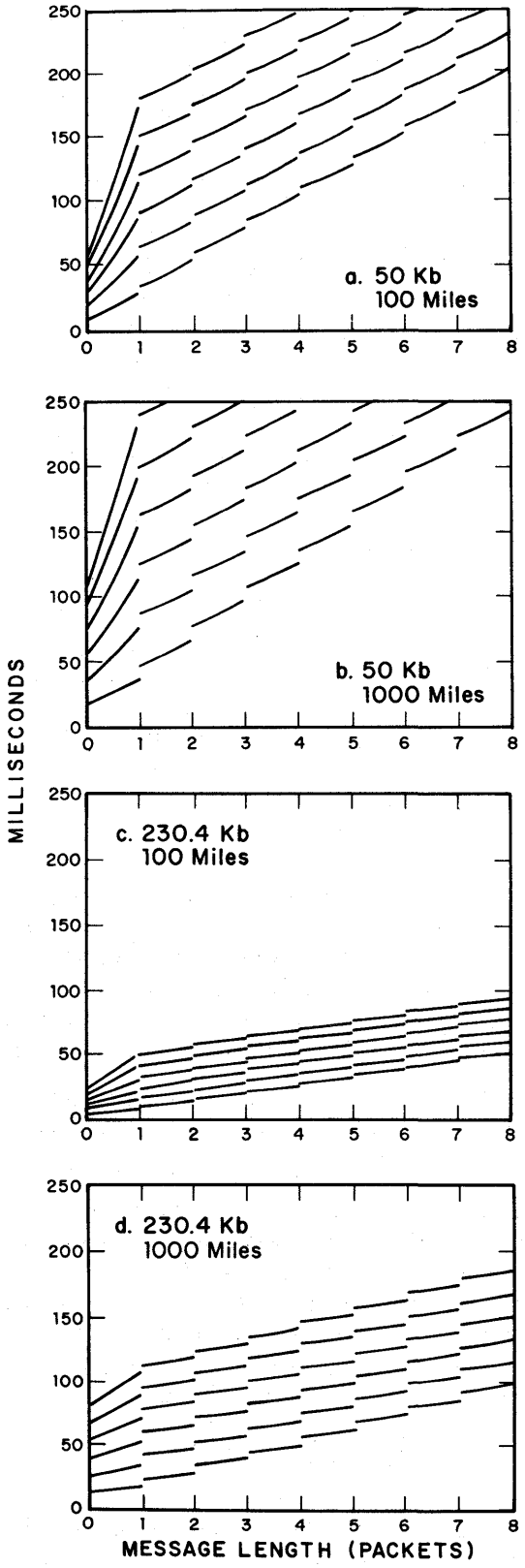


Figure 6—Line traffic and throughput vs. message length. The upper curves plot maximum line traffic, the lower curves plot maximum throughput

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]

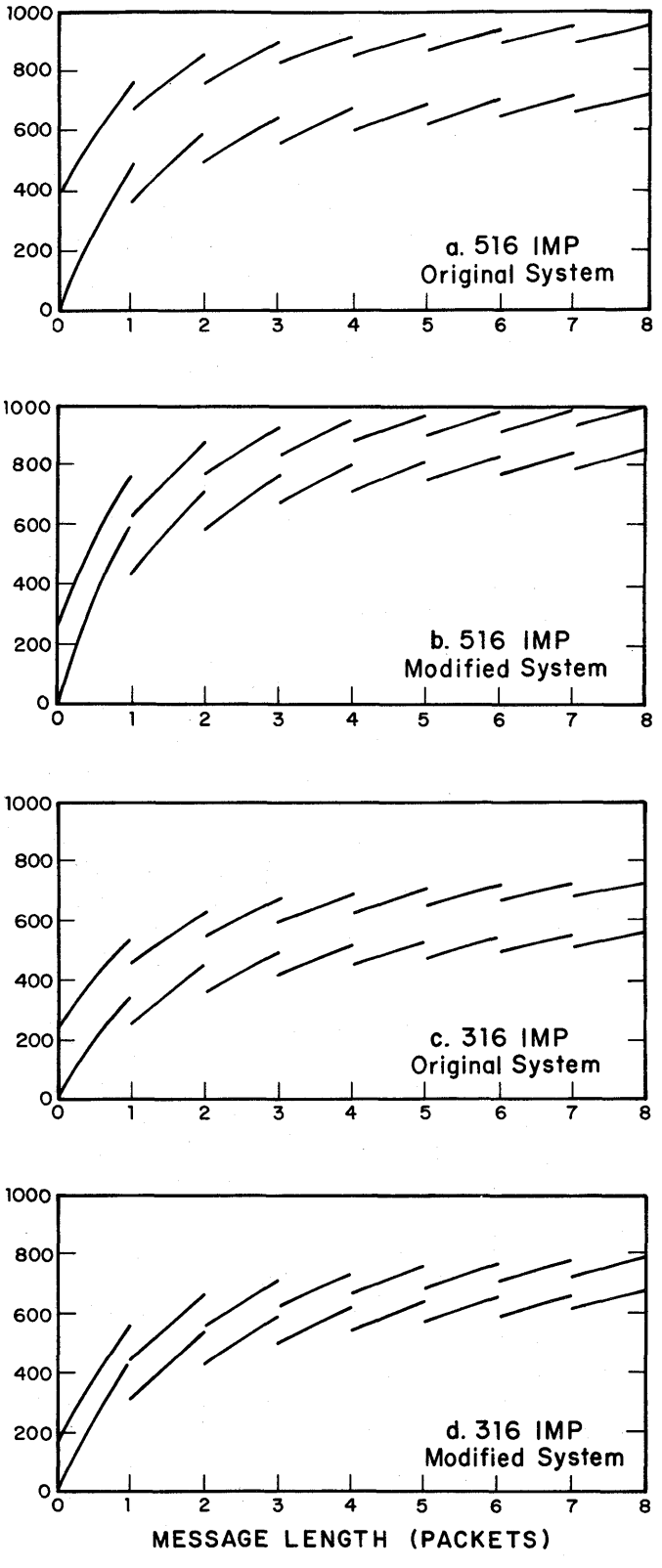


Figure 7—Minimum round trip delay vs. message length. Curves show delay for 1-6 hops

*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 $\mu$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.

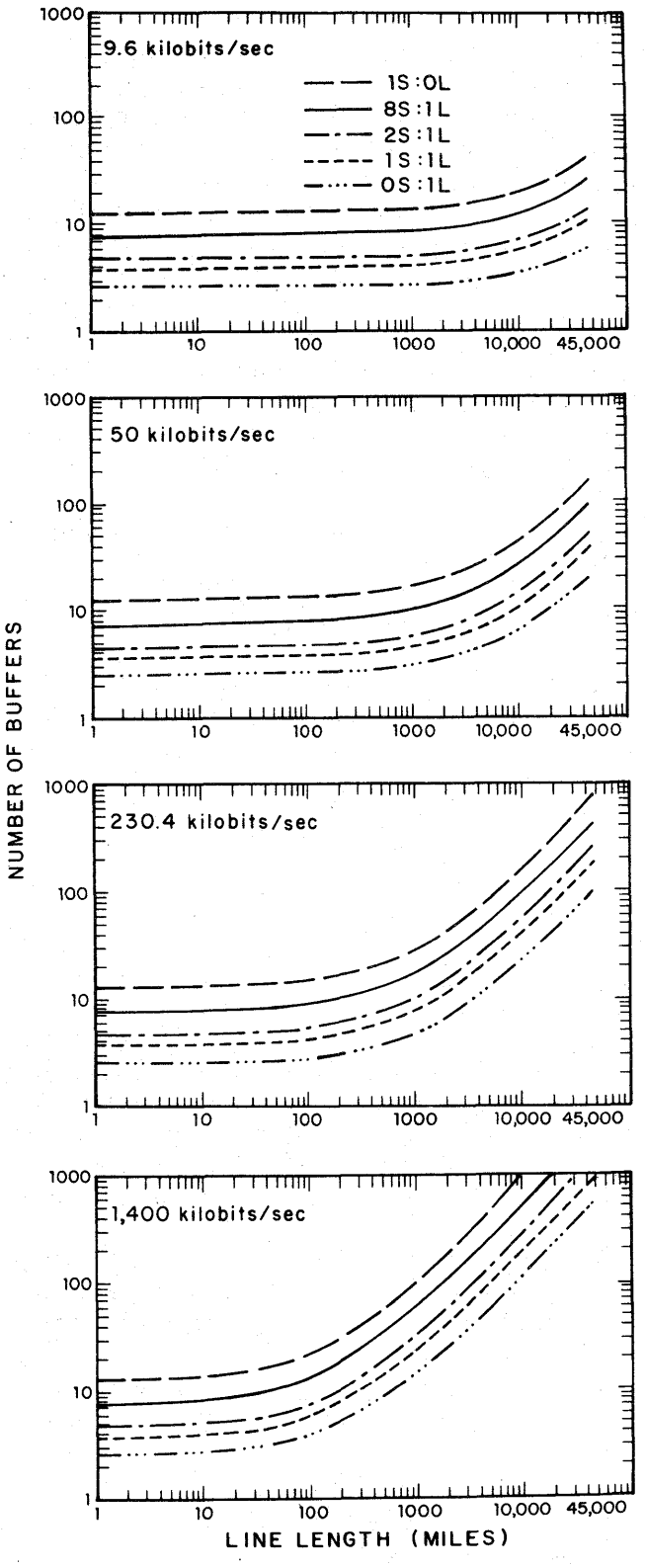


Figure 8—Number of buffers for full line utilization. Traffic mixes are shown as the ratio of number of short packets (S) to number of long packets (L)

light" delay, and ranges from 50 μsec 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 subnetwork. 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 E KAHN
*Terminal access to the ARPA computer network*
Courant Computer Symposium 3—Computer Networks
Courant Institute New York November 1970

L KLEINROCK
*Analytic and simulation methods in computer network design*
Proceedings of AFIPS 1970 Spring Joint Computer Conference
Vol 36 pp 569-579

A A McKENZIE B P COSELL J M McQUILLAN
M J THROPE
*The network control center for the ARPA network*
To be presented at the International Conference on Computer
Communications Washington D C October 1972

L G ROBERTS
*Extension of packet communication technology to a hand-held personal terminal*
Proceedings of AFIPS 1972 Spring Joint Computer Conference
Vol 40 pp 295-298

L G ROBERTS B D WESSLER
*Computer network development to achieve resource sharing*
Proceedings of AFIPS 1970 Spring Joint Computer Conference
Vol 36 pp 543-549

R THOMAS D A HENDERSON
*McROSS—A multi-computer programming system*
Proceedings of AFIPS 1972 Spring Joint Computer Conference
Vol 40 pp 281-294