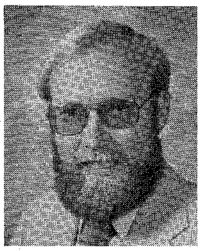
The ARPA Network Design Decisions*

John M. McQuillan and David C. Walden

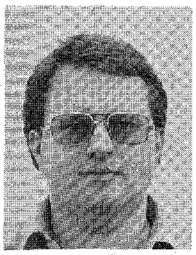
Bolt Beranek and Newman Inc., 50 Moulton Street, Cambridge, Massachusetts 02138, USA

A number of key decisions made in the design of the ARPA Network over a five-year period serve as the context for an analysis of the fundamental properties and requirements of packet-switching networks and formulation of the fundamental criteria for evaluating network performance. The decisions described fall into the threee major areas of network equipment design, store-and-forward subnetwork system design, and source-to-destination system design, and each decision is examined in detail.

Key words: Packet-switching, protocols, networks, computer communication, interprocess communication, internetting, ARPANET.



John McQuillan received his B.A., M.A., and Ph.D. degrees in Applied Mathematics from Harvard University. He has worked in the Computer Systems Division of Bolt Beranek and Newman Inc. since 1971. At the time this paper was written, Dr. McQuillan had responsibility for development and maintenance of the ARPA Network "IMP" software. Over the past several years he has been active in studying advanced computer communications systems and network architectures. He is presently Manager of Bolt Beranek and Newman's Systems Analysis Department, which specializes in such studies.



David Walden graduated from San Francisco State College. Since then he has worked in the Space Communications Division of M.I.T.'s Lincoln Laboratory, for A/S Norsk Data-Elektronikk in Oslo, Norway, and in the Computer Systems Division of Bolt Beranek and Newman Inc. Mr. Walden is one of the original team which designed and implemented the ARPA Network "IMP". He is currently Assistant Division Director of Bolt Beranek and Newman's Computer Systems Division, where his specialty is computer communications systems.

Preface

This paper was prepared at the time (July 1975) that the Advanced Research Projects Agency (ARPA) Network was turned over by ARPA to the Defense Communications Agency and the network changed from an experimental system to an operational one. The network design decisions that form the context of the paper are therefore the decisions that had been made up to the time of transition. Of course the network has continued to evolve, since July 1975, beyond the state described here.

1. Introduction

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

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

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

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

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

* The work described here was supported under Contracts DAHC15-69-C-0179, F08606-73-C-0027, and F08606-75-C-0032.

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

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

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

Packet. The unit of data exchanged between adjacent nodes.

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

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

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

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

Node–node transmission procedures. The set of procedures governing the flow of packets between adjacent nodes.

Source–destination transmission procedures. The set of procedures governing the flow of messages between source node and destination node.

Host–node transmission procedures. The set of procedures governing the flow of information between a Host and the node to which that Host is directly connected.

Host–host transmission procedures. The set of procedures governing the flow of information between the source Host and the destination Host.

Within the class of network under consideration, there are already several operational networks and many network designs. The ARPA Network [17] is made up of over fifty node computers called Interface Message Processors or IMPs and over seventy Hosts. The Cyclades Network [33] is a French network consisting of about six nodes and about two Hosts per node. The Societe Internationale de Telecommunication Aeronautique (SITA) Network [7] connects centers in eight or so cities mostly in Europe. The European Informatics Network (EIN) [3], also known as Cost-11, is currently in a design stage and will be a network interconnecting about six computers in several Common Market countries. Some other packet-switching network designs include: Audodin II [35], NPL [11], PCI [1], RCP [13], and Telenet [1].

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

As the ARPA Network is discussed at length throughout the remainder of this paper, we next summarize how the IMP in the ARPA Network performs its functions as a message switching center and interface between Host computers. Fig. 1 shows a diagram of message flow in the ARPA Network and illustrates some of the terminology. The Host sends the IMP a *message* with up to 8063 data bits. The source IMP breaks this up into *packets* with up to 1008 data bits. When the packet is successfully received at each IMP, an *acknowledgment* or *ack* is sent back to the previous IMP. When the message arrives at the destination IMP it is *reassembled,* that is, the packets are combined into a message again. The message is sent to the destination Host and when it has been accepted, a *Ready For Next Message* which we abbreviate as *RFNM* is sent back to the source Host. The RFNM is also a packet and it is acknowledged. Several points are worth noting. First, acks are not actually separate transmissions, but are piggy-backed in packets to cut down on overhead. Next, packets on the links

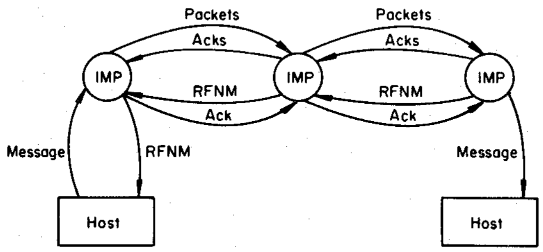


Fig. 1. Message flow in the ARPANET.

between IMPs are checksummed in the modem interface hardware and the IMP employs a positive acknowledgment retransmission scheme: that is, if a packet is in error, it is not acknowledged. It is then retransmitted until an acknowledge is received. Further, an IMP may send the several packets of a message out on different links. Because of retransmission (out of order) of a packet on a link and transmission of packets on alternate links, the packets of a message may arrive at the destination IMP out of order and must be reassembled into the correct order for transmission into the Host.

2. Fundamental issues

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

2.1. Network properties and requirements

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

a. Finite, fluctuating delay–A result of the basic line bandwitdth, speed of light delays, queueing in the nodes, line errors, etc.

b. Finite, fluctuating bandwidth–A result of network overhead, line errors, use of the network by many sources, etc.

c. Finite packet error rate (duplicate or lost packets)–A result of the acknowledgment system in any store-and-forward discipline (this is a different use of the term "error rate" than in traditional telephony).

Duplicate packets are caused when a node goes down after receiving a packet and forwarding it without having sent the acknowledgment. The previous node then generates a duplicate with its retransmission of the packet. Packets are lost when a node goes down after receiving a packet and acknowledging it before the successful transmission of the packet to the next node. An attempt to prevent lost and duplicate packets must fail as there is a tradeoff between minimizing duplicate packets and minimizing lost packets. If the nodes avoid duplication of packets whenever possible, more packets are lost. Conversely, if the nodes retransmit whenever packets may be lost, more packets are duplicated.

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

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

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

e. Finite storage–A property of the nodes.

f. Differing source and destination bandwidths–Largely a property of the Hosts.

(Note: the question is frequently raised whether the source and destination nodes or the source and destination Hosts should solve these problems. This question is addressed in a later section.)

A slightly different treatment of this subject can be found in [34].

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

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

b. Pipelining–The finite bandwidth of the network may necessitate the pipelining of each message flowing through the network by breaking it up into packets in order to decrease delay. The bandwidth of the circuits may be low enough so that forwarding the entire message at each node in the path results in excessive delay. By breaking the message into packets, the nodes are able to forward the first packet of the message through the network ahead of the later ones. For a message of P packets and a path of H hops (with small propagation delays), the minimum delay is proportional to $P + H - 1$ instead of $P*H$, where the proportionality constant is the packet length divided by the transmission rate. (See section 2.2.1 below for a derivation and more exact results.)

c. Error control–The node-to-node packet processing algorithm must exercise error control, with an acknowledgment system in order to deal with the finite packet error rate of the circuits. It must also detect when a circuit becomes unusable, and when to begin to use it again. In the source-to-destination message processing algorithm, the destination may need to exercise some controls to detect missing and duplicated messages or portions of messages, which would appear as incorrect data to the end user. Further, acknowledgments of message delivery or non-delivery may be useful, possibly to trigger retransmission. This mechanism in turn requires error control and retransmission itself, since the delivery reports can be lost or duplicated. The usual technique is to assign some unique number to identify each data unit and to time out unanswered units. The error correction mechanism is invoked infrequently, as it is needed only to recover from node or line failures.

d. Sequencing–Since packet sequences can be received out of order, the destination must use a sequence number technique of some form to deliver messages in correct order, and packets in order within messages, despite any scrambling effect that may take place while several messages are in transit. The sequencing mechanism is frequently invoked since it is needed to recover from line errors.

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

f. Flow control–The different source and destination data rates may necessitate implicit or explicit flow control rules to prevent the network from becoming congested when the destination is slower than the source. These rules can be tied to the sequencing mechanism, with no more messages (packets) accepted after a certain number, or tied to the storage allocation technique, with no more messages (packets) accepted until a certain amount of storage is free, or the rules can be independent of these features. In satisfying the above six requirements, the algorithm often exercises contention resolution rules to allocate resources among several users. The twin problems of any such facility are:

fairness–resources should be used by all users fairly (perhaps in accordance with appropriate priority rules);

deadlock prevention–resources must be allocated so as to avoid deadlocks.

Notice that each of these algorithms is implemented by a *distributed computation.* IMP-to-IMP transmission control involves cooperation between every pair of neighboring IMPs. Source-to-destination transmission control is a distributed process between the pair of IMPs exchanging a message. Finally, flow control and routing are distributed algorithms which involve all the IMPs in the network. Each IMP makes local decisions about global functions. The process of routing messages from source to destination involves all the IMPs in the network, in order that the best path for the message be chosen and agreed upon. Such distributed computations are quite different from conventional algorithms. They are not initialized, nor do they run to completion and halt. In a real sense, the ARPA routing calculation, flow control techniques, and so on, have been in progress for 5 years now, since some part of the Network has always been running for all of that time. These algorithms are continuously active processes on a large number of different processors. In fact, the number of processors and the interconnection between them is subject to change at any moment. They must run completely without human intervention. They perform contention resolution among bidders for shared resources, and they must do so without races or deadlocks. (We have also come to believe that it is essential to have a reset mechanism to unlock "impossible" deadlocks and other conditions that may result from hardware or software failures.)

2.2. Network performance goals

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

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

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

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

2.2.1. Delay

In this section we consider what delay performance characteristics are possible in a given network. The topics discussed below include the identification of the components of delay, an analysis of the minimum delay possible for a round trip through a network, the reason for breaking messages up into packets, a brief look at queueing delay, and a discussion of delays for interactive traffic. Some of this material was presented in [24], and some of the best other references to this subject matter are [20,22, 28].

Components of delay. We will first discuss the delay experienced by a single packet transmitted over a single hop. The components of delay which we will take to be fundamental variables are as follows:

1. L = propagation delay or speed of light latency. This is the delay for the first bit of the packet to traverse the circuit. L = (circuit length [mile])/(signal propagation rate [mile/sec]).

2. T = transmission delay. This is the time for the bits of a packet to be clocked out on the circuit. T = (number of bits in packet [bits])/(transmission rate [bits/sec]).

In the analysis below we will use T_p to denote T for a packet and T_r to denote T for a RFNM, which we will assume is a minimum length packet.

3. C = nodal processing delay. This is the time it takes the node to process the packet. It has two fixed components, corresponding to the store operation and the forward operation, or receive and transmit, plus a random component due to queued tasks of higher priority. This component measures the interference experienced by packets queued for processor service. (In the following, we ignore delays at the source and destination nodes due to message processing.)

$$C = \underset{\text{receive}}{C_r\,[\text{sec}]} + \underset{\text{transmit}}{C_t\,[\text{sec}]} + \underset{\text{queueing -- random}}{C_q\,[\text{sec}]}.$$

A typical range of C is probably 1 to 10 milliseconds for an IMP-like node.

4. D_q = queueing delay. This is the time that the packet must wait for the transmission of the packets which precede it on the output queue, including the output time of the packet currently being transmitted. Thus, C_q measures input queueing delay, waiting for central processor service, while D_q measures output queueing delay, waiting for circuit service.

5. D_r = delays due to retransmissions. This is the time that the packet must wait in the event that its first transmission is unsuccessful. This may happen if it was in error or if the other node refused it for some

Table 1

Some representative propagation delays

Distance (miles)	L (msec)	Line type
10	0.054	intra-city line
100	0.54	inter-city line
1000	5.4	long line
3000	16.2	cross-country line
10000	54	very long terrestrial circuit
45000	272	satellite link

reason, or, in the case of broadcast circuits, there was a collision with another packet.

In general, we will assume that the first two components are much greater than the last three. Tables 1 and 2 give some representative values for L and T. (Note that a 50 Kbs circuit can transmit 1000 bits in 20 ms and that the first bit can travel about 4000 miles in that time. So a bit is about 4 miles long at 50 Kbs!)

Minimum round trip delay. Now we can examine the minimum round trip delay, by taking the case of $C_q = D_q = D_r = 0$. Consider a message with P packets traveling over a path of H hops. If the delay at hop i is

$$D(i) = L(i) + T(i) + C(i),$$

we can define the natural quantity D(ave), average hop delay, as follows:

$$D(\text{ave}) = (1/H)*\sum_{i=1}^{H} [L(i) + T(i) + C(i)]. \qquad (1)$$

Table 2

Some representative transmission delays

	Short packet $T_p = T_r$ 152 bits	Long packet T_p 1160 bits
Circuit bandwidth (kB)	(msec)	(msec)
9.6	15.7	120.5
50	3.04	23.2
230.4	0.66	5.03
1344	0.106	0.81

We can also define the less obvious variable D(max):

$$D(\max) = \sum_{i=1}^{H} [T(i) + C_t(i)]. \qquad (2)$$

With these definitions, we can make the following two statements:

1. The first packet experiences delay equal to $H*D(\text{ave})$.

2. The remaining $P - 1$ packets follow through the network, each packet at most one hop behind the preceding packet; this adds $(P - 1)*D(\max)$ to the total delay.

This analysis can be illustrated by the numerical example of table 3. Note that the bottleneck hop, in this case hop 2, has the largest $T(i) + C_t(i)$, here equal to 3.5, but not the highest total delay. That is, hop 3 has total delay of 6, compared to the delay of 5 over hop 2. Note also that more than one packet is being transmitted at the same time, giving a pipelined effect, and reducing the total delay for the message.

This means that the delay for a single packet message is

$$D(\text{SP}) = \sum_{i=1}^{H} [L(i) + T_p(i) + C(i)]. \qquad (3)$$

and the delay for a multipacket message is

$$D(\text{MP}) = (P-1)*\text{Max}[i = 1, H]\ [T(i) + C_t(i)] + D(\text{SP}). \qquad (4)$$

The delay for a RFNM is

$$D(\text{RFNM}) = \sum_{i=1}^{H} [L(i) + T_r(i) + C(i)]. \qquad (5)$$

Therefore, the minimum round trip delay for a mes-

Table 3

$P = 3$
$H = 3$
$C_t = C_r = 0.5$

HOP	1	2	3
L	0	1	3
T	2	3	2

Total delay = 21, as shown in fig. 2a.

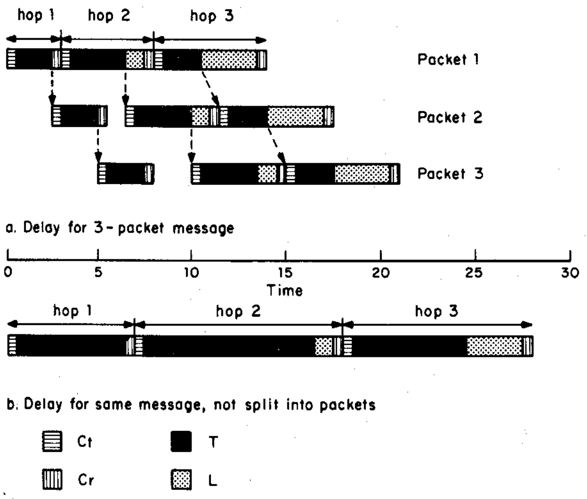

hop 1
hop 2
hop 3
Packet 1
Packet 2
Packet 3
a. Delay for 3-packet message
Time
b. Delay for same message, not split into packets
Ct
T
Cr
L

Fig. 2. An example of delays for a 3-packet message.

sage of P packets over H hops is

$$D(\text{MRT}) = \sum_{i=1}^{H} [2*L(i) + 2*C(i) + T_p(i) + T_r(i)] + (P-1)*\text{Max}[i=1,H]\,[T_p(i) + C_t(i)]. \tag{6}$$

If the values of L, C, C_t, and T are the same for each hop, then we have a simplified minimum round trip delay

$$D(\text{MRTS}) = \underbrace{H*(L + T_p + C)}_{\text{first packet}} + \underbrace{(P-1)*(T_p + C_t)}_{\text{subsequent packets}} + \underbrace{H*(L + T_r + C)}_{\text{RFNM}}. \tag{7}$$

Some curves are given in fig. 3 which illustrate the minimum round trip delay through a network for a range of message lengths and path lengths, for two sets of line speeds and lengths over paths of 1 to 6 hops. These graphs are from [24].

The rationale for packetizing messages. As a slight digression, we now consider the rationale for breaking up messages into packets in order to reduce delay. It will be shown that this rationale is similar to that for any pipelining technique, and the variables of interest will be identified. We will carry through this discussion for the simplified case of identical values of L, C, C_t, and T at each hop. First we rewrite eq. (7) as

$$D(\text{MRTS}) = (P-1)*(T_p + C_t) + H*[2*(L + C) + T_p + T_r]. \tag{8}$$

Note that the only one of the variables L, C, C_t, and

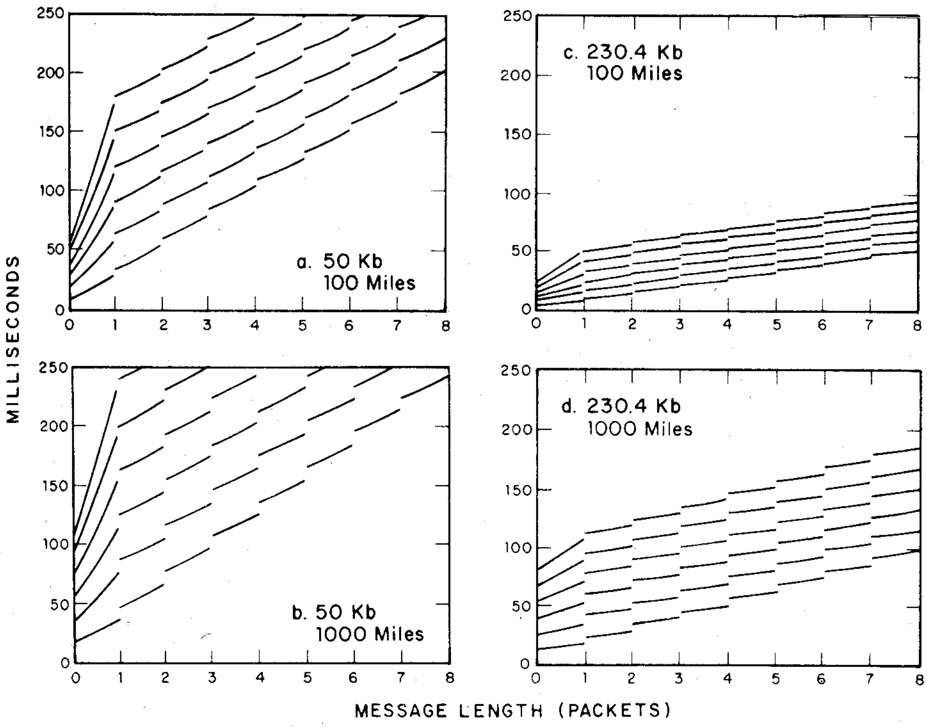


Fig. 3. Minimum round trip delay vs. message length.

T which is a function of packet length is T. Let us suppose that instead of a message of P packets, each of which has a transmission delay of T, the message is sent as a whole, with a transmission delay of $P*T$. The total delay can be calculated as above, considering this new entity as a long single-packet message. Using the equation for D(MRTS), we get an equation for minimum round trip delay, simplified, not packetized

$$D(\mathrm{MRTSNP}) = H*(L + P*T_p + C) + H*(L + T_r + C). \quad (9)$$

We can rewrite this as

$$D(\mathrm{MRTSNP}) = (P - 1)*H*T_p + H*[2*(L + C) + T_p + T_r]. \quad (10)$$

Subtracting eq. (8) from eq. (10) we get a difference in delay

$$D(\mathrm{MRTDIF}) = (P - 1)*[(H - 1)*T_p - C_t]. \quad (11)$$

We make the following observations:

1. Clearly, if $P = 1$, D(MRTDIF) = 0. There is no difference between the two techniques for single-packet messages.

2. Assuming $P > 1$, if $H = 1$, $D(\mathrm{MRTDIF}) = -(P - 1)*C_t < 0$. That is, if $H = 1$ and $P > 1$, it involves more delay to break a message into packets because of the added processor overhead (generally small).

3. Assuming $T_p > C_t$, that the processor time is small, if $P > 1$ and $H > 1$, then $D(\mathrm{MRTDIF}) > 0$. This is the usual case, and constitutes the basic reason to break messages into packets.

We can return to the numerical example used earlier with the equivalent conditions for a single packet message as long as the 3 packet message considered above (see table 4):

TABLE 4

$P = 1$
$H = 3$
$C_t = C_r = 0.5$

HOP	1	2	3	total
C	1	1	1	3
L	0	1	3	4
T	6	9	6	21
total	7	11	10	28

Total delay = 28, as opposed to 21 for the packetized case (see fig. 2b).

Queueing delay. We have examined the minimum round trip delay as a function of message length, network path length, and the packetizing strategy. It is appropriate at this point to analyze the effects of additional delays which may be present. To perform the minimum delay analysis, we made 3 assumptions, each of which should be re-examined at this point:

1. $C_q = 0$. The packet arriving may have to wait on an input queue before it is serviced by the processor. This time is generally quite small, but it is a random variable which may take on large values if there are time-consuming high-priority tasks in the system.

2. $D_q = 0$. This is the major assumption which must be modified in the analysis of actual delays in a network. A great deal of theoretical work has been done in studying queueing delay, particularly by Kleinrock [20,21], who has used both analysis and simulation in this regard. For the simplified discussion at hand, it is sufficient to note that each packet on the queue adds an additional $T + C_t$ to the delay of all packets behind it on the queue.

3. $D_r = 0$. The accuracy of this last assumption varies widely with the type of network and line being considered. For some wideband circuits, particularly satellite channels, the error rates are very low and few retransmissions may be necessary for reasons of errors. However, satellite links may be used in a broadcast competition mode so that some packets are lost in collision with others. Finally, there is always some chance that the adjacent node will refuse to accept a packet for lack of processing resources.

Low delay for interactive traffic. Perhaps the most important consideration about delay in a network is this: some traffic consists of interactive, high-priority messages and this traffic must be delivered to its destination as rapidly as possible. This is in contrast to bulk transfer traffic which is not so delay-critical. The most obvious case of such interactive traffic is most man–computer dialogue, which consists of rather short messages between the computer and a man at a terminal. Here there is a definite threshold for delay. Below this threshold, delay is acceptable, and above it delay is unacceptable. There is no added benefit if the delay is considerably below the threshold, and it is likely that once the delay is much above the threshold, almost any value of delay is equally unacceptable.

Another way of looking at the bimodal nature of network traffic is to consider that much of the delay for an interactive message is in the network itself. That is, it is generated and quickly sent into the network. When it is delivered, it will be processed quickly at the destination Host. Bulk transfers on the other hand may experience lengthy delays outside the network due to buffering considerations and the very size of the data (secondary storage or tapes or cards may be involved in the data transfer, greatly increasing delay).

2.2.2. Throughput

In this section we consider what throughput performance characteristics are possible in a given network. Some of this material was presented in [24] and [25]. The topics discussed below include an analysis of nodal processor bandwidth, an examination of circuit overhead, a quantification of the buffering in packets required for a network line and the buffering in messages required for a network path, the throughput requirements of bulk transfer traffic, and the tradeoff between delay and throughput.

The effective bandwidth of the node processor. We will first examine the effective processing capability of the node computer in a network with variable message length. Some numerical examples will be given to support the intuitive notion that the processor is most effective for long messages, given some very general assumptions about the packet processing involved.

We begin by defining some new quantities of interest in studying throughput in networks. The quantities which we will take as fundamental variables are as follows:

1. B_d = the number of Host data bits in a packet.

2. B_s = the number of software overhead bits per packet. These bits include header information such as address, and identifying information such as message number.

3. B_h = the number of hardware overhead bits per packet. These are typically framing bits for the circuit, and error detection bits such as checksums or redundant information.

4. P = the number of packets per message, as above.

5. B_{tot} = the total number of Host data bits per message.

$B_{tot} = P*B_d$ – unused bits in the last packet.

Now we will examine how long it takes the node processor to store and forward a message. As noted above, we ignore the processing at the source node and at the destination node. The time required to process a store-and-forward message is a function of the following parameters:

6. C = the packet processing time, as above. We assume that C is independent of packet length or type. To the extent that this is not true, the length-dependent component of C can be accounted for in item 8 below.

7. BWP_0 = the fraction of the bandwidth of the processor taken by overhead. Due to certain necessary periodic processes within the node, notably the routing computation, effective processor bandwidth is reduced.

8. BWIO = the I/O rate of the node in bits/sec. We assume here that BWIO is a linear function of the number of bits in the message. In most I/O architectures, it is probably a function of the number of computer words in the message, which is identical apart from unused bits in the last word. We are also assuming that the I/O transfer steals cycles from the processor, reducing its effective bandwidth.

9. I = the I/O transfer time in seconds. We will denote the I/O time for a packet, a message, and a RFNM as I_p, I_m, and I_r respectively:

$$I_p = 2*(B_d + B_s)/\mathrm{BWIO},$$

$$I_m = 2*(B_{tot} + P*B_s)/\mathrm{BWIO},$$

$$I_r = 2*B_s/\mathrm{BWIO}.$$

10. M_T = the total time taken to process a message.

$$M_T = C*(P+1)*(1+\mathrm{BWP}_0) + I_m + I_r.$$

11. BWP_d = the maximum data bandwidth that the node can support.

$$\mathrm{BWP}_d = B_{tot}/M_T.$$

This is the number of Host data bits per second that the node can process.

12. BWP_ℓ = the maximum line bandwidth that the node can support.

$$\mathrm{BWP}_\ell = [B_{tot} + (P+1)*(B_s+B_h)]/M_T.$$

This represents a processing capability limit on the number and speed of the circuits that can be connected to a node. The difference between the two quantities, $\mathrm{BWP}_\ell - \mathrm{BWP}_d$, is a measure of the line overhead at a given message length.

At this point, a numerical example may be illustrative. As presented in [24], in the ARPA Network, the

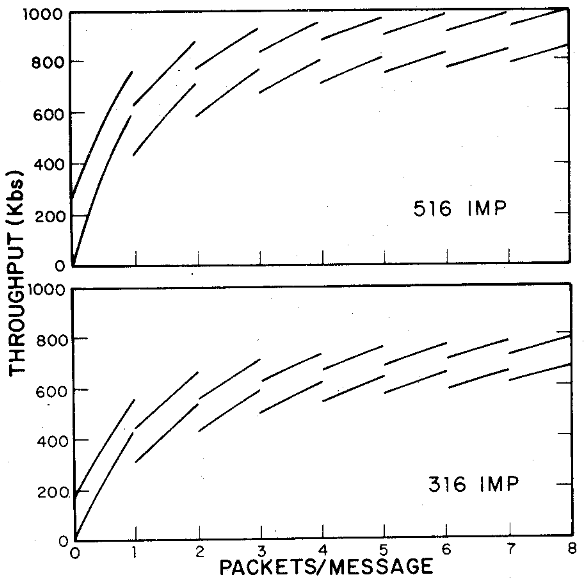


Fig. 4. IMP processor bandwidth vs. message length.

processor bandwidth of the IMP, both BWP_d and BWP_ℓ, can be plotted as functions of the message length, and some results are given in fig. 4. (The discontinuities indicated in the curves are the result of packetizing messages.)

The effective bandwidth of the network circuits. In this section we will consider some of the factors acting as overhead to reduce the effective circuit bandwidth for network lines. That is, we wish to catalog all the kinds of transmission that take place on the network lines that are not actual Host data bits, and from this accounting determine which factors are the dominant ones. There are three basic kinds of line overhead:

1. Line overhead in bits/packet. We have detailed two components of this overhead earlier, B_s and B_h.

2. Line overhead in bits/message. In the terminology we have been using, the RFNM is overhead on a message basis, and under our assumption that it is a minimum length packet, it contributes a number of bits equal to $B_s + B_h$.

3. Line overhead in bits/second of network system traffic. Some examples of this kind of traffic are given, along with the approximate order of magnitude for the traffic rate in the ARPA Network, in table 5.

We will make the assumption that routing messages are the primary source of traffic other than messages between Hosts. For this reason, we will

Table 5

	Traffic rate in ARPA Network (bits/sec)
routing	1000
line alive/dead	100
NCC status reports	10
core reloads and dumps	1–10

define the variables

B_r = the number of bits in a routing message,

F_r = the frequency of routing messages [1/sec],

$BWC_r = B_r * F_r =$

the bandwidth of the circuit used for routing,

and ignore the other components of periodic overhead.

We can now total the overhead from all considerations, by converting to a common dimension, bits/sec. In order to do this, we must introduce another variable,

F_p = the number of packets per second.

We then can express the total bandwidth of the circuit given to overhead as

$$BWC_0 = \underbrace{B_r * F_r}_{\text{routing}} + \underbrace{(B_s + B_h) * F_p}_{\text{packets}} + \underbrace{(B_s + B_h) * F_p / P}_{\text{RFNMs}}.$$

We can rewrite this as

$$BWC_0 = BWC_r + (B_s + B_h) * F_p * (P + 1)/P.$$

In comparison with this overhead rate is the actual data rate on the circuit,

$$BWC_d = B_d * F_p.$$

We can evaluate the fractional overhead percentage as

$$BWC_0/BWC_d = (BWC_r)/(B_d * F_p) + (B_s + B_h) * (P + 1)/(B_d * P).$$

Another quantity of interest is the maximum data rate that can be attained for given values of the systems parameters. Given a circuit with bandwith BWC, the maximum data rate occurs when F_p is at a maximum

$$F_p(\max) = (BWC - BWC_0)/B_d.$$

Substituting the expression for BWC_0 above, we get

$$F_p(\max) = (BWC - B_r*F_r)/$$
$$[B_d + (B_s + B_h)*(P+1)/P] \ ,$$

The numerator indicates that the routing message bandwidth comes off the top, leaving a reduced effective bandwidth, which is then used for both data bits and packet and message overhead bits. The graphs in fig. 5 show the behavior of some of these variables; the variable plotted is $F_p(\max)*B_d$, that is $BWC - BWC_0$, which is the maximum data rate for a given message size.

The packet buffering required for network circuits. We now turn to an examination of the number of packet buffers required to keep a communications circuit fully loaded. This number is a function not only of line bandwidth and distance but also of packet length, nodal delays, and acknowledgment strategy. We will assume that the node buffers each packet that it transmits until it receives an acknowledgment, meanwhile transmitting other packets to utilize the circuit efficiently. If it does not receive the acknowledgment in the expected time, it retransmits the packet. We also assume that packet buffers are a fixed size large enough to hold the largest packet. The expected time for an acknowledgment to return is the sum of:

1. T_p = the transmission time for the packet, a function of line bandwidth.
2. L = speed-of-light delay for the first bit of the packet to arrive at the other node, a function of line length.
3. $C = C_r + C_t$ = the processing delay in the other IMP, to receive the packet and return the acknowlegment.
4. D_q = queueing delay for the returning acknowledgment, the time it waits for any other transmissions ahead of it.
5. T_a = the transmission time for the acknowledgment.
6. L = speed-of-light delay for the first bit of the acknowledgment to arrive at the first node.
7. C_r = the processing delay for the acknowledgment.

Our first simplifying assumption is that the processing times are small compared to the other delays and can therefore be ignored:

$$C_r = C_t = 0.$$

We can then state that the minimum number of packet buffers needed to keep a circuit fully loaded is

$$BF_p = (T_p + L + D_q + T_a + L)/T_p.$$

This can be rewritten in somewhat more meaningful form as

$$BF_p = 1 + 2*L/T_p + (D_g + T_a)/T_p$$

This expression indicates that one buffer is always necessary, to account for the packet transmission time itself. More buffers may be required if the circuit is long compared to the packet transmission time, or if the acknowledgment transmission takes a long time compared to the packet. Stated differently, the number of buffers needed to keep a line full is

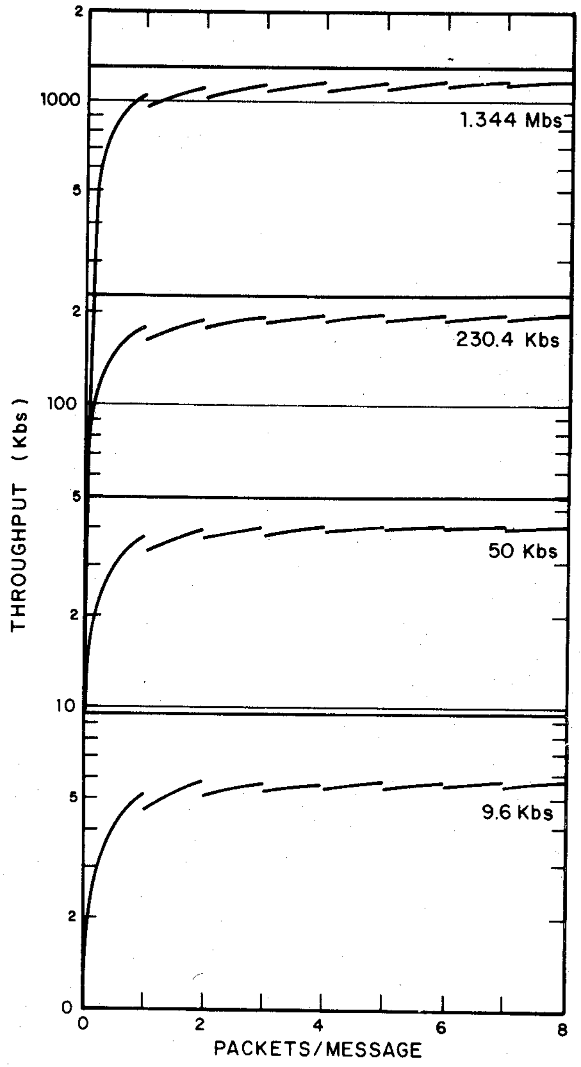


Fig. 5. Effective circuit bandwidth vs. message length.

proportional to the length of the line and its speed and inversely proportional to the packet size, with the addition of a constant term.

In order to proceed further with the analysis, we need to introduce two new terms:

T_s = the transmission time of the shortest allowable packet

T_ℓ = the transmission time of the longest allowable packet

We also need to postulate a traffic mix of long and short packets, with x/y the ratio of short packets to long packets in the channel. Now we can define D_q and T_a in terms of T_s and T_ℓ. We make a worst-case assumption for D_q: $D_q = T_\ell$, the acknowledgment has highest priority (equivalently, it piggybacks on all packets), but it must wait for the transmission of a maximum-length packet which has just begun.

The assumption for T_a is rather arbitrary:

$T_a = (T_s + T_\ell)/2$, the acknowledgment piggybacks on an "average" length packet.

We now state the result for the number of packet buffers required given the above set of assumptions:

$$\mathrm{BF}_p = 1 + [2*L + T_\ell + (T_s + T_\ell)/2]/T_p.$$

Using the ARPA Network values of T_s and T_ℓ given above in table 2, and choosing a variety of line lengths and traffic mixes (shown as the ratio of short packets, S, to long packets, L), we can present some numerical results as a family of curves shown in fig. 6. Note that the knee of the curves occurs at progressively shorter distances with increasing line speeds. In fact, if we define the knee to occur when the linear term is equal to half of the constant term, then the knee occurs when

$$L = (T_s + 2*T_p + 3*T_\ell)/4,$$

or for a line length $225*10^6$/BWC miles. 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.

The message buffering required for network paths. This section takes up a topic which closely parallels that of the last section. Here we will examine the number of messages needed to obtain full bandwidth over a network path of many lines. That is, we will compute how many messages must be in flight between two nodes in order to keep all the intermediate lines fully loaded. Actually, the best one can do

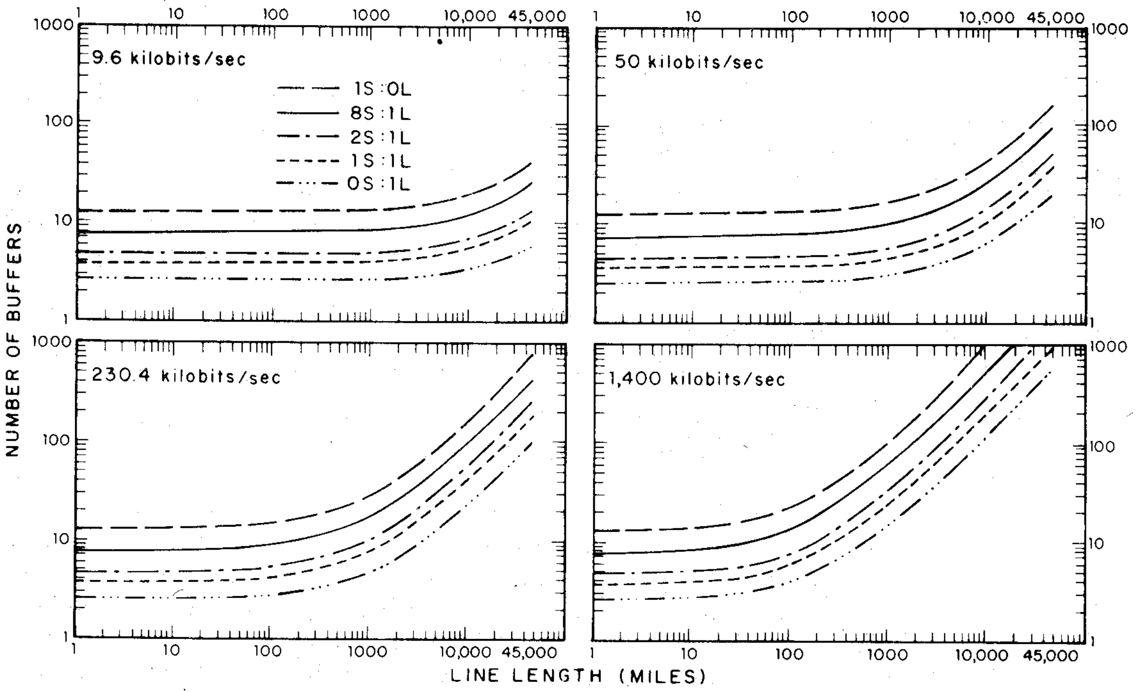


Fig. 6. Packet buffering for full line utilization.

is to keep all the lines in the path of the lowest circuit bandwidth fully loaded. It turns out that this analysis is quite simple given all the definitiions of the preceding sections. The number of message buffers needed is computed by taking the round trip delay for a message and dividing it by the time taken to transmit a single message. That is,

$$\mathrm{BF_m} = D(\mathrm{MRT})/[P*(T_p + C_t)].$$

Using eq. (6) in section 2.2.1, we can obtain a more detailed expression for the simple case of equal delay at each hop:

$$\mathrm{BF_m} = (P-1)/P + H*[2*(L+C) + T_p + T_r]/[P*(T_p + C_t)].$$

It is clear from this expression, and on intuitive grounds, that $\mathrm{BF_m}$ is a minimum for maximum length messages, that is for large *P*. The curves in fig. 7 show the dependence of $\mathrm{BF_m}$ on line characteristics and on the length of the network path, for $P = 8$ and ARPA Network values of the parameters. For each of four line speeds, the buffering requirements are plotted for network paths made up of a number of land lines (the length of the lines is given with each curve). Also shown are the requirements for the same network paths with the addition of one satellite link running at the same bandwidth as the land lines.

The consequences of the above discussion are several. For the communications subnetwork, it means that the nodes must be able to do bookkeeping

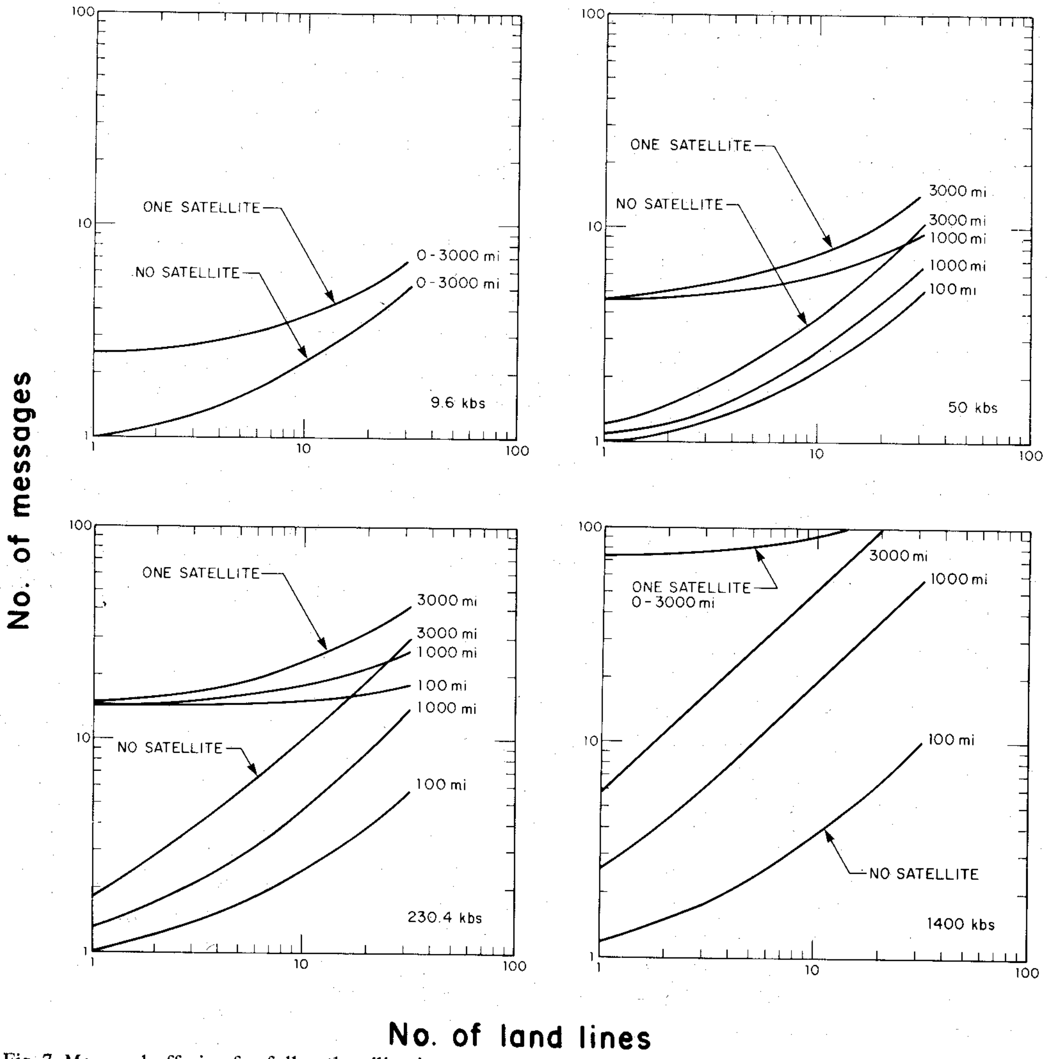


Fig. 7. Message buffering for full path utilization.

on several messages in flight between two nodes. Further, the network must buffer these messages in the memory of the source or destination node for the duration of their flight, in addition to the packet buffering that takes place instantaneously at the intermediate nodes along the path. This has some important ramifications for the design of the software for the node computer. A second set of issues is the effect of message buffering on Host computers. It is clear that the communication protocols that the Hosts use must also be engineered to the parameters of the network, if they are to obtain full throughout levels.

High throughput for long data transfers. The several topics examined in this section all point to a single conclusion: the larger the packets in a message, and the larger the messages in a data transfer, the higher the level of throughput that is potentially attainable. For reasons of processor overhead, circuit overhead, and buffering considerations within the nodes, it is always better to have long packets and messages if high data rates are desired.

The tradeoff between low delay and high throughput. It is clear that the two goals of low delay and high throughput are often in conflict because the node has limited resources with which to service its Hosts and lines. It is difficult to guarantee both low delay and high throughput to several competing sources. The approach taken in the ARPA Network is as follows: the IMP program has been designed to perform well under bimodal traffic conditions. It provides quick delivery for short interactive messages and high throughput rates for long files of data. This optimization of the program for a specific model of traffic behavior occurs at many levels, and is essential for balanced performance characteristics. (Other conflicts exist, such as the conflict between high throughput and congestion/deadlock control mechanisms.)

2.2.3. Cost

In this section we present some of the primary issues regarding the cost of computer networks. There are two kinds of costs considered here: the cost of the actual network components, and the cost of the use of the network. The first cost is a measure of the expense of connecting some component to the network, and the second is a measure of the expense of utilizing the resources of that component. We are concerned here with outlining the effects of these costs on the balance among the various system parameters of the network. The more specific effects routing algorithms have on the cost of networks are discussed in section 2.3, on routing.

Table 5

Some representative line costs

Bandwidth (Kb/sec)	Termination cost ($/month)	Line cost ($/month/mile)
9.6	650	0.40
19.2	850	2.50
50	850	5.00
230.4	1350	30.00

Low cost for network connectivity. The first point to take up in consideration of the cost of networks is the cost of connectivity. The basic variables are circuit costs and node costs. Some typical values for these costs in an actual network may be illustrative; the values for the ARPA Network are shown in tables 5 and 6.

Several implications follow from these figures:

1. The layout problem for networks is an important one, since reductions in the total number of circuit miles in a network represents substantial dollar savings.

2. Low cost comes at the expense of both low delay and high throughput if lower line speeds are chosen.

3. Low cost also comes at the expense of low delay and high throughput if fewer lines are used, since more traffic is forced to use each line.

4. As a consequence of these points, some network designers, notably the Network Analysis Corporation [29,30], attempt to find a relatively low cost network layout and then test it by simulation to determine if average message delay is below some threshold and the throughput obtained (when all nodes

Table 6

Some representative node costs

Machine Type	Cost (K$)	Configuration (Kb total)
516 IMP	100	up to 4 Hosts (400) up to 5 lines (800) up to 7 devices total
316 IMP	50	up to 4 Hosts (300) up to 5 lines (600) up to 7 devices total
316 TIP	100	up to 2 Hosts (300) up to 3 lines (600) up to 64 terminals (100)

send to all other nodes) is above some threshold.

Low cost for network use. A second aspect of the cost of networks is the cost of the use of the network, which may be counted in one of several ways:

1. \$/bit, \$/character, \$/packet, or \$/message costs for the shipping of data through the network.
2. These charges may be rated per mile or per hop or may be distance-independent.
3. There may be different grades of service at different costs.

These costs are the translation into dollars of the utilization rates for various network resources. These resources can be catalogued as follows:

1. line bandwidth
2. node processor bandwidth
3. node storage

2.2.4. Reliability

In this section we present some of the primary issues regarding the reliability of computer networks. In parallel with the discussion of network cost above, there are two basic topics examined here: the reliability of the network connections themselves, and the reliability of the network services. In the first instance, we are interested in how reliable the network components are. In the second case, the subject is the reliability of the use of the network facilities for data transmission. Again we are focussing in this section on the broad design issues which affect network performance as a whole. In section 2.3, on routing, we examine closely the effects that routing algorithms can have on network reliability.

High reliability of network connectivity. One can attempt to minimize the probability that a line or node will be inoperative, and thus to reduce the probability that a node cannot communicate with the rest of the network. In practice, this is a matter of maintaining a high mean time between failures and a low mean time to repair. One can also measure reliability in terms of the number of nodes and/or lines necessary to disconnect the network, taking into consideration the probability of the various events. Alternatively, one can consider the size of the components of the disconnected network, or the fraction of node pairs not connected by any network path.

A different level of solution to the problem of network reliability is redundant network design, primarily in the layout of the circuits connecting the nodes. In this way, a network can be constructed which is much more reliable as a whole than any one component. In the ARPA Network, a design constraint has been that an IMP must be connected to the network by at least two circuits, so that the probability that an IMP cannot communicate with the network is very small. Of course, this principle can be applied to other components in the network as well. The node computers can be backed up with alternate computers, or may be equipped with redundant interfaces and processors. The Host computers also may wish to be connected to the network at more than one point by means of separate communications facilities.

High reliability of network use. The reliability of the message processing can be measured in terms of the percentage of messages delivered, or the detected error rate, or the undetected error rate. Measures to improve the reliability of message processing range from error detecting and correcting hardware to redundancy in software to backup message storage. The costs of these approaches are multiple: they add to the complexity of the system, and may degrade its performance, in addition to representing a dollars cost. In general, the communications subnetwork becomes more costly as these measures of reliability are improved. At some point, it becomes appropriate to pass the cost of these improvements on to the user. However, a minimum level of reliability is necessary for the operation of the communications subnetwork. Guarantees concerning a grade of service better than this minimum level might reasonably cost more.

The tradeoff between low cost and high reliability. It is clear that there is a natural tradeoff between low-cost networks and high-reliability networks. This tradeoff exists in building either sparse networks or highly-connected networks, and in providing special mechanisms to ensure the reliable transmission of data or choosing not to implement such safeguards. In short, the price for reliability must be paid somewhere, either in the actual cost of constructing and maintaining the network, or in the cost to the user of unreliable network service. It is difficult to generalize, but it may often be the case that a low-cost network without sufficient measures for reliability may prove more costly to use in the long run than a network with a higher cost for higher network reliability. Stated differently, it may be cheaper to build a network to be fault-tolerant, redundant, and error-detecting than to build these measures into each user process that communicates with the network.

2.3. Key design choices

We believe there are three major areas in which the key choices must be made in designing a packet-switching network. First, there is network equipment design, including the node computer, the network circuits, the Host-to-node connections, and overall connectivity. Second, there is store-and-forward subnetwork system design, primarily the routing algorithm and the node-to-node transmission procedures. Third, there is source-to-destination system design, which encompasses end-to-end transmission procedures and the division of responsibility between Hosts and nodes. These topics are covered in the following sections. (Note: there are strong interactions between the topics discussed in the second and third areas. The end-to-end traffic requirements of a specific user can only be met if the store-and-forward subnetwork has mechanisms which act in concert with the source-to-destination mechanisms to provide the required performance. Discussion of this interaction, an important consideration in packet-switching network design, is beyond the scope of this paper.)

3. Network equipment design

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

3.1. The node computer

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

3.1.1. Processor

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

A different aspect of the node computer which can also affect throughput is its responsiveness. Because circuits are synchronous devices, they require service with very tight time requirements. If the node does not notice that input has completed on a given circuit, and does not prepare for a new input within a given time, the next input arriving on that circuit will be lost. Likewise on output, the node must be responsive in order to keep the circuits fully loaded. This requirement suggests that some form of interrupt system [17] or high-speed polling device [18] is necessary to keep response latency low, and that the overhead of an operating system and task scheduler and dispatcher may be prohibitive. Finally, we note that the amount of time required by the node to process input and output is most critical in determining the minimum packet size, since it is with packets of this size that the highest packet arrival and departure rates (and thus processing requirements) can be observed. Of course, data buffering in the device interfaces can partially alleviate these problems.

3.1.2. Memory

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

The size of the memory, naturally, is another key parameter. It has been our experience [17,24] that the program and associated data structures take up the majority of storage in the node. The remainder of memory is devoted to buffering of two kinds: packet buffering between adjacent nodes, and message buffering between source and destination nodes. These requirements can be calculated quite simply in each case as the product of the maximum data rate to be supported times the round trip time (for a returning acknowledgment). In large networks it may be necessary to rely on sophisticated compression techniques to ensure that tables for the routing algorithm, the source-to-destination transmission procedures, and so on, do not require excessive storage. The node storage in the ARPA Network has been barely adequate.

3.1.3. I/O

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

In addition, the design of the line termination hardware can have a significant impact on the processor loading and responsiveness. For example, in the ARPA Network, the line termination interfaces include sufficient hardware to calculate the 24-bit CRC used between IMPs, thus relieving the processor of the burden of the calculation. These interfaces also operate via direct memory access channels; the processor is interrupt-driven and instructs the interface only once per packet.

3.1.4. General architecture

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

Another aspect of node computer architecture is its reliability, particularly for large systems with many lines and Hosts. A failure of such a system has a large impact on network performance. We have studied these issues of performance, cost, and reliability of node computers in a packet-switching network, and have developed, under ARPA sponsorship, a new approach to this problem. Our computer, called the Pluribus, is a multiprocessor made up of minicomputers with modular processors, memory, and I/O components, and a distributed bus architecture to interconnect them [18]. Because of its specially designed hardware and software features [32], it promises to be a highly reliable system. We point out that many of these issues of performance, cost, and reliability could become crtically important in very large networks serving thousands of Hosts and terminals.

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

3.2. The network circuits

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

3.2.1. Bandwidth

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

3.2.2. Delay

The major effect of circuits with appreciable delay in a system requiring packet acknowledgement is that they require more buffering in the nodes to keep them fully loaded. That is, the node must maintain more packets in flight at once over a circuit with longer delay. This effect may be so large (a circuit using a satellite has a delay of a quarter of a second) as to require significantly more memory in the nodes [24]. This memory is needed at the nodes connected to the circuit to permit sufficient packet buffering for node-to-node transmission using the circuit. The subtle point is that additional buffering is also required at all nodes in the network that may need to maintain high source-to-destination rates over network paths which include this circuit. If they are to provide maximum throughput, they need sufficient message buffering to keep the entire network path fully loaded.

3.2.3. Reliability

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

An example of an acceptable statement of error performance would be as follows: The circuit operates in two modes. Mode 1: no continuous sequence of packet errors longer than two seconds, with the average packet error rate less than one in a thousand. Mode 2: a continuous sequence of errors longer than two seconds with the following frequency distribution:

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

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

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

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

3.3. The Host-to-node connections

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

3.3.1. Bandwidth

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

3.3.2. Reliability

The reliability of the Host connection is an important aspect of the network design; several points are worth noting. First, the connection should have a packet error rate which is at least as low as the network circuits. This can be accomplished by a highly reliable direct connection locally or by error-detection and retransmission. The use of error control procedures implies that the Host-node transmission procedures resemble the node-node transmission procedures which are discussed in a later section. Second, if the Host application requires extremely high reliability, a Host-to-Host data checksum and message sequence check are both useful for detecting infrequent network failures. Third, if the Host requires uninterrupted network service, and the Host is reliable enough itself to justify such service, multiple connections of the Host to various nodes can improve the availability of the network. This option complicates matters for the source-to-destination transmission procedures in the nodes (e.g., sequencing) since there may be more than one possible destination node serving the Host.

3.4. Overall connectivity

The subject of network topology is a complex one [15], and we limit ourselves here to a few general observations. In practice, it seems that the connectivity of the nodes in the network should be relatively uniform. It is obvious that nodes with only a single line are to be avoided for reliability considerations but nodes with many circuits also present a reliability problem since they remove so much network connectivity when they are down. We also feel that the direction for future evolution of network geometries will be towards a "central office" kind of layout with relatively fewer nodes and with a high fan-in of nearby Hosts and terminals. This tendency will become more pronounced as higher reliability in the node computer becomes possible, even for large systems. One reason that we favor this approach is that a large node computer presents an increased opportunity for shared use of the node resources (processor and memory) among many different devices leading to a much more efficient and cost-effective implementation. This trend will mean that in the future, even more than now, a key cost of network topology will be the ultimate data connection to the user (Host or terminal), who may be far from the central office. Concentrators and multiplexors have been the traditional solution; in packet-switching networks, a small node computer should fill this function. In conclusion, we see flexibility and extensibility as two key requirements for the node computer. These factors together with increasing performance and fan-in requirements imply a very high reliability standard as well.

4. Store-and-forward subnetwork system design

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

4.1. The routing algorithm

The fundamental step in designing a routing algorithm is the choice of the control regime to be used in the operation of the algorithm. Non-adaptive algorithms make no real attempt to adjust to changing network conditions; no routing information is exchanged by the nodes, and no observations or measurements are made at individual nodes. Centralized adaptive algorithms utilize a central authority which dictates the routing decisions to the individual nodes in response to network changes. Isolated adaptive algorithms operate independently with each node making exclusive use of local data to adapt to changing conditions. Distributed adaptive algorithms utilize internode cooperation and the exchange of information to arrive at routing decisions. (For a much more detailed discussion, see [26].)

4.1.1. Non-adaptive algorithms

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

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

At their best, in the absence of network component failure, fixed routing algorithms are inefficient. While the routing tables can be fixed to be optimal for some traffic flow, fixed routing is inevitably inefficient to the extent that network traffic flows vary from the optimal traffic flow. Unreliability and inefficiency are also characteristic of two alternative techniques to fixed routing which fall under the heading of non-adaptive algorithms: fixed routing with fixed alternate routes and random routing [26].

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

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

4.1.2. Centralized adaptive algorithms

In a centralized adaptive algorithm, the nodes send the information needed to make a routing decision to a Routing Control Center (RCC) which dictates its decision back to the nodes for actual use. The advantages claimed for a centralized algorithm are: a) the routing computation is simpler to understand than a non-centralized algorithm, and the computation itself can follow one of several well known algorithms, e.g. [14]; b) the nodes are relieved of the burden and overhead of the routing computation; c) more nearly optimal routing is possible because of the sophistication that is possible in a centralized algorithm; and d) routing "loops" (a possible temporary property of distributed algorithms) can be avoided.

Unfortunately, the processor bandwidth utilization at the center is likely to be very heavy. The classical algorithms that a centralized approach might use generally run in time proportional to N to the third (where N is the number of nodes in the network), while their distributed counterparts can run (through parallel execution) in time proportional to N to the second. While it may be a saving to remove computation from the nodes, it may not be possible to perform a cubic computation on a large network in real time on a single computer, no matter how powerful [26].

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

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

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

A related difficulty with centralized algorithms lies in the fact that when a node or line fails in the network, the failed component may have been on the previously best path between the RCC and the nodes trying to report the failure. In this case, just at the time the RCC needs routes over which to receive and transmit routing information, no routes are available; the availability of new routes requires the very change the RCC is unsuccessfully attempting to distribute. Solutions which have been proposed to solve this "deadlock" are slow, complicated, awkward, and frequently rely on the temporary use of distributed algorithms [16].

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

4.1.3. Isolated adaptive algorithms

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

There are two basic approaches to be employed, separately or in tandem, to the process of learning and forgetting. We call these approaches positive feedback and negative feedback. One way to implement positive feedback was suggested by Baran as part of his hot-potato routing doctrine [2]. Each node increments the handover number in a packet as it forwards the packet. Then the handover number is used in a "backwards learning" technique to estimate the transit time from the current node to the source of the packet. Clearly, this scheme has drawbacks because it lacks any direct way of adapting to changes. If no packets from a given source are routed through a node by the rest of the network, the node has no information about which route to choose in sending a message to that source. In general, as part of a positive feedback loop, the routing algorithm must periodically try routes other than the current best ones, since it has no direct way of knowing if better routes exist. Thus, there must always be some level of traffic traveling on any route that the nodes are to learn about, since it is only by feedback from traffic that they can learn.

The other half of an adaptive isolated algorithm is the negative feedback cycle. One technique to use here is to penalize the choice of a given path when a packet is detected to have returned over the same path without being delivered to its destination. The relation of this technique to the exploratory nature of positive feedback is evident.

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

4.1.4. Distributed adaptive algorithms

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

Consider the following example of a distributed adaptive algorithm. Each node estimates the "distance" it expects a packet to have to traverse to reach each possible destination over each of its output lines.

Periodically, it selects the minimum distance estimate for each destination and passes these estimates to its immediate neighbors. Each node then constructs its own routing table by combining its neighbors' estimates with its own estimates of distance to each neighbor. For each destination, the table is then made to specify that selected output line for which the sum of the estimated distance to the neighbor plus the neighbor's distance estimate to the destination is smallest.

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

The first point is that distributed algorithms are slow in adapting to some kinds of change; in particular, the algorithm reacts quickly to good news, and slowly to bad news. If the number of hops to a given node decreases, the nodes soon all agree on the new, lower, number. If the hop count increases, the nodes will not believe the reports of higher counts while they still have neighbors with the old, lower values. This is demonstrated in [26]. Another point is that there is no way for a node to know ahead of time what the next-best or fall-back path will be in the event of a failure, or indeed if one exists. In fact, there must be some finite time, the network response time, between the occurrence of a change in the network and the adaptation of the routing algorithm to that change. This time depends on the size and shape of the network.

We have come to conclude that the routing algorithm should continue to use the best route to a given destination, both for updating and forwarding, for some time period after it gets worse. That is, the algorithm should report to the adjacent nodes the current value of the previous best route and use it for routing packets for a given time interval. We call this "hold down" [26]. One way to look at this is to distinguish between changes in the network topology and traffic that necessitate changing the choice of the best route, and those changes which merely affect the characteristics of the route, like hop count, delay, and throughput. In the case when the identity of the path remains the same, the mechanism of hold down provides an instantaneous adaptation to the changes in the characteristics of the path; certainly, this is optimal. When the identity of the path must change, the time to adapt is equal to the absolute minimum of one network response time, while the other nodes have a chance to react to the worsening of the best path and to decide on the next best path. (Please note that this is a very simplified description of hold down. A more complete description states in detail when hold down should be invoked and for what duration. Such a description may be found in [26], and more is being learned [31].)

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

4.1.5. Routing processing goals

In this section we outline some of the desirable characteristics for the processing of routing information. There are six divergent goals for the routing algorithm in its task of accepting the input data about the network and generating the required output.

Simplicity. Simplicity is the goal we list first, because it assumes increasing importance as further requirements are placed on the routing algorithm and the trend to complexity grows. There are two distinct kinds of simplicity that are of great value here, and in any computer algorithm. First, it is almost essential that the routing program running in a network be simple enough for a man to understand what is happening in a given situation. This is desirable not from the point of view of keeping the man in control, but merely to permit him to follow the operation of the algorithm and find problems and suggest improvements in its performance. This may sound trivial, but in a very large network even the most basic measurements and the most elementary observations are difficult to undertake. Thus it is useful if the routing algorithm itself does not present further complexities to the man trying to interpret its behavior. In practice, it should be possible to understand or predict the behavior of the routing algorithm without difficulty. For example, order-dependent or non-deterministic rules may be too obscure to follow, particularly in a network environment.

The second kind of simplicity that is desirable is simplicity in the design and structure of the algorithm, so that it can be coded in a small, simple program. It is more likely that the program will be efficient and reliable if it can be expressed simply to the computer. It is always good software design to write simple programs, and this is especially true in a network environment, where the programs must run continuously in real time, and run on many computers.

Reliability. It is critical that the routing algorithm be reliable in the face of node and line failures. Such failures must be expected, and, indeed, when they occur the successful operation of the routing calculation is most essential. Therefore, the momentary or prolonged malfunction of any component in the network should not interfere with the steady, accurate process of calculating the best routes for traffic in the network. When parts of the network are isolated from each other, and when they are reunited, the changeover should be managed smoothly. If a node fails to receive one or two routing messages due to line errors or node problems, or receives extra messages, or erroneous data, the global reliability of the distributed computation should not be affected.

Steady state solution. A basic requirement of any routing algorithm is that, given a static set of inputs, it should arrive at a steady state solution which is accurate and stable. This is a very elementary goal, but one that should not be overlooked. The operation of the algorithm should not be so approximate that its outputs oscillate under static input conditions. For instance, random or heuristic algorithms are undesirable for this reason. Also, order-dependent algorithms and techniques with many special cases may not always arrive at the same solution to a given set of routing input data with slight changes in the specifications. Attention to this goal is particularly important in the early stages of designing an algorithm. Then, when test cases are being thought up, it is essential that the algorithm arrive at an accurate and stable solution for all the simple static tests that can be devised.

This means, for instance, that in thinking about algorithms that should work for a network with a connectivity that is assumed to change, one should test the algorithm on the following kinds of networks:

single node
loop networks
star networks
tree networks
union of a loop and a tree
union of two loops
series-parallel combinations

and so on. The algorithm should also be tested with various traffic requirements, such as:

no traffic
one-way traffic along a single path
two-way traffic along a single path
two traffic streams on separate paths
two traffic streams with a line in common

and so on. Solving the routing problem for a static situation is much easier than for a dynamically changing one. It is also much easier to verify that the routing actually performs as desired.

Adaptation to change. In a real network, of course, the determination of the output data outlined above is not a one-time calculation. As we have pointed out, the routing algorithm must run continuously, and its input data may change at any time. This leads us to the next requirement, that the algorithm be quickly adaptive to changes in network topology and traffic patterns. This point is of central importance, since routing for fixed inputs is a trivial problem by comparison. The computation to determine whether any paths exist between a pair of nodes must be sensitive to changes in the configuration of the network. The circuits and nodes in the network may fail, isolating some nodes from others, or there may be an alternate path remaining to connect them. That is, traffic should be routed around broken lines and nodes as long as any path exists to the desired destination. This may even require that a packet be returned over a path it has already traveled. For instance, consider a packet going the short way around a circle from its source to its destination when a line in the path ahead breaks. The packet should turn around and go back the other way around the circle. Similarly, circuits and nodes may be added to the network, and an adaptive routing algorithm has the advantage that these changes may happen smoothly and without any human intervention in the operation of the network. The algorithm must also be sensitive to changing traffic levels and patterns, since these will affect the computation of the paths of low delay and high capacity.

Adaptation to change–priority of routing. One important aspect of the requirement that a routing algorithm be adaptive to changing network conditions is that it implies a priority structure to the task processing in the node computer. Routing must always have higher priority than packet processing because it may be essential to change the routes being used to some new routes, especially at moments of high traffic load. If routing cannot be recomputed because of higher priority tasks, then the network performance will degrade significantly as congestion sets in due to out-of-date routing information. In practice, this means that:

1. The input of routing messages must always be possible.

a. Storage must be reserved for the messages (a reserve of the common free list is necessary since the node cannot know ahead of input time whether the next input will be a packet or a routing message).

b. The input process must be able to run often so that messages are not lost.

2. The output of routing messages must always be possible.

a. Storage must be reserved for the messages (this can be dedicated fixed table storage).

b. The output process must be able to run often enough to send routing messages as required, and they should have higher priority than any other transmission.

3. The routing update computation must always be possible.

a. Storage must be reserved for the routing update (this also can be dedicated tables).

b. The updating process must be able to run periodically, either on a clock interrupt or called by the input and output processes which have been guaranteed to run frequently enough.

Adaptation to change–speed. Given that the routing algorithm is adaptive to changes in the network, there are further desirable characteristics we can list. It should adapt as quickly as possible. Once the pattern of traffic has changed in the network, the old paths may become suboptimal, and perhaps even counter-productive, interfering with other useful traffic. The nodes should decide quickly and in a harmonious, cooperative manner how to use the resources of the network. When the routing algorithm must adapt, it should do so smoothly, without oscillation, and without creating undue conflicts in other parts of the network.

Global optimality. There are other goals in addition to the local choice of paths of low delay and high throughput. It is important that the routing algorithm meet certain global requirements as well. For instance, the algorithm should lead the nodes in the network to a global optimum in utilizing shared resources. It is not enough for an individual node to find a good path to another node, it must do so in the light of the needs of other nodes in the network. It is possible to think up routing algorithms which stabilize at several operating points in a given situation, and only one of them is the global best use of resources. A good routing algorithm routes high bandwidth traffic to achieve the maximum global flow. Suppose, as in fig. 8, it is desired to send 50 kilobits/second of traffic from node 1 to node 4 and at the same time it is desired to send 50 kilobits/second from node 6 to node 5. If the traffic from 6 to 5 as well as the traffic from 1 to 4 must pass over the link from 6 to 5, the global flow is 50 kilobits/second. If, however, either the traffic from 1 to 4 or the traffic from 6 to 5 were to go around the other way via the link from 2 to 3, the global flow would be 100 kilobits/second which is the maximum global flow under the desired traffic inputs. Of course, it is clearly best for the 1 to 4 traffic to go via the link from 2 to 3 since that also minimizes the global delay.

Fairness. This suggests that the algorithm should be fair to competition for shared resources. While fairness can sometimes be quantified, it is often a subjective judgment, but it is vital that the routing algorithm operating at one node does not prevent another node from gaining a share of network services. Consider fig. 9, in which it is desired to send 1 unit of

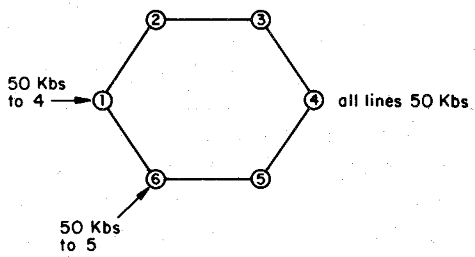


Fig. 8. Global optimum traffic flow.

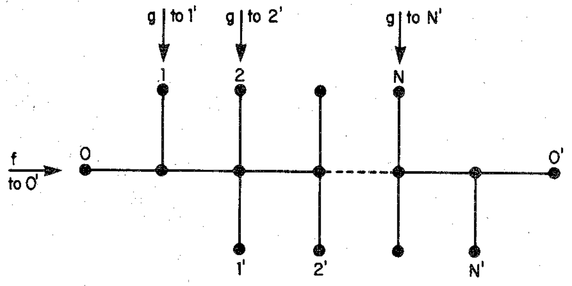


Fig. 9. Fairness.

1. How Routing Affects These Goals
2. The Appropriate Metric for Evaluation

Low Delay	High Throughput
ave. round trip delay	fraction of total bandwidth obtained
Low Cost	High Reliability
$/month membership	% time disconnected
$/bit or message	% messages undelivered

Fig. 10. Routing performance measures.

traffic from node 0 to node $0'$, from 1 to $1'$, from 2 to $2'$, and so on. Clearly the maximal global flow is attained under this desired loading by stopping all traffic from 0 to $0'$ since any traffic from 0 to $0'$ reduces the global flow. But this maximum flow assignment is unfair. Let f be the flow from 0 to $0'$ and g be the flow from i to i' for i equals 1 to N. Then $f+g \leqslant 1$.

$$\text{TF} = \text{Total flow} = f + N*g,$$

$$\text{Max TF } [f=0, g=1] = N,$$

$$\text{Min Tf } [f=1, g=0] = 1.$$

Fairness demands that 0 be able to get some traffic to $0'$, even if it decreases the global flow. It might be considered fair to give all nodes an equal share of the available bandwidth, or to allocate bandwidth to each node in proportion to demand or to available capacity. For instance, one assignment which is certainly fairer than $f=0, g=1$ is:

$$f = \tfrac{1}{2},\ g = \tfrac{1}{2};\ \text{TF} = \tfrac{1}{2}(N+1).$$

A different approach is:

$$f = 1/N,\ g = (N-1)/N;\ \text{TF} = N - 1 + (1/N)$$

which is quite close to the maximum flow for large N. Each definition of "fair" has some merits and drawbacks, but the important point is that the routing algorithm should be designed to adhere to some fairness doctrine, or else some traffic will be locked out.

4.1.6. Routing performance measures

In this section, we will take up the question of how to evaluate the performance of a routing algorithm. We will use the four basic factors underlying the performance of the network as a whole in considering the routing algorithm. The two central concerns are pictured in fig. 10.

Delay. As we indicated in the section above on network delays, the most important point about delay usually is that interactive traffic experience the minimum delay possible, although sometimes uniformity of delay is important. Apart from general considerations such as giving such traffic higher prioritly than bulk traffic, what effect can routing have on delay performance? One way to answer this question is to review the components of delay introduced above, and to note what action the routing algorithm can take with regard to each. Before giving this list, it should be noted that the actual values of the variables in question here may vary enormously from one network to another, and within a given network. Therefore, some of the considerations will certainly outweigh others.

1. Propagation delays. The routing algorithm may keep track of the speed-of-light delays in the network, which may vary significantly if there are some long lines, and certainly if there are satellite links. Although these delays are fixed and beyond the control of the algorithm, it can choose paths with the least propagation delay.

2. Transmission delays. The routing algorithm may also keep track of the bandwidth of the circuits in the network, to know the transmission delays that packets will experience. Again, these delays are not likely to be dynamic, but the routing computation can use fast lines and avoid slow lines where possible.

3. Nodal processing delays. Here delay is more likely to have a large dynamic range. Some nodes may have different traffic loads than others by several orders of magnitude, and the nodes themselves may have different capacities, so the input queueing delays for processing service may vary. If this component

can be significant, the routing algorithm should monitor the values for processing delays at the nodes.

4. Queueing delay. The case of queueing delays on circuits is similar. There will most certainly be some lines which are loaded more heavily than others, and the queueing delays are inversely proportional to line bandwidth, so long queues on slow lines lead to very long delays. In both of these cases, not only can the routing algorithm avoid long delays by choosing alternate routes, but it is also the major determinant of processing and queueing delays. That is, the routing algorithm may be structured to sense the buildup of these delays and change the routes being used accordingly, explicitly to reduce the load on individual nodes or lines.

5. Retransmission delays. This case is somewhat similar as well. The routing algorithm should adapt to any sizable number of retransmissions on any circuit, since they affect many of the other variables. By reducing the effective bandwidth of the circuit, retransmissions increase the processing and queueing delay for a given circuit is not strictly constant. This is especially true for satellite links used in broadcast mode.

Throughput. The next topic we will examine is that of network throughput as related to routing. We have stated that the important goal here is that large data tranfers, as opposed to interactive traffic, gain high throughput. A list similar to that for delay can be proposed; these are issues which may be important for the achievement of high throughput, depending on the kind of network considered:

1. Circuit bandwidth. The routing algorithm must ascertain the effective bandwidth of the circuits in the paths that it chooses for high throughput traffic. This includes such effects as the use of the line by other nodes, the deterioration due to retransmissions, and so on. It is clear that in this regard the routing program is capable of much more than a passive measurement role. Since it is routing which determines at each node how much traffic to send over a particular line, it is possible to regulate the flow over high throughput paths if desired. It is even possible, in order to provide as high a utilization rate as possible, to ensure that all traffic is long packets and messages, although sending all interactive traffic by other routes may penalize that traffic in terms of delays.

2. Node bandwidth. The same comments apply to node bandwidth.

3. Buffering. We will assume that routing and flow control are different operations and are unrelated. Thus, the routing program need not be concerned with the existence of buffering at any level in the network.

4. Multiple paths. We have already mentioned the necessity for load-splitting in certain network situations. It is likely to be much more relevant for high-throughput applications than for low-delay traffic, but the technique of using many paths to a destination is of fundamental importance.

Cost. We now turn to a consideration of the relationship between routing and network cost. There are 3 basic network resources which the routing algorithm utilizes, and it can take several actions to utilize them effectively:

1. Line bandwidth. A primary consideration here is that the routing algorithm elect the paths with the fewest intermediate nodes, thus minimizing the total line bandwidth utilization. This is an important argument, and the basis for several algorithms based on shortest path routing. We should note that this goal may be in direct conflict with the goals of low delay and high throughput, though there are often cases, particularly in networks with uniform node and line characteristics, when shortest path routing is an excellent policy.

2. Node bandwidth. A second cost factor, similar to the first, is node bandwidth. Again, the routing algorithm which chooses short paths over long ones is at an advantage here. In both cases, the routing program would do well to seek underutilized resources rather than concentrating traffic on a few paths.

3. Node storage. The final cost factor is node storage, a specific instance of the higher cost of concentrated traffic. The routing algorithm has the capability of keeping the network queues as short as possible if that is a specific objective. Avoiding the inefficient use of storage in overlong queues may also tend to defer problems of congestion, which also make the network efficiency suboptimal.

A different cost consideration is that of network connectivity. Here too, the routing algorithm is an important factor in determining cost, though in a more indirect fashion. The cost of connecting the network is related to how adaptive the routing procedures are in practice. For instance, some networks have been proposed with fixed routing matrices giving two alternate routes to each node. In this kind of a network, it is essential to provide enough connectivity so that nodes are seldom declared unreachable by the routing algorithm (this may happen even though a network path exists between them). Of course, this raises the cost of the network. A similar problem holds for routing algorithms which adapt slowly, or only with human intervention, or only with some given probability of accuracy, and so on. When we discuss area routing later in Section 4, it will be clear that the problem of rapid and accurate determination of reachability is an important and difficult problem in networks with hundreds of nodes or more.

Reliability. The last performance measure we will discuss is network reliability. Routing can have several different kinds of effects on reliability. In terms of network use, the important measure is the fraction of messages which are undelivered due to failures in the communications subnetwork. The routing program has as its main function in the network the efficient and reliable delivery of messages to their destinations. Despite all kinds of component failures, from lines to nodes to Hosts, the routing algorithm should continue to deliver messages properly or report that they are undeliverable because the destination is unreachable or not functioning.

The parallel requirement concerning the reliability of network connectivity has been examined in the section above on network cost. The appropriate measure here is the fraction of the time that the routing algorithm is in error concerning the reachability of some node.

A different set of issues arises in the relationship between the routing program itself and network reliability. Given the central role of the routing process in any network, it is particularly important that routing never break down altogether, the way most systems, software and hardware, eventually do. The reason is clear: if the routing in the network is incorrect or nonfunctioning, the network is completely unusable. This means that a different measure of routing performance is the percentage of network unavailability that is due to routing algorithm failures. In summary, the routing program must also be considered as another module of network software, with some given level of reliability, which is more sensitive in terms of network reliability than most other modules, because of its global impact.

4.1.7. Routing cost measures

There are five basic costs involved in the continuous operation of a routing algorithm, and they are listed in fig. 11. All these factors act to reduce the effective capabilities of the nodes and lines in the network with regard to the processing of data packets. In

Nodal Bandwidth	**Line Bandwidth**
bits/sec/node	**bits/sec/line**
Nodal Delay	**Line Delay**
sec/node	**sec/line**
Nodal Storage	
bits/node	

Fig. 11. Routing costs.

other words, these costs are various kinds of network system overhead. First, there is the CPU utilization at each node in the network needed to perform the calculation of the new best routes to all destinations. Second, there is the delay at each node associated with the processing of the routing information, introducing delays in the processing of packets. Third, there is the strorage needed at each node as a data base for the routing calculation, and for the exchange of routing information with other nodes. Fourth, there is the line bandwidth used for the exchange of routing information between nodes. Finally, there is the line delay caused by the fact that sometimes a routing message is being transmitted at a time when a data packet is queued. We will now consider each of these costs in more detail.

Line bandwidth. The first cost factor to consider is the fraction of the available line bandwidth needed to exchange routing messages between nodes. Clearly, this bandwidth is a function of the size of the routing message and its frequency as previously stated in section 2.2.2:

$$\mathrm{BWC}_r = B_r * F_r.$$

Depending on the algorithm, one may choose also to make the bandwidth used for routing a fixed number of bits per second, regardless of available line bandwidth, or one may wish to keep the bandwidth used for routing below some acceptable overhead fraction of the line bandwidth. Thus, on slow lines, routing would be sent less often or in an abbreviated form. One may also choose different priority strategies for the transmission of routing. As we have pointed out, routing messages are very important, and should probably take precedence over most other traffic. However, it may be desirable to specify that some classes of transmission take priority over routing messages. In this way, the routing messages need not introduce added delays to special high-priority messages, though they still represent a reduction in the effective bandwidth of the communications circuits.

Line delay. Next we examine the added delays on circuits caused by the transmission of routing messages. Consider a single line with one routing message of length B_r sent with frequency F_r per second, and a very light data traffic load. Then the probability of a data packet having to wait is the ratio of the time duration in which routing is being sent to the routing period. For a given circuit bandwidth BWC, this probability of line delay is:

$\mathrm{BWC}_r/\mathrm{BWC} = B_r * F_r/\mathrm{BWC}$.

Given that a packet must wait, the average wait is half the maximum, or

$\mathrm{ALD} = B_r/(2*\mathrm{BWC})$.

The expected line delay is the product of the probability of line delay and the average line delay:

$\mathrm{ELD} = (\mathrm{BWC}_r/\mathrm{BWC})*\mathrm{ALD}$.

That is,

$\mathrm{ELD} = (B_r*B_r*F_r)/(2*\mathrm{BWC}*\mathrm{BWC})$.

This means that the delay to packets due to routing messages increases as follows:

linearly with increasing routing frequency,
quadratically with increasing routing message length,
quadratically with decreasing line bandwidth.

The situation is depicted for several representative values of the parameters in fig. 12.

The reason that the analysis presented above is valid only for light traffic loads is that a queueing phenomenon causes total network delay to increase

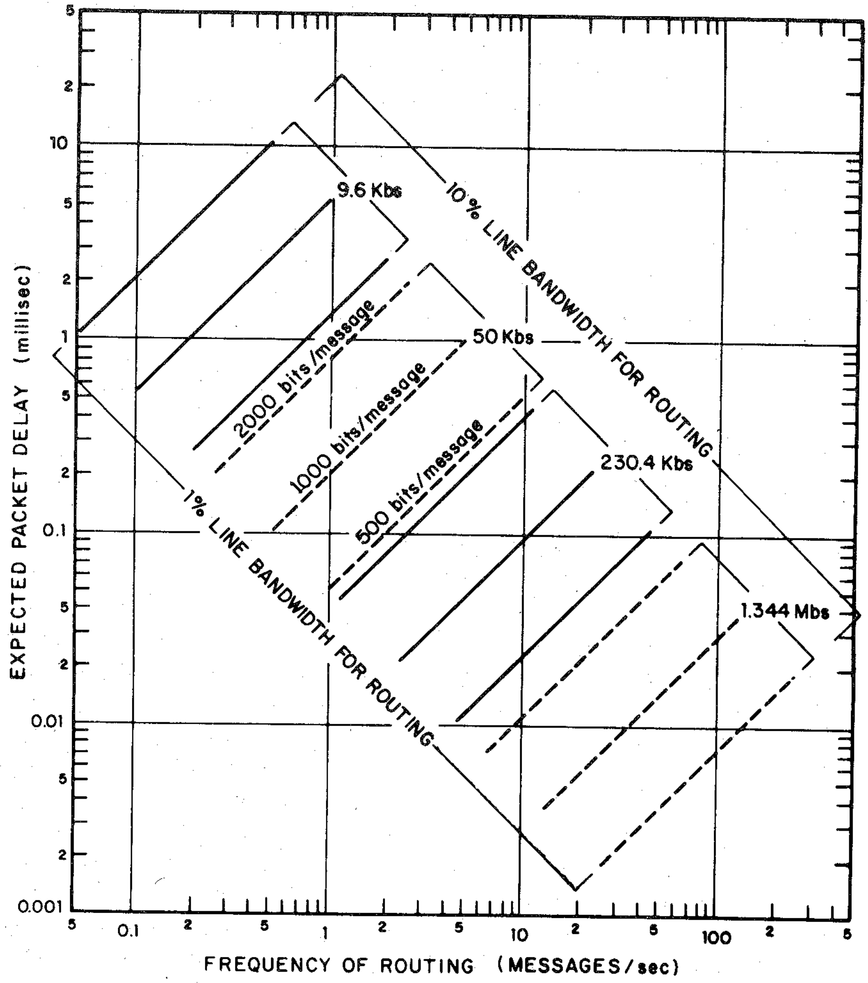


Fig. 12. Expected packet delays due to routing messages.

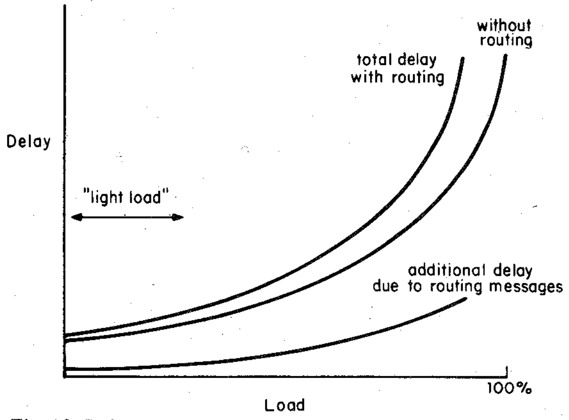


Fig. 13. Delay due to routing vs. traffic load.

nonlinearly with load. Therefore, the added traffic due to the presence of routing messages has a correspondingly greater effect in terms of delay at high loads, as shown in fig. 13.

Nodal bandwidth. The calculation of the best routes to all destinations represents a steady, periodic demand for CPU processing time. One can view the computation as a certain percentage of overhead in the CPU bandwidth available for message processing. We have called this factor $\mathrm{BWP_r}$, the fraction of the bandwidth of the processor used for routing. In general, it has two components, one based on the time taken to process routing messages on each line, and the other a simple periodic component:

$$\mathrm{BWP_r} = \mathrm{NLN}*P_r*F_r + P_p,$$

where NLN is the number of lines per node, P_r is the processor time for routing messages, F_r is the frequency of routing messages, and P_p is the fractional overhead of processor time for periodic processing of routing. For instance, if $\mathrm{NLN} = 2$, $F_r = 5$ messages/sec, and $P_p = 1\%$, then $\mathrm{BWP_r} = 3\%$. If $\mathrm{BWP_r}$ becomes too large, the processor becomes much less cost-effective in its primary role as message processor. Therefore, a major cost consideration in evaluating a routing algorithm is the number of CPU cycles per second it requires in each node of the network.

Nodal delay. Along with the reduction in effective nodal processing power comes the effect of delays in the processing of packets while the routing computation is proceeding, given that it takes priority over data processing. Suppose one routing message is received from each of NLN adjacent nodes with frequency F_r messages per second. Then, given the time P_r and the fractional time P_p above, there are three cases to consider for nodal delay:

1. $\mathrm{NLN}*P_r*F_r \gg P_p$, message processing dominates;
2. $\mathrm{NLN}*P_r*F_r \ll P_p$, periodic processing dominates;
3. $\mathrm{NLN}*P_r*F_r = P_p$, factors have equal magnitude.

If we assume that $\mathrm{NLN}*P_r*F_r < 1$ (= total processor bandwidth), and that packet processing time is infinitesimal, we can analyze each of these cases:

1. The probability that a packet has to wait is

$$\mathrm{NLN}*F_r*P_r$$

and if it must wait for only one message, it experiences an average wait of

$$\tfrac{1}{2}P_r.$$

This means that the expected delay is

$$\tfrac{1}{2}\mathrm{NLN}*F_r*P_r*P_r$$

which is a lower bound on the actual value. It can be shown with a more detailed analysis that this is the expected value for delay under the condition that $1/(F_r*P_r) \gg \mathrm{NLN}$, which means that there are many more time periods in which the processor is able to process routing inputs than there are adjacent nodes to send the routing. In practice, this is the only assumption that makes sense, since otherwise $\mathrm{BWP_r}$ becomes close to unity, and there is no bandwidth available for data traffic.

2. This case can be analyzed as line delay was analyzed above, based on the length and frequency of the periodic processing, P_p.

3. This case can be analyzed on the basis of the first two cases, summing the effects of each of the terms.

Nodal storage. The final cost factor we will discuss is the storage required at each node to maintain the routing information. This cost may vary greatly among various algorithms, depending on how much information is needed in the routing computation, and also on the method of exchanging routing messages. The node must save the incoming routing information in some fashion, then the routing computation may generate other, new information, and the transmission of routing messages to the adjacent nodes may call for still more storage for data. In a small communications processor, memory is dedicated to a fairly small program, and message buffers. Any storage used for the routing calculation must be taken from the message buffer pool, either once and for all at design time, or dynamically. Therefore, the storage requirements of the routing algorithm represent still another overhead factor.

4.1.8. The ARPA network routing algorithm

In this section we describe the routing algorithm originally installed in the ARPA Network, and examine it from the point of view of the performance and cost criteria outlined above.

The ARPA Network algorithm can be summarized as follows. This 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 which line to use in transmitting a packet addressed to another destination. This selection is made by a simple table lookup procedure. For each possible destination, an entry in the routing table designates the appropriate next line in the path.

Each IMP maintains a network delay table which gives an estimate of the delay it expects a packet to encounter in reaching every possible destination over each of its output lines. This table and other tables mentioned below are shown in fig. 14 as kept by IMP 2, for example. Thus, the delay from IMP 2 to IMP 5 using line 3 is found to be 4 in the Network Delay Table. Periodically, every $\frac{2}{3}$ of a second, the IMP selects the minimum delay to each destination and puts it in the minimum delay table. It also notes the line giving the minimum delay and keeps the number of the line in a table for use in routing packets. Also every $\frac{2}{3}$ of a second, the IMP passes its minimum delay table to each of its immediate neighbors, that is, it sends the minimum delay table out each of its phone lines. Of course, before the minimum delay table is transmitted to the neighboring IMPs, the IMP sets the minimum delay to itself to zero.

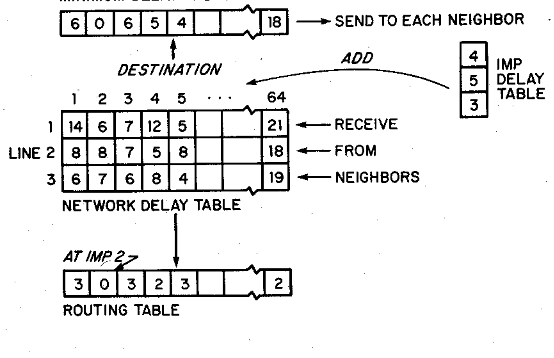


Fig. 14. The ARPA network routing tables.

Since all of the neighbors of an IMP are also sending out their minimum delay table every $\frac{2}{3}$ second, with their own entry set to zero, an IMP receives a minimum delay table from each of its neighbors every $\frac{2}{3}$ second. These tables are read in over the rows of the delay table as they arrive. The row to be written over is the row corresponding to the phone line that the arriving minimum delay table came in over. After all the neighbors' estimates have arrived, the IMP adds the delay saved by the IMP itself to the neighbors' estimates. This is done by adding the IMP delay table, i.e., the contribution of this IMP to the total delay to each destination, to each column of the delay table. Thus the IMP has an estimate of the total delay to each destination over the best path to that destination.

In parallel with this computation, the IMPs also compute and propagate shortest path information in a similar fashion. This information is used only in the determination of connectivity. An upper limit of the number of lines in the longest path in the network is used as the cut-off for disconnected or nonexistent nodes.

Now let us consider the performance of this algorithm. First of all, it explicitly determines the connectivity of the network, since all IMPs are continuously exchanging the length of the shortest path from each IMP to each other IMP. Information travels at roughly $\frac{2}{3}$ of a second per line, so that changes in topology are recognized by the whole network in a matter of a few seconds. This figure is probably acceptable if one assumes that the network connectivity does not change too often. Second, the algorithm also explicitly calculates the path of least delay. However, here the approximations due to the low frequency of routing update mean that the delay for traffic at one instant is a function of the traffic of several seconds before. This could potentially lead to oscillations and poor line utilization. The ARPA Network algorithm attempts to head off this class of problems by biasing delay heavily toward the shortest path. That is, delay is measured by the number of packets on an output queue, plus a fixed increment, so that even an empty queue represents additional delay.

The algorithm has several faults, some of which are relatively simple to cure, and others which are more fundamental in nature. The strong bias in the algorithm towards the shortest path is basically a good idea, and leads to stable flows near optimum values. However, the bias makes the algorithm somewhat insensitive to changes in traffic patterns, so that global optimization of delay and throughput is not likely as network loading increases. A second fault is that the algorithm only maintains one route per destination, updated every $\frac{2}{3}$ second. This means that no load-splitting is possible, at least not on a short term basis. The algorithm could be modified to use one of several routes to each destination, with weights assigned to each. Further, the algorithm might maintain additional data on the loading of the various paths, to facilitate more rapid adaptation to changes in traffic. This change, combined with the expansion to several routes, might also lead to smoother and more uniform adaptation.

The ARPA Network routing algorithm is quite a good design in many respects. Perhaps its strongest point is that it is simple. The IMP does not have to know the topology of the network, or even the identity of its neighbors. When IMPs and lines go down, the algorithm functions as usual, and the new routing information propagates through the network by a process of exchanges between neighbors. Therefore, the algorithm scores quite well in reliability. Although there are no explicit controls to ensure fairness to competition, the algorithm does relatively well in this category as well.

Finally, the algorithm is not a costly one in terms of the measures discussed above. The program in the IMP picks the minimum delay and hop counts from the routing messages received from each neighbor, for all destinations. Thus, the calculation is proportional to the number of IMPs in the network, and the number of lines connected to each IMP. The routing computation takes up about 5% of the CPU bandwidth of the IMP. The delay and hop information is packed into a single 16-bit word, so that the routing message sent out each line consists of 64 words, one for each IMP in the network, plus some header information. This amounts to less than 2% of the bandwidth of a 50 Kbs line. At these low bandwidth rates, added node delays and line delays are not appreciable. In addition to the storage required for sending the routing message out each line (one copy of the message is shared by all lines), the IMP reserves storage for receiving routing messages from each of its lines. These tables, together with its own directory of the best line to each destination, amount to about 3% of the core storage on an IMP. In summary, the IMP routing algorithm is a simple, inexpensive algorithm which performs well in steady state, and in reacting to small changes in traffic.

4.2. Node-to-node transmission procedures

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

4.2.1. Buffering and pipelining

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

4.2.2. Error control

We have discussed many of the aspects of node-to-node error control above: the need for a packet checksum, its size, the basis of the acknowledgment/retransmission system, the decision on whether the line is usable, and so on. These procedures are critical for network reliability, and they should therefore run smoothly in the face of any kind of node or circuit failure. Where possible, the procedures should be self-synchronizing; at least they should be free from deadlock and easy to resynchronize [24].

4.2.3. Storage allocation and flow control

Storage allocation can be fairly simple for the packet processing algorithms. The sender must hold a copy of the packet until it receives an acknowledgment; the receiver can accept the packet if it is without error and there is an available buffer. The receiver should not use the last free buffer in memory, since that would cut off the flow of control information such as routing and acknowledgments. In accepting too many packets, there is also the chance of a storage-based deadlock in which two nodes are trying to send to each other and have no more room to accept packets. This is explained fully in [19].

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

4.2.4. Priority

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

4.2.5. Packet size

There has been much thought given in the packet-switching community to the proper size for packets. Large packets have a lower probability of successful transmission over an error-prone telephone line (and this drives the packet size down), while overhead considerations (longer packets have a lower percentage overhead) drive packet size up. The delay-lowering effects of pipelining become more pronounced as packet size decreases, generally impkroving store-and-forward delay characteristics; further, decreasing packet size decreases, generally improving store-and-because they are waiting behind full length packets. However, as the packet size goes down, potential effective throughput also goes down due to overhead. Metcalfe has previously commented on some of these points [28].

Kleinrock and Naylor [23] recently suggested that the ARPA Network packet size was suboptimal and should perhaps be reduced from about 1000 bits to 250 bits. This was based on optimization of node buffer utilization for the observed traffic mix in the network. However, in [10], we point out that the relative cost of node buffer storage vs. circuits is possibly such that one should not try to optimize node buffer storage. The true tradeoff which governs packet size might well be efficient use of phone line bandwidth (driving packet size larger) vs. delay characteristics (driving packet size smaller). If buffer storage is limiting, one should just buy more (up to the limits of the address space, of course). Further, it is probably true that if one is trying for high bandwidth utilization, buffer size must be large. That is, high bandwidth utilization probably implies the use of large packets, which implies full buffers; when idle, the buffer size does not matter.

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

Davies and Barber [12] have been cited as recommending a minimum length "packet" of about 2000 bits because they have concluded that most of the messages currently exchanged within banks and airlines fit nicely in one packet of this size. To clarify this point, we note that they use the term "packet" for the unit of information we call a "message" and thus are not actually addressing the issue of packet size. We discuss message size below.

4.2.6. The ARPA Network IMP-to-IMP transmission control procedure

We now take a close look at the algorithm used in the ARPA Network for IMP-to-IMP transmission control. As has been noted elsewhere, the inter-IMP modem interface hardware has the capability of generating checksums for outgoing packets and checking the checksums on incoming packets. This allows packets which are damaged in transmission to be detected and discarded without acknowledgment. Packets correctly received are acknowledged. A good IMP-to-IMP transmission control algorithm must detect errors, acknowledge good transmissions, and provide retransmission in the event of errors. In addition, the IMP-to-IMP transmission control algorithm is improved if it detects duplicates that are sometimes generated by retransmission. An algorithm which performs all four of these tasks is described below (it is like the HDLC data communications standard with a

1-bit sequence number per logical channel and up to 8 logical channels).

A number of logical "channels" are maintained between each pair of IMPs. Consider only one channel to begin with, and further consider packet transmissions in only one direction on this channel. Of course, acknowledgments go the other direction on the channel. At both the transmit and receive end of this channel a one bit sequence number is kept. We call this bit an odd/even bit. Both transmit and receive odd/even bits are initialized to be zero. Also, at the transmit end, a used/unused bit is kept for the channel. It is of course initialized to zero, meaning unused. When it is time to transmit a packet, a check is first made for the channel's being unused. If it was previously unused, it is marked as used and the packet is transmitted. The state of the transmit odd/even bit is included with the packet. When the packet arrives at the receiver, assuming the packet is received correctly, the packet's odd/even bit is checked against the receive odd/even bit. If they match, the packet is accepted and the receive odd/even bit is complemented. Otherwise, the packet would be ignored. In any case the receive odd/even bit is returned as an acknowledgment. At the transmitter, if the acknowledgment bit does not match the transmit odd/even bit, the packet has been successfully sent and acknowledged and the packet can be discarded, the channel marked unused, and the transmit odd/even bit complemented. Otherwise the acknowledgment is a duplicate and is ignored. Suppose now a second copy of the packet arrives at the receiver, a packet which was sent before the first acknowledgment had a chance to get back to the transmitter. When this packet arrives at the receiver, its odd/even bit does not match the receive odd/even bit and so that packet is discarded as a duplicate. Nonetheless, an acknowledgment is sent for the packet using the present state of the receive odd/even bit. When the acknowledgment gets to the transmitter, it does match the transmit odd/even bit, so the acknowledgment is a duplicate and is ignored.

The acknowledgment bit is the state of the receive odd/even bit after it is complemented rather than before it is complemented. Hence the need for the "not match" rule when the acknowledgment arrives at the transmitter. A closely related algorithm was reported in [4].

Because of the potentially long distances between IMPs, one channel is not enough to keep the inter-IMP lines fully loaded. Therefore, eight logical channels are supplied between each pair of IMPs (32 are supplied between Satellite IMPs). It is not necessary to maintain ordering of IMP-to-IMP transmissions since packet ordering is performed at the destination IMP. This means that the transmit channels can be filled in any convenient order, and at the receive side, packets can be forwarded onwards as soon as they are correctly received regardless of the channel over which they arrived.

To avoid requiring separate packets for acknowledgments, acknowledgment bits are "piggy-backed" in packets going the other way on the line. In fact, all eight receive odd/even bits are transmitted with every packet going the other way. In the absence of any traffic going the other way on the line, a packet carrying only the eight acknowledgments is sent. In either case, the acknowledgments get back to the transmitter as fast as possible. Therefore, the transmitter knows very soon whether a packet requires retransmission or not, allowing the use of a minimal timeout before retransmission. "Piggy-backing" all acknowledgments into every packet going the other way saves program bandwidth, line bandwidth, and buffer space over a system which sends individual acknowledgments in individual packets.

In view of the use of a number of channels and the delay encountered on long lines, some packets might have to wait an inordinately long time for transmission. Traffic that is essentially interactive should not be subjected to waiting for several thousand-bit packets to be transmitted, multiplying by ten or more the effective delay seen by the source. Therefore, the IMPs maintain two sets of queues for each output line, and service all priority transmissions before any regular transmissions. Preemptive priority is not employed.

4.2.7. Details of the inter-IMP packet format

Fig. 15 shows the format of packets as they appear on the inter-IMP circuits. Packets are permitted to be variable length up to a maximum of about 1000 bits. The IMP's modem interface transmission hardware adds framing characters to each packet as it is transmitted onto an inter-IMP circuit. At the front of the packet two characters (DLE and STX) are added indicating the beginning of the packet. At the end of the packet two characters (DLE and ETX) are added indicating the end of the packet. The checksum is appended after the end of packet characters. The IMP's modem interface reception hardware has the capability of detecting the start and the end of a

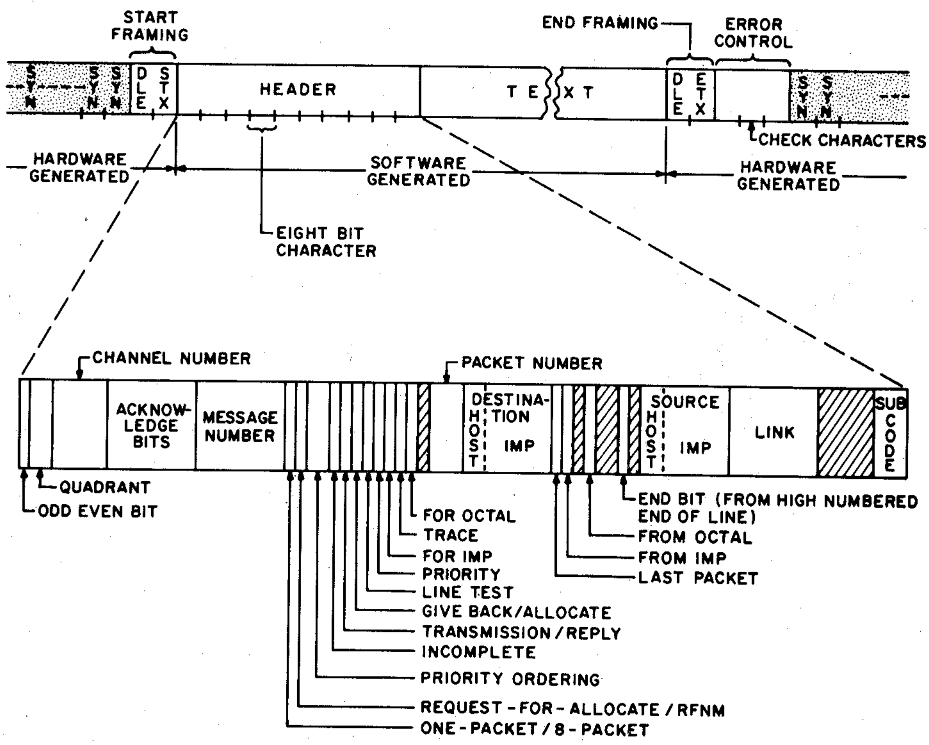


Fig. 15. Inter-IMP packet format.

packet from these framing characters thus freeing the program from the burden of detecting packet boundaries. Between packets the transmission hardware automatically generates synchronizing characters (SYN) which the reception hardware automatically discards.

There is no restriction on the content of a packet. Arbitrary sequences of bits may be transmitted without restriction. This transparency is achieved by a method known as DLE-doubling. If the data in the packet itself contains a DLE-character (the character which introduces the packet start and packet end sequences), that DLE-character in the data is doubled by the transmission hardware. At the receiver, the hardware collapses double DLEs in the data back into one; so there is complete transparency on the inter-IMP channels. This is a very important point. All too many networks require transmission to be limited to characters from a particular character set. Besides preventing the network's users from sending arbitrary messages, the designers of such networks are themselves prevented from such things as loading programs over the network.

In addition to showing the part of the packet format done with hardware, fig. 15 also shows the channel field, "piggy-back" acknowledgment field, etc., mentioned in the preceding section. The portions of the packet which carry the end-to-end control information (e.g., destination address, message sequence numbers, etc.) are also shown.

Notice that much of the inter-IMP communication algorithm is performed with hardware. Software is particularly bad at generating powerful checksums, scanning for packet boundaries, and so forth, especially when such chores must be done on a character-by-character or bit-by-bit basis. Therefore, they are done with hardware in the ARPA Network.

4.2.8. Inter-IMP buffering and allocation

Consider fig. 16. Two-way single-packet traffic flows between A and A′ and also between B and B′, and is constrained by network topology to use the circuit between IMPs C and D. Suppose all the buffer storage in IMP C can become filled with packets on the output queue to IMP D, and all the buffer storage in IMP D can become filled with packets on the output queue to IMP C. In this case, both IMP C and IMP D would be engaged in a direct confrontation in which both IMPs must lose all incoming packets (and acknowledgments) as neither machine has any buffer

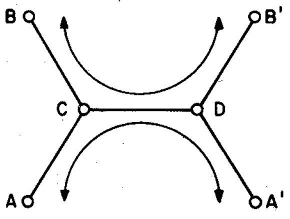


Fig. 16. Store-and-forward lockup.

space with which to receive inputs. We call this a store-and-forward lockup.

It is straightforward to prevent such store-and-forward lockups, and this is done in the ARPA Network implementation. The key technique used is to guarantee that sufficient buffers are reserved so that it is always possible to input and to output one packet over each circuit. Thus, packets (and acknowledgments) can always be passed between neighboring machines (albeit at a trickle, perhaps). In particular, in the IMP system, one buffer is always allocated for output on each line, guaranteeing that output is always possible; 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. Additionally, an attempt is made to provide enough store-and-forward buffers so that all lines may operate at full capacity.

We conclude this section with two interesting notes: 1) negative acknowledgments could be useful in activating dormant buffers more quickly, but they add complexity and are not used in the ARPA Network; 2) more complex forms of store-and-forward lockup than that given in the example above are possible, and while most are protected against in the ARPA Network implementation, at least one rare case is not protected against. See [19] for further discussion of this point.

5. Source-to-destination system design

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

5.1. End-to-end transmission procedures

There is a considerable controversy at the present time over whether or not a store-and-forward subnetwork of nodes should concern itself with end-to-end transmission procedures. Many workers [33] feel that the subnetwork should be close to a pure packet carrier with little concern for maintaining message order, for high levels of correct message delivery, for message buffering in the subnetwork, etc. Other workers, including ourselves [10], feel that the subnetwork should take responsibility for many of the end-to-end message processing procedures. Of course, there are some workers who hold to positions in between [7]. However, many design issues remain constant whether these functions are performed at Host level or subnetwork level, and we discuss these constants in this section.

5.1.1. Buffering and pipelining

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

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

5.1.2. Error control

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

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

An inherent characteristic of packet-switching networks is that some messages or portions of messages (i.e., packets) will fail to be delivered, and there will be some duplicate delivery of messages or portions of messages, as described in the section on network properties. (Please note that throughout the remainder of this subsection we use the word "message" to mean either messages or portions of messages (i.e., packets).)

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

Duplicate messages can be detected with an identifying sequence number such that messages which arrive from a prior point in the sequence are recognized as duplicates. What should be noted carefully here is that duplicate messages can arrive at the destination up to some time, possibly quite long, after the original copy, and the sequence number must not complete a full cycle during this period. For example, if a network goal is to be able to transmit 200 minimum length messages per second from the source to the destination and each needs a unique sequence number, and if it is possible for messages to arrive at the destination up to 15 seconds after initial transmission from the source, then the sequence number must be able to uniquely identify at least 3000 packets. It is usually no trouble to calculate the maximum number of messages that can be sent during some time interval. What is more difficult is to limit the maximum time after which duplicate messages will no longer arrive at the destination. One method is to put a timer in each message which is counted down as the message traverses the network; if the timer ever counts out, the message is discarded as too old, thus guaranteeing that no messages older than the initial setting of the timer will be delivered to the destination. Alternatively, in practice one can make a reasonably good approximate calculation of the maximum arrival time through study of all the worst case paths through the network and all the worst case combinations of events which might cause messages to loop around in the network.

In either case, there certainly must be mechanisms to resynchronize the sequence numbers between the source and the destination at node start-up time, to recover from a node failure, etc. A good practice is to resynchronize the sequence numbers occasionally even though they are not known to be out of step. A good frequency with which to do redundant resynchronization would be every time a message has not been sent for longer than the maximum delivery time. In fact, this is the maximum frequency with which the resynchronization can be done (without additional mechanisms); if duplicates are to be detected reliably, the sequence number at the destination must function without disruption for the maximum delivery time after the "last message" has been sent. If it is desirable or necessary to resynchronize the sequence numbers more often than the maximum time, an additional "use" number must be attached to the sequence number to uniquely identify which "instance" of this set of sequence numbers is in effect; and, of course, the packets must also carry the use number. This point is addressed in greater detail in [27] and [36].

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

Finally, there must be some inquiry-response system from the source to the destination to complete the process of detecting lost messages. When the proper reply or acknowledgment has not been received for too long, the source may inquire whether the destination has received the message in question. Alternatively, the source may simply retransmit the message in question. In any case, this source inquiry and retransmission system must also function in the face of duplicated or lost inquiries and inquiry response control messages. As with the inter-node acknowledgment and retransmission system, the end-to-end acknowledgment and retransmission system must depend on positive acknowledgments from the destination to the source and on explicit inquiries or retransmissions from the source. Negative acknowledgments from the destination to the source are never sufficient (because they might get lost) and are only useful (albeit sometimes very useful) for increased efficiency.

5.1.3. Storage allocation and flow control

One of the fundamental rules of communications systems is that the source cannot simply send data to the destination without some mechanism for guaranteeing storage for that data. In very primitive systems one can guarantee a rate of disposal of data, as to a line printer, and not exceed that rate at the data source. In more sophisticated systems there seem to be only two alternatives. Either one can explicitly reserve space at the destination for a known amount of data in advance of its transmission, or one can declare the transmitted copy of the data expendable, sending additional copies from the source until there is an acknowledgment from the destination. The first alternative is the high bandwidth solution: when there is no space, only tiny messages travel back and forth between the source and destination for the purpose of reserving destination storage. The second alternative is the low delay solution: the text of the message propagates as fast as possible. See [24] for a more lengthy discussion.

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

As we stated earlier, our experience and analysis convinces us that if both low delay and high throughput are desired, then there must be mechanisms to handle each, since high throughput and low delay are conflicting goals. This is true, in particular, for the storage allocation mechanism. It has occasionally been suggested [6], mainly for the sake of simplicity, that only the low delay solution be used; that is, messages are transmitted from the source without reservation of space at the destination. Those people making the choice never to reserve space at the destination frequently assert that high bandwidth will still be possible through use of a mechanism whereby the source sends messages toward the destination, notes the arrival of acknowledgments from the destination, uses these acknowledgments to estimate the destination reception rate, and adjusts its transmissions to match that rate. We feel that such schemes may be quite difficult to parameterize for efficient control and therefore may result in reduced effective bandwidth and increased effective delay. If the source never sends to the destination so fast that the destination must discard anything, then the delay is very low, but the throughput is not as high as it might be. Further, unless the source pushes now and then, it will never discover that the destination is able to increase its throughput. On the other hand, when the source is pushing hard enough, the destination may suddenly cut back on its throughput, causing all the messages which will be discarded at the destination due to the sudden cutback to have to be retransmitted, increasing effective delay. If the destination could be predicted to accept traffic at a steady rate and vary this rate only very slowly, the type of feedback system cited above might work. In this case, unacknowledged messages should be retransmitted from the source to the destination shortly after the expected time for the acknowledgment to return has elapsed, if minimum delay and maximum throughput are to be obtained (this is in contrast to the often suggested practice of keying retransmissions to the discard rate). However, in practice, the time for the acknowledgment to return is likely to be very difficult to predict due to variations (possibly rapid) in the transit time of the communications channel and particularly in the response time of the destination. Furthermore, the greater the sum of transit time and response time, the looser and less efficient the feedback loop will be. In fact, there appear to be oscillatory conditions which can occur where performance degrades completely. (Note that if there is much possibility of message loss, then the acknowledgment and retransmission system should allow quite selective retransmission of messages rather than, for instance, requiring a complete window of messages to be retransmitted to effect retransmission of the specific messages requiring it; otherwise, message retransmission will use excessive bandwidth.)

The above discussion assumes that all mechanisms are attempting to minimize the probability of message discard. If, in addition to possible discards at the destination, the communications channel solves its internal problems (e.g., potential deadlocks) with cavalier discarding of messages, or if the destination solves its internal problems with cavalier discarding of messages, the detrimental effects of discarding (reduced effective bandwidth and increased effective dealy) are probably drastically increased. Further, the above discussion assumed the destination was able to minimize the probability of discard. While this may be possible for a single source, we think it is unlikely that the destination will be able to resolve, in a way that does not entail excessive discards, the contention for destination storage from multiple uncoordinated sources. As reported in [19], detrimental contention for destination storage, in the absence of a storage reservation mechanism, happens practically continuously under even modest traffic loads, and in a way uncoordinated with the rates and strategies of the various sources. As a result, well-behaved Hosts may unavoidably be penalized for the actions of poorly-behaved Hosts.

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

5.1.4. Precedence and preemption

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

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

We think the correct strategy is probably a mixture of the strategies above. Possibly some resources, on a very limited basis, should be reserved for the tiny amount of flash traffic. This guarantees minimum delay without any queueing latency. For the rest of the traffic, the normal delivery times are probably acceptable. The presence of higher priority traffic can cause gradual throttling of lower priority traffic, without loss of state information. As the time to do this graceful throttling is normally only a fraction of a second, the higher priority traffic has no real reason to demand instantaneous, information-losing preemption of the lower priority traffic.

5.1.5. Message size

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

We have already noted the relationship between delay and packet size. Delay for a small priority message is, to first order, proportional to the packet size of the other traffic in the network. Thus, small packets are desirable. Larger packets becomes desirable only when lines become so long or fast that propagation delay is larger than transmission time.

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

When a channel has an appreciable delay, it is necessary to buffer several pieces of data in the channel at one time in order to obtain full utilization of the channel. It makes little difference whether these pieces are called packets which must be reassembled or messages which must be delivered in order.

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

5.1.6. Multiplexing and addressing

Up to this point in our paper, we have not been very specific about whether the above-mentioned flow control, sequencing, error control, etc. mechanisms were performed for each pair of communicating processes, or whether several processes communicating between a given pair of source and destination nodes share a set of these control mechanisms. The tradeoff is between overhead and precision of control. If many conversations are multiplexed on each instance of a source-to-destination control mechanism, the control overhead is lower than if each conversation has its own control mechanism. On the other hand, if several conversations are multiplexed on the same control mechanism, all the conversations tend to have to be treated equally (e.g., if one is stopped, all are stopped); while if each conversation has its own control mechanism, exact decisions about the allocation of various resources to the various conversations can be made. To give some examples of the latter, conversations over separate control mechanisms can be given differing allocations, priorities, treatments of error conditions, etc.

Another issue is the management of the space available for control mechanisms when it is insufficient to handle the number of conversations competing for the communications channel. Should latecomers be left out until resources are available, or should some way be found to multiplex the available control mechanisms in time among the demanding conversations? We believe the latter should be done. The key here is not to allow users to explicitily acquire and hold resources (e.g., control mechanism space) needed for interprocess communication. Instead, the system should notice which users are actively communicating and dynamically gather the needed resources by garbage-collecting the resources previously being used by users which appear inactive. This dynamic assignment of resources is obviously not fundamentally different from the scheduling of any limited resource in an operating system (e.g., memory, CPU cycles, the I/O channel to the disk) and therefore has all the normal possibilities for thrashing, unfairness, and so on, if care is not taken.

Once decisions in all of the above areas of multiplexing are made, one must choose the addressing mechanism and formats to be used. This is usually quite straightforward. The main point here is that addressing comes last; but very often we see designs begun by choosing the addressing system and format. A similar statement can be made about the choice of all other message formats.

5.2. Division of responsibility between subnetwork and Host

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

5.2.1. Extent of message processing in the subnetwork

There has been considerable discussion in the packet-switching community about the amount and kind of message processing that should be done in communications subnetworks. An important part of the ARPA Network design which has become controversial is the ARPA Network system of messages and packets within the subnetwork, ordering of messages, guaranteed message delivery, and so on. In particular, the idea has been put forth that such functions should reside at Host level rather than subnetwork level [8,9,33].

We summarize the principle usually given for eliminating message processing from the communications subnetwork: a) for complete reliability, Hosts must do the same jobs, and therefore the nodes should not; b) Host/Host performance may be degraded by the nodes doing these jobs; c) network interconnections may be impeded by the nodes doing message processing; d) lockups can happen in subnetwork message processing; e) the node would become simpler and have more buffering capacity if it did not have to do message processing.

The last point is true, although the extent of simplification and the additional buffering is probably not significant, but we believe the other statements are subject to some question. We have previously [10,27] given our detailed reasons for this belief. Here we simply summarize our main contentions about the place of message processing facilities in networks:

a. A layering of functions, a hierarchy of control, is essential in a complex network environment. For efficiency, nodes must control subnetwork resources, and Hosts must control Host resources. For reliability, the basic subnetwork environment must be under the effective control of the node program–Hosts should not be able to affect the usefulness of the network to other Hosts. For maintainability, the fundamental message processing program should be node software, which can be changed under central control and much more simply than all Host programs. For debugging, a hierarchy of procedures is essential, since otherwise the solution of any network difficulty will require investigating all programs (including Host programs) for possible involvement in the trouble.

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

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

d. It is basically cheaper to do message processing in the nodes (small inexpensive computers) than in the Hosts, and it has very few detrimental effects.

5.2.2. Peripheral processor connections

In a number of cases, an organization has desired to connect a large Host to a network by inserting an additional minicomputer between the main Host and the node. The general notion has been to locate the Host-Host transmission procedures in this additional machine, thus relieving the main Host from coping with these tasks. Stated reasons for this notion include:

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

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

First, the I/O arrangments between the main Host and any preexisting peripheral processor were not designed for network connection and usually present timing and bandwidth constraints that greatly degrade performance. More seriously, the logical protocols that may have preexisted will almost certainly preclude the main Host from acting as a general purpose Host on the network. For instance, while initial requirements may only indicate a need for simple file transfers to a single distant point, requirements tend to change in the face of new facilities, and the network cannot then be used to full advantage [9].

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

The long term solution to the dilemma is to have the various manufacturers support hardware and software interfaces that connect to widely used networks. This is not likely to occur until commercial networks exist and are widely available. In the meantime, potential Host organizations that wish to use early networks (like the ARPA Network) should try to find ways to put the network connection directly into the main Host. An anthropomorphic illustration may be helpful: the network is, among other things, a set of standardized protocols or languages. A potential network Host is in the position of a person who needs to have dealings with people who speak a language he does not know. If he does not want to learn the language, he can indeed choose to use an interpreter, but performance is poor, the process is very inconvenient, expensive, and unpleasant, and subtle meaning is always lost. The situation is quite similar when a Host tries to work through a peripheral processor. If a Host wishes to interact with a network, it is usually unrealistic to try to make the Host think that the network is a card reader or some other familiar peripheral. As usual, you get what you pay for.

5.2.3. Other message services

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

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

5.3. The ARPA Network source-to-destination transmission algorithms

In this section we describe the mechanisms by which the IMPs in the ARPA Network manage the flow of data from a source Host to a destination Host.

5.3.1. The ARPA Network message format

In the ARPA Network, a Host presents messages to the IMP to which the Host is directly connected. These messages must be less than about 8100 bits long, and the messages are transmitted between the Host and IMP over the Host/IMP interface, which is rigorously defined in [5]. This interface has two parts, the hardware part and the software part.

The hardware part of the Host/IMP interface itself has two parts, a standard portion supplied with the IMP which is (almost) identical for all Hosts and a special portion supplied by the Host. For simplicity and power, the standard interface has been defined to be full duplex. Electrically, the standard interface follows a bit-by-bit handshaking procedure. The procedure is something like "I'm ready to send a bit." "I'm ready to receive a bit." "Here's the bit." "I got the bit." "Good!" The procedure also provides for saying "That's the last bit in the message." In our opinion this procedure is important. The ARPA Network has to connect together all kinds of different computers with different word lengths, different speeds, different loading, and so forth; and it is desirable to place only minimal constraints on the Hosts' behavior. The above procedure permits this. The asynchronous, bit-by-bit serial interface permits both the Host and the IMP to be able to start and stop transmission whenever necessary. Incidentally, the IMP's side of the above procedure is implemented entirely with hardware. (Note: an optional synchronous Host/IMP interface is also available as described in Appendix F of [5]. It is considerably more complicated, less flexible, and slower than the normal asynchronous, serial interface; and its use is recommended only when a communications interface suitable for operation over long distances is required. HDLC would have been a better choice had it been available at the time.)

The software interface between an IMP and a Host is also simple, using a minimum number of control messages. The Host specifies to its IMP the destination of a message and a few other things in the first 32 bits of the message, called the leader. Messages arriving at the Hosts have the same information in the first 32 bits of the messages, except that the destination is replaced by the source.

Neither the hardware nor the software interface between the IMP and the Host places any constraint on the content of messages other than that they must have legal leaders and must have less than the maximum length. In other words, messages may be sent through the network containing arbitrary sequences of bits.

An IMP breaks messages arriving from its Hosts into packets 1000 or fewer bits long. As the IMP segments a message into packets, it appends to the front of each packet some control information called the header. The header contains the destination, the packet number within the message, a message sequence number which is used for reconstructing the message stream as the messages arrive at the destination, and other control information (e.g., priority information) copied from the message into each constituent packet. This message segmentation and message reconstruction is completely invisible to the Hosts in the ARPA Network implementation of messages and packets.

5.3.2. The ARPA Network source-to-destination transmission procedure

We have already noted that the ARPA Network implementation uses the technique of breaking messages into packets to minimize the delay seen for long transmissions over many hops. The ARPA Network implementation also allows several messages to be in transit simultaneously between a given pair of Hosts. However, the several messages and the packets within the messages may arrive at the destination IMP out of order, and in the event of a broken IMP or line, there may be duplicates. The task of the ARPA Network source-to-destination transmission procedure is to reorder packets and messages at their destination, to cull duplicates, and after all the packets of a message have arrived, pass the message on to the destination Host and return an end-to-end acknowledgment called a Ready For Next Message (or RFNM) to the source.

Up to eight messages are allowed to be in transit from a given source Host to a given destination Host at one time. There is a Host/Host specific sequence number (which we call the message number) assigned out of a message number space of 256 to each message by the source IMP. The destination IMP has a window of eight acceptable message numbers out of this message number space of 256. Messages with out-of-range message numbers, as well as duplicate messages and duplicate packets, are discarded at the destination IMP. Ready For Next Message messages are returned for each message correctly received. The Hosts know nothing about these message numbers: they are used internally by the communications subnetwork of IMPs to order message delivery into the destination Host.

It is desirable to have a priority path as well as a normal path between pairs of Hosts. This is provided in the ARPA Network through use of a second message number between each pair of Hosts. Thus there are two independent message number streams between a pair of Hosts, and messages in one stream (e.g., the priority stream) can be used faster than message numbers in the other (e.g., the normal stream) without interference between the two streams.

In addition to the window of acceptable message numbers that the source and destination IMPs maintain, there is a set of bits corresponding to outstanding messages. The source IMP keeps track of whether a response has come in (typically in the form of a Ready For Next Message) for each message sent. The destination IMP keeps track of whether the message is complete (that is, whether all the packets have arrived). The source IMP also times out the message number, and if a response has not been received for a message for too long a period (e.g., thirty seconds), the source IMP sends a control message with the timed-out message number questioning the possibility of an incomplete transmission. The destination IMP must always return a Ready for Next Message for such a control message stating whether it saw the original message or not, and the source IMP will send the message number in question every few seconds until it receives a response or one of the other of the IMPs decides that there is a hopeless deadlock and more massive corrective action is required (this is discussed below). This technique allows the source and destination IMPs to be synchronized in the event of a lost message or Ready For Next Message. It should be noted that this kind of failure is very infrequent, and happens only when an intermediate IMP fails and in doing so destroys a message.

Of course, as mentioned earlier, the very control messages used for duplicate detection, lost message detection, and order control, must themselves be controlled for duplication, loss, and order. The ARPA Network algorithms do this.

If the IMPs placed no restriction on the number of messages that could simultaneously be in transit to a given destination IMP other than the Host/Host message number window of eight mentioned above, then reassembly storage at the destination IMP could be completely used up by partially reassembled messages from several Hosts, and the IMPs neighboring the destination IMP could fill with store-and-forward packets for the destination IMP. Once this kind of congestion developed, a lockup which has been called reassembly lockup can easily occur when the missing packets for the messages being reassembled are held two or more hops away from the destination. This phenomenon has been discussed extensively in [19].

The IMPs control such congestion by a method based on strict allocation of destination IMP storage. When an IMP has a multi-packet message to send, it first sends off a "request for allocation" of reassembly space to the destination IMP. Some time later it will receive an "allocate" message, which means that the destination IMP has reserved space in which to reassemble the multi-packet message, and then the source IMP can send the multi-packet message. This procedure ensures that the destination is never swamped and the reassembly lockup will not occur.

The request/allocate sequence does introduce a certain amount of overhead, however, for multi-packet messages. Since it is desirable to provide as much bandwidth as possible for multi-packet messages, a mechanism is provided such that there is no necessity for the "request for allocation" in the case of later messages in a steady stream of traffic. When the destination IMP has given a multi-packet message to one of its Hosts, it returns a Ready For Next Message to the source and at the same time allocates reassembly storage for an anticipated next message. The source IMP receives, in effect, a new allocate with the Ready For Next Message and if the source Host sends another message to the destination within 125 milliseconds, the message can be transmitted without waiting for the "request for allocation/allocate" sequence. If the source Host waits too long, the source IMP will return the allocation to the destination IMP with a "give back" message. After this, the next time the Host tries to send, the IMP will again transmit a "request for allocation" and wait for an "allocate" before proceeding.

For single-packet messages it is possible to do away with some of the delay inherent in the "request for allocation" mechanism used for multi-packet messages. The single-packet messages can be sent along with their "request for allocation", if a copy of the message is saved at the source IMP. If the destination IMP can take the message, it does so immediately and returns a Ready For Next Message to the source. If there is not enough storage at the destination, the destination IMP discards the messages, and it sends back an "allocate" when storage becomes available. When the source IMP receives this allocate, it retransmits the message (this time without the request indication). This modified mechanism used for single packet messages is logically identical to that used for multi-packet messages, but it is optimized to take advantage of the highly probable case that the destination will be able to find storage for a single packet immediately.

Once again, and as noted before, all the allocation control messages must themselves be allocated if deadlocks are to be avoided, and the ARPA implementation does this.

5.3.3. The ARPA Network data structures for source-to-destination transmission

In order to successfully and efficiently handle the large number of conversations with various Hosts that can be simultaneously in progress, all of the data structures in the IMP associated with message processing take the form of blocks dynamically gathered from a pool of blocks, each containing a few words of storage. The alternative of keeping linear tables indexed by IMP, Host, or anything else is prohibitively expensive as the network becomes large.

One example of such a dynamic data structure is the transaction block which the source IMP creates when a message is initiated. This block keeps track of whether the message has a copy kept at the source, whether it needs an allocate, and so on. A key function of the transaction block is to hold a copy of the message header to identify the message. When a Ready For Next Message returns from the destination, a Ready For Next Message for the source Host can be formatted in place in the transaction block, solving a difficult storage allocation problem. The transaction blocks can be on a free list, on an active list (i.e., message outstanding), or on a Host queue (i.e., Ready For Next Message to be sent). In addition, they can hold the information as to whether an "allocate" was sent back with the Ready For Next Message.

Another example of a dynamic block is the reassembly block which is kept at the destination IMP for the purpose of ordering the packets of a multi-packet message. It also contains the message header and a list of which packets have arrived. If none has arrived yet, it constitutes a record of an allocation that has been sent to a particular Host or IMP. It can also be on several queues: a free list, an active list, or (theoretically) a Host queue (this last is an implementation option which was not chosen in the case of the IMP system). Notice that in each of these cases, that of the transaction block and the reassembly block, the data structure represents an important resource. There can only be a finite number of blocks; their use must be allocated among several competing source and destination Hosts; critical questions of efficiency and fairness arise in the process of block allocation.

In keeping with this general philosophy of table structure, message blocks are used to keep track of the Host/Host message numbers. This concept allows the set of Hosts on an IMP to send messages to an arbitrary number of other Hosts over a wide range of addresses, with a limited number of message blocks. Several issues arise with consideration to dynamic message blocks:

There must be control messages between the source and destination IMPs of the form "get a block" and "got a block", in order to establish a "conversation" between a given pair of Hosts on the two IMPs.

There must be an error control mechanism to detect duplicate or missing "get a block" and "got a block" messages.

Once a conversation is established, messages can flow. Then there must be a technique to distinguish messages in this conversation from old duplicates from a previous conversation between this pair of Hosts. The messages and packets must carry some identifying number for this purpose.

Conversations should be able to be broken by either end if an IMP finds its storage for message blocks filling up. The messages to do this, "do a reset" and "did a reset" as we call them, must also be error-controlled.

Conversations should begin without undue startup delay.

The dynamic tables should be simplex; that is, one-directional message numbers should be used to avoid deadlocks of the form A tries to talk to B, B tries to talk to C, C tries to talk to A, and none of A, B, or C has any more free blocks to use to begin a new conversation.

The tables should be fast to access at both the source and destination IMPs.

The method used the the ARPA Network for implementing this system is based on a small pool of blocks, each of which carries a "use number", four bits wide, permanently associated with the block. All packets exchanged between IMPs carry the block number to be used in processing the packet and the use number. As explained in more detail below, these numbers provide the key information necessary for error control in a dynamic block environment.

The system works as follows: When a Host at an IMP gives its IMP a message, the IMP first looks to see if a block exists for that source Host to that destination IMP and Host. Instead of searching through all blocks, a "bucket sort" (a simple hash) is performed, to cut the average search length by some number (by 16 in actual fact) using a few bits of the key (source Host, destination IMP, destination Host) to begin the sort. It then checks successive blocks in the bucket led to by the sort (all the blocks in a single bucket are actually chained together for speed of access). If no block is found already existing for this key, a new block must be acquired at both the source and destination IMPs for this Host/Host conversation. The source IMP gets a block from the list of free blocks (the case of no free blocks is discussed later). It puts the new block at the top of the bucket (head of the chain) that it just searched. The program then copies in all the key information, adds one to the use number of the block, initializes the transmit message number entry, turns on a bit to indicate that the block is not yet in use, and calculates the index of the block it found.

The program then constructs a "get a block" message which includes the index number calculated just above and the use number from the block and sends the "get a block" message to the destination IMP. The source IMP then waits for an answering "got a block". The "get a block" must be retransmitted every few seconds if no answer returns. When a "got a block" is returned, the initialization bit is cleared, and the foreign block and use number which arrived in the "got a block" are copied into the source block. Now the message can be sent using the message number window, allocation, and ordering techniques described above.

At the destination IMP, when a "get a block" is received, the program tries to get a free block. If the free list is empty, nothing more is done (in effect, the "get a block" request is ignored). If a free block is available, all the key data carried in the "get a block" is copied into the block and a "got a block" message is constructed including the key data and sent to the source.

When the source IMP sends a packet to the destination, it carries the foreign block number and use number which are kept in the source block. The destination IMP uses this number to calculate the address of its message block and verifies the key information including the use number. In all other ways, the logic discussed above for accepting packets within the message number window is followed. When a Ready For Next Message is generated at the destination IMP, it carries back the block number and use number kept in the destination block. This allows the source IMP to find its block and detect duplicate Ready For Next Messages in a simple manner.

We have explained how blocks are acquired. It is also necessary to discard blocks. There are two timers in the transmit and receive block. One counts two seconds of inactivity, the other two minutes of inactivity. A block *may* be discarded after two seconds of idle time, and a block *must* be discarded after two minutes of idle time. The two-second timer serves the function of quickly time-multiplexing the use of the dynamic block by many different conversations. The two-minute timer is used merely in a background manner to refresh the pool of free blocks by garbage collecting blocks associated with conversations long inactive, thus avoiding more often expensive searches for a free block at the instant a new free block is required. The two-minute timer could be made longer if desired, to allow Hosts to pause longer in a conversation without incurring some setup delay; the two-second timer value is more critical. If, as we assume, duplicate packets may arrive at an IMP up to thirty seconds after initial arrival, then a mechanism is needed to allow blocks to be created and deleted more often than every thirty seconds that protects against the same pair of Hosts using the same block twice and not catching a duplicate. The 4-bit use number allows sixteen cycles of acquisition and discard of the same block in any thirty second interval. Therefore, at least two seconds must elapse after acquisition of a block before it can be discarded. The rule that the message number must be idle for two seconds before a block can be discarded is actually stronger, but it seems a good rule to prevent thrashing and inefficiencies. The two minute timer serves to keep everything in synchronization in steady state (actually the timer runs for two minutes on the transmit side, and for slightly longer on the receive side, to prevent races).

The best policy to follow for choosing when to delete blocks seems to be for an IMP to attempt to find deletable blocks (either transmit or receive) when its free block list goes below some clip, say 10% of the total pool. When this happens, if the program can locate a transmit block that can be deleted, it sends out a "reset" message to the destination, which causes the destination to discard its block and to send a "reset reply" back to the source which causes the source to discard its block. If the program finds a receive block to delete, it sends a "reset request" message to the source which then follows the above protocol for performing a reset. On all these messages, the block number and use number provide a duplicate detection facility, since a given block with a particular use number can only be reset once.

At this point, it is worth noting the duplicate detection mechanism applied to the "get a block" and "got a block" messages. The "get a block" carries no identifying information other than the addresses of the source and destination and the source block number and use number. If a duplicate arrives during the conversation it initiated, it can be detected; likewise, if it arrives during any other later conversation between those two Hosts it can be detected. The only problem arises if the "get a block" duplicate arrives at the destination when no block exists between the two Hosts. Then the destination IMP must get a block and return a "got a block" to the source. If the source receives a "got a block" when it is not expecting one, it must send out a "reset" to clear the destination. This fixes the problem, and since the program ignores "reset" and "reset reply" messages which do not match the block and use number then active, it also takes care of the case of duplicate "got a block" messages and unwanted "reset" and "reset reply" messages generated to deal with these circumstances.

This concludes our discussion of dynamic message blocks. They do not present a large penalty in storage, delay or packet size; they are reliable because they are maintained dynamically and all communications are error controlled; they allow the available blocks to be quickly multiplexed in time among a number of conversations (albeit at reduced performance for each conversation) rather than shutting some conversations out as the available message transmission resources become fully used; and they allow separate message numbers between each Host/Host pair even when the total number of Hosts in the network is beyond a number where fixed linear tables indexed by Host would be possible. In fact, expanding on the final point, the dynamic message block mechanism permits multiple message number streams between a given Host/Host pair. The ARPA implementation currently permits two, one priority stream and one normal stream; but additional streams are easy to imagine (e.g., special "permanent" streams which cannot be preempted after two seconds, streams which follow a different ordering or retransmission criterion or even the lack of one, streams which have varying message number windows depending perhaps on stated Host needs). Altogether, the dynamic message block mechanism appears to permit very flexible and reliable operation. For further information on these techniques, see [27].

Epilogue

Since the ARPA Network was turned over to the Defense Communications Agency in mid-1975, the network has been considered primarily an operational entity with the emphasis on stability and lack of change. Nonetheless, as network usage has continued to increase over the past two years, some flaws in the network's design have become apparent and the network design has continued to evolve, albeit at a slow rate. Difficulties and developments have centered in the areas of routing in the face of congestion, accommodation of non-homogeneous network elements, and expansion of the network to be able to address larger numbers of Hosts and IMPs.

It appears that the ARPA Network will continue in its current mode of operation for at least two or three more years before its potential replacement by the second generation of packet-switching network which the U.S. government is building. During this period, no doubt there will be additional incremental changes. We expect that the ARPA Network will remain one of the more interesting developments in computer communications technology for several years to come.

Acknowledgments

Groups and individuals too numerous to mention by name have taken part in and influenced the design of the ARPA Network. We recognize their contributions; our own efforts are modest compared with the sum of the creative energy which has gone into this project. We express our thanks to R. Brooks and S. Schindler for their help with the preparation of the manuscript, and to the referees for their helpful comments.

References

[1] Auerbach Publishers Inc., Public Packet Switching Networks, Data Processing Manual No. 3-08-04 (1974).

[2] Baran, On Distributed Communications: I. Introduction to Distributed Communications Networks, Rand Corp. Memo RM-3420-PR (August 1964) 37.

[3] D.L.A. Barber (ed.), A Specification for a European Informatics Network, Co-Operation Europeenne dans le Domaine de la Recherche Scientifique et Technique (January 4, 1974).

[4] K.A. Bartlett, R.A. Scantlebury and P.T. Wilkinson, A Note on Reliable Full Duplex Transmission Over Half Duplex Lines, Commun. ACM, 12 (1969) 260–261.

[5] Specification for the Interconnection of a Host and an IMP, Bolt Beranek and Newman Report 1822, December 1974 revision.

[6] D. Belsnes, Flow Control in Packet Switching Networks, INWG Note No. 63 (October 1974).

[7] G.J. Brant and G.J. Chretien, Methods to Control and Operate a Message-Switching Network, Computer-Communications Network and Teletraffic, Polytechnic Press of the Polytechnical Institute of Brooklyn, Brooklyn, N.Y. (1972).

[8] V. Cerf and R. Kahn, A Protocol for Packet Network Inter-Communications, IEEE Trans. Communications COM-22 (1974) 637–648.

[9] V. Cerf, An Assessment of ARPANET Protocols, RFC 635, NIC 30489 (April 1974), a limited number of copies available for the cost of reproduction and handling from INWG, c/o Prof. V. Cerf, Digital Systems Laboratory, Stanford, CA. 94305.

[10] R. Crowther, F.E. Heart, A.A. McKenzie, J.M. McQuillan and D.C. Walden, Network Design Issues, BBN Report No. 2918 (November 1974) to be available from National Technical Information Service.

[11] D.W. Davies, K.A. Bartlett, R.A. Scantlebury and P.T. Wilkinson, A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals, Proc. ACM Symp. on Operating Systems Principles (October 1967).

[12] D.W. Davies and D.L.A. Barber, Communication Networks for Computers (London, John Wiley and Sons, 1973).

[13] R.F. Despres, A Packet Switching Network with Graceful Saturated Operation, Proc. First Internat. Conf. Computer Communication (October 1972) 345–351.

[14] R.W. Floyd, Algorithm 97, Shortest Path, CACM 5(6) (June 1962) 345.

[15] H. Frank, R.E. Kahn and L. Kleinrock, Computer Communications Network Design–Experience with Theory and Practice, AFIPS Conf. Proc., Vol. 40 (June 1972) 255–270; also in Networks 2 (1972) 135–166; also in Advances in Computer Communication, W.W. Chu (ed.), Artech House Inc. (1974) 254–269.

[16] M. Gerla, Deterministic and Adaptive Routing Policies in Packet-Switched Computer Networks, Proc. Third ACM Data Communications Symp. (November 1973) 23–28.

[17] F.E. Heart, R.E. Kahn, S.M. Ornstein, W.R. Crowther and D.C. Walden, The Interface Message Processor for the ARPA Computer Network, AFIPS Conf. Proceedings, Vol. 36 (June 1970) 551–567; also in Advances in Computer Communications, W.W. Chu (ed.), Artech House Inc. (1974) 300–316.

[18] F.E. Heart, S.M. Ornstein, W.R. Crowther and W.B. Barker, A New Minicomputer/Multiprocessor for the ARPA Network, AFIPS Conf. Proc., Vol. 42 (June 1973) 529–537; also in Selected Papers: Internat. Advanced Study Institute, Computer Communication Networks, R.L. Grimsdale and F.F. Kuo (eds.), University of Sussex, Brighton, England (September 1973); also in Advances in Computer Communications, W.W. Chu (ed.) Artech House Inc. (1974) 329–337.

[19] R.E. Kahn and W.R. Crowther, Flow Control in a Resource-Sharing Computer Network, Proc. Second ACM/IEEE Symp. on Problems in the Optimization of Data Communications Systems, Palo Alto, California (October 1971) 108–116; also in IEEE Trans. Communications COM-20 (1972) 539–546.

[20] L. Kleinrock, Communications Nets: Stochastic Message Flow and Delay, Dover Publications, New York (1964).

[21] L. Kleinrock, Analytic and Simulation Methods in Computer Network Design, AFIPS Conf. Proc. Vol. 36 (June 1970) 569–579.

[22] L. Kleinrock and G.L. Fultz, Adaptive Routing Techniques for Store-and-Forward Computer Communication Networks, International Computer State of the Art Report No. 6; Computer Networks, Infotech Information Ltd. Maidenhead, Berkshire, England (1971) 541–562; also in Proc. Internat. Conf. on Communications, Montreal, Quebec, Canada (June 1971) 39.1–39.8.

[23] L. Kleinrock and W. Naylor, On Measured Behavior of the ARPA Network, AFIPS Conf. Proc., Vol. 43 (May 1974) 767–780.

[24] J.M. McQuillan, W.R. Crowther, B.P. Cosell, D.C. Walden and F.E. Heart, Improvements in the Design and Performance of the ARPA Network, AFIPS Conf. Proc. Vol. 41 (December 1972) 741–754.

[25] J.M. McQuillan, Throughput in the ARPA Network – Analysis and Measurement, BBN Report 2491 (January 1973).

[26] J.M. McQuillan, Adaptive Routing Algorithms for Distributed Computer Networks, BBN Report No. 2831 (May 1974) available from the National Technical Information Service, AD781467.

[27] J.M. McQuillan, The Evolution of Message Processing Techniques in the ARPA Network, in: International Computer State of the Art Report No. 24: Network Systems and Software, Infotech, Maidenhead, England, 541–578.

[28] R.M. Metcalfe, Packet Communication, Massachusetts Institute of Technology Project MAC Report MAC TR-114 (December 1973).

[29] NAC First Semiannual Technical Report for the Project, Analysis and Optimization of Store-and-Forward Computer Networks (June 1970).

[30] NAC Second Semiannual Technical Report for the Project, Analysis and Optimization of Store-and-Forward Computer Networks (January 1971).

[31] H. Opderbeck and W. Naylor, ARPA Network Measurement Center, personal communication (November 1974).

[32] S.M. Ornstein, W.R. Crowther, M.F. Kraley, R.D. Bressler, A. Michel and F.E. Heart, Pluribus–A Reliable Multiprocessor, AFIPS Conf. Proc., Vol. 44 (May 1975) 551–559.

[33] L. Pouzin, Presentation and Major Design Aspects of the Cyclades Computer Network, Proc. Third ACM Data Communications Symp. (November 1973) 80–88.

[34] L. Pouzin, Basic Elements of a Network Data Link Control Procedure (NDLC), INWG 54, NIC 30375 (January 1974), a limited number of copies available for the cost of reproduction and handling from INWG, c/o Prof. V. Cerf, Digital Systems Laboratory, Stanford, CA. 94305.

[35] R.D. Rosner, A Digital Data Network Concept for the Defense Communications System, Proc. National Telecommunications Conf., Atlanta (November 1973) 22C1-6.

[36] R.S. Tomlinson, Selecting Sequence Numbers, INWG–Protocol Note 2 (August 1974), a limited number of copies available for the cost of reproduction and handling from INWG, c/o Prof. V. Cerf, Digital Systems Laboratory, Stanford, CA. 94305.