Local area networks
and open systems interconnection
: final report /J.F. Hayes.

Government of Canada
Department of Communications

DOC CONTRACTOR REPORT

DOC-CR-CS-(YEAR)-(Serial #)
1982 0031

DEPARTMENT OF COMMUNICATIONS - OTTAWA - CANADA

COMMUNICATION SYSTEMS RESEARCH AND DEVELOPMENT

TITLE: 2 Local Area Networks and Open Systems Interconnection : final report

AUTHOR(S): 1 Jeremiah F. Hayes

ISSUED BY CONTRACTOR AS REPORT NO: 82-3

CONTRACTOR: McGill University
Industrial Research

DEPARTMENT OF SUPPLY AND SERVICES CONTRACT NO: 1450 36100-1-0166

DOC SCIENTIFIC AUTHORITY: W.A. McCrum

CLASSIFICATION: Unclassified

This report presents the views of the author(s). Publication of this report does not constitute DOC approval of the report's findings or conclusions. This report is available outside the Department by special arrangement.

DATE:

FINAL REPORT

LOCAL AREA NETWORKS AND OPEN SYSTEMS INTERCONNECTION

Principal Investigator: J.F. Hayes

Department of Electrical Engineering
McGill University
Montreal, Canada.

March, 1982.

FINAL REPORT

LOCAL AREA NETWORKS AND OPEN SYSTEMS INTERCONNECTION

I. Introduction

Local Area Networks (LAN) are a class of data networks having limited geographical areas, usually within a kilometer. Networks confined to a single office building, shopping center or university campus are prime examples of LAN's. The emergence of LAN's is part of the general growth of computer and digital technology, however the introduction of Office Automation and Distributed Processing Systems has furnished additional impetus. In both of these applications LAN techniques play a significant role.

In the past Local Area Networks were defined in terms of geographical extent and data rate, however in view of the rapid growth of technology, more useful definition of Local Area Networks may be in terms of usage and configuration. The purpose of Local Area Networks is to provide a common communication channel among a number of users in the same limited geographical area. The emphasis is upon ease and flexibility in providing access. Due to the limited geographical area bandwidth is not the critical commodity that it is in larger networks. Thus the access to the network can be simplified at the cost of bandwidth. Data networks covering a large area require redundancy to ensure operation against failures. For example one of the features of the ARPA net[1] is two paths between source destination pairs. Because of the limited geographical extent of the typical Local Area Network, it is in something of a protected environment

and this redundancy is unnecessary. This simplifies the topology since only a single path need be provided between source-destination pairs. In current practice three configurations are prevalent - ring, bus and star (see Figure 1). (Precise definitions for topologies and access techniques will be given in the sequel). As matters now stand the ring and the bus topologies are receiving the most attention. However it appears that the star topology is well suited to the optical fiber medium. As optical fiber finds application we would expect more work on the star topology.

There are a large number of access techniques, i.e., techniques for sharing the line among users. Of these two are under active consideration for standards - Token Passing and Carrier Sense Multiple Access. Historically, Token Passing was developed in connection with the ring topology.[2] CSMA with collision detection is the latest development in random access techniques. The first of these techniques was part of the ALOHA radio system.[3] Random access became part of Local Area Network technology through the Ethernet[4] although there had been some analysis of random access in connection with local distribution.[5] The Ethernet and the associated CSMA protocol seems to be wedded to the bus topology. Nevertheless in connection with standards activity there is an application of the Token Passing technique to bus architecture. In the main body of this report we shall compare Token Passing and CSMA. We shall also make comparisons with alternative access techniques.

A third component of Local Area Networks, in addition to the topology and the access technique, is the physical medium. In our

investigation we have encountered three types: twisted pairs, coaxial cable and optical fiber. Twisted pairs operate at a rate of approximately 1 M bps. Because of the long established T 1 technology, there is a strong tendency to operate at a rate of 1.544 M bps. As we shall see presently physical considerations impel the ring topology for use in connection with twisted pairs. The access techniques are those that are appropriate to the ring topology.

At this writing coaxial cable seems to be the most widely employed medium for Local Area Networks. The speed range mentioned in connection with coaxial cable systems is in the range of 1 to 20 M bps. Existing systems seem to be at the middle to lower end of the range. The great advantage of coaxial cable is its flexibility of operation. Coaxial cable can be used in either the bus or the ring topology. Moreover coaxial cable presents no problem with regard to access technique so long as the technique is appropriate to the topology. As we shall see the great advantage of coaxial cable is the ease of connection, particularly in the bus topology.

The salient advantage of optical fiber seems to be high data rate ; speeds up to 50 M bps are obtainable without great cost. The speed of 44.736 M bps is attractive in view of the existing T3 technology. There are other advantages to optical fiber. The fiber is immune to electromagnetic interference, is chemically inert material and is an insulator for high voltage. All of these properties may be important for particular applications however they are difficult to

assess within the context of our study. The salient limitation to optical fiber lies in the lack of easy access. This property precludes a bus topology implemented with fiber and one is driven to alternative topologies.

The foregoing represents a brief overview of Local Area Networks. In the succeeding sections of this report we shall delve in some detail into several issues relevant to the design of LAN's. In section II we consider standards activities relevant to the local area networks. We review the Open Systems Interconnection concept by the International Standards organization. We also review the work of the IEEE 802 committee on a Local Area Network standards. Section III is devoted to the consideration of the ring topology. The emphasis is upon a comparison of the Token Passing technique with what is commonly called buffer insertion. The buffer insertion technique allows more than one station at a time to use the system. The comparisons in this and in the next section of the report are based on mathematical models based on Queueing Theory. The bus topology and the appertaining station access techniques are studied in section IV. Included are a comparison of CSMA and Token Passing. This section also contains a comparison of collision resolution by means of random retransmission, as in the current version of CSMA, and tree search. Section V of the report deals with the HDLC protocol. The focus in this section is upon a simulation program for HDLC which includes an emulation of the media access protocol. This simulation program is the first component of a projected large software system designed to evaluate Local Area Network protocols. The penultimate section of the

report deals with the role of optical fiber in Local Area Networks. As indicated above the particular properties of optical fiber compel network configurations which are different from those appropriate to metallic media. In a final section we draw our conclusions and chart our course for future work.

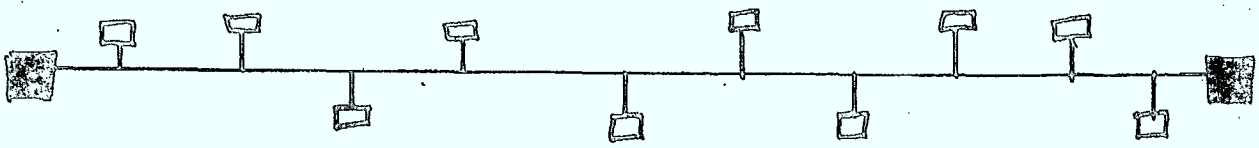


FIGURE 1.1A - BUS CONFIGURATION

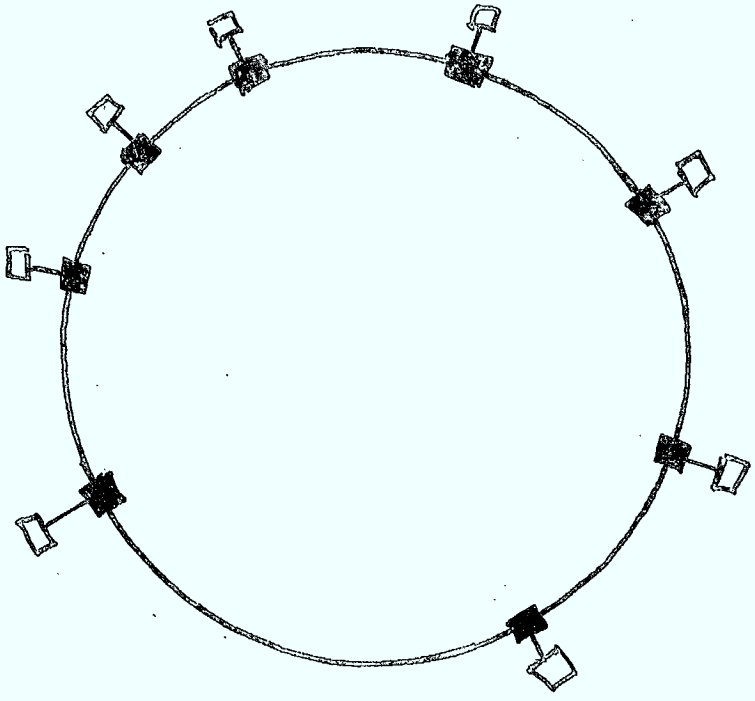


FIGURE 1.1B - RING CONFIGURATION

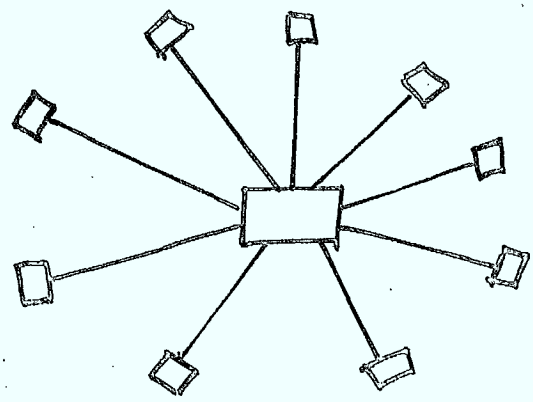


FIGURE 1.1C - STAR CONFIGURATION

II. Standards - OSI and 802

The International Standards Organization has established a general architecture, Open Systems Interconnection (OSI), for Computer Communication Networks.† The architecture segments the communication's protocols involved into seven layers and specifies the interface between layers. Within the layers standard protocols such as RS 232, X21 and the emerging 802 standard (see below), operate autonomously. The only requirement is that there be a proper interface with adjacent layers.

The seven layers of the protocol are represented on Figure 2. The lowest or the most elemental level is the Physical layer. This is the layer in which the physical link is established between a pair of terminals so that they may exchange zeroes and ones. For example, in this layer we have the CCITT V series recommendations involving the operation of voice band modems. A good deal of our work is concerned with the second layer, the Link layer. Functions related to flow on the line such as flow control and error checking reside. This layer involves sending blocks of data in frames. The line access protocols, such as Token Passing and CSMA , is a sublayer of the link layer. A higher sublayer within the link control is a flow control protocol such as HDLC . (We shall explain HDLC in detail in section V of this report). At the third layer, the Network layer, protocols dealing with several links in tandem reside. For example, for large networks, routing protocols would reside in this layer. In the context of Local Area Networks this layer would be concerned with the establishment of a virtual circuit

---

† An excellent overview of OSI is given by Zimmerman[6].

For example the recently established X25 standard would fall within the netowrk layer. The first three layers are in the province of the network but the next, the Transport layer, is where protocols for end-to-end control by the user reside. The first four levels make up what is called for obvious reasons the Transport service. The next three layers make up the Session Service Subsystem. At the Session layer sessions between users are initiated and terminated. The printing and the display of data is controlled in the Presentation layer. In the highest level, the Applications layer the control of the files and data bases is carried out. These last four levels are not of immediate concern to our work.

Also directly relevant to our work is the effort of the IEEE 802 committee to establish a standard for Local Area Networks.[7] Most of the committee seems to have been concerned with the media access technique and on the physical properties of the media. The media access techniques that are under study are Token Passing and CSMA/CD . The speeds that are supported for the two techniques are 1, 5, 10 and 20 M bps, at broadband and baseband. A subcommittee has considered performance criteria and has done some work on the comparison of the different approaches.[8] As part of this work an extensive survey of the literature has been carried out.

In connection with the physical media work has been done on obtaining standards for coaxial cable[7] and for optical fiber.. We have only recently become aware of the work on optical fiber.[9] This work has been particularly valuable to us since it reinforces conclusions we had reached through independent effort.

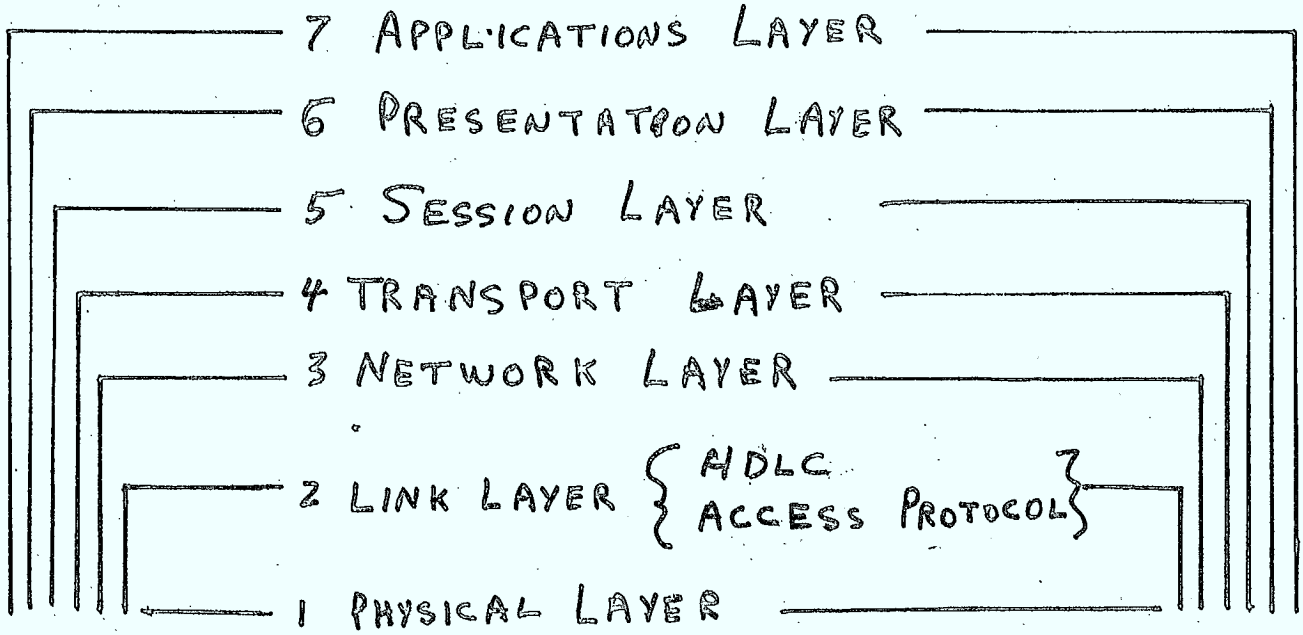


FIGURE 2.1 OPEN SYSTEMS INTERCONNECTION

III. Ring Systems

One of the two most prevalent topologies for Local Area Networks is the ring or, synonymously, the loop structure. The idea of using a ring for computer communications was first proposed by Farmer and Newhall in 1969.[2] The Farmer-Newhall loop was followed shortly by an alternative proposal by Pierce.[10,11] These two techniques form the basis of the current work on ring systems. The fundamental differences between the two systems lie in the technique for granting access to users. Farmer and Newhall originated Token Passing whereby the station possessing the token has the exclusive right to transmit on the ring to any other station. Flow on the ring is in one direction. (See Figure 3.1). In terms of the 802 standard the medium is sequential. When a station has transmitted all that it is going to transmit, an End-of-Message Character (EOM) is transmitted. The EOM is in effect the token. Upon receiving the End-of-Message character, the next station downstream assumes the right to transmit. If it has nothing the token is immediately passed on to the following station downstream. This technique was implemented using T2 technology implying a data rate of 6.312 M bps.

The protocol associated with the Farmer-Newhall system has remained very much the same over the years and is very much the same as the token passing scheme which is being considered by the 802 Committee.[12] However a number of alternatives based on the access technique used in the Pierce loop have been developed.[13-17] In the original proposal flow on the line was slotted into fixed duration frames (see Figure 3.2). The stan-

dard T1 frame was used in an early implementation of the technique.[18] At the beginning of each frame there is a single bit marker indicating whether or not the frame is empty or full. A station perceiving a frame to be empty can fill the frame with its own packets. If the station sees that a frame is full it checks the address bits that succeed the empty or full marker. If the packet is destined for the station it is removed thereby freeing the frame. Clearly there must be enough buffering in the node to allow examination of addresses before the marker bit is transmitted. Flow on the line is given priority and a message consisting of a number of packets may be interrupted in the course of transmission (see Figure 3.2 for an illustrative example). Due to this interruption each packet must be accompanied by source-destination addresses in order to sort out the flow on the line. In applications where there are a large proportion of variable length packets this could lead to a large part of the line capacity being taken up by overhead. The increase in overhead may be balanced by the fact that more than one station may access the line simultaneously.

The need for excessive address overhead is due to the fact that the protocol for sharing the line is, in queueing theory terms, preemptive, i.e., traffic already on the line can preempt the line from a local station.[19-20] The advantage of this discipline is simplicity of operation. Once a packet is on the line it suffers no random delay, only a constant processing delay in each of the stations on its route. For multiple packet messages there is a random reassembly delay.

An alternative to the Pierce loop protocol is the so-called Buffer Insertion protocol.[14] The essential feature of this protocol is that it follows a non-preemptive discipline in which traffic already on the line has priority but may not interrupt the transmission of messages already in progress (see Figure 3.3). Since a destination is assured of receiving a complete message without interruption only one address need accompany a message. There is, however, an attendant increase in complexity: once a message is on the line its delay is random since it must wait for the termination of message transmission at intervening stations. Moreover, buffers allocated to line traffic can overflow unless a limit is placed on the maximum duration of messages. As in the case of the Pierce Loop line utilization is improved since more than one terminal may transmit at a time.

Since its inception questions concerning the reliability of ring systems have been raised. A break in continuity would render the system inoperative. This question has been addressed from two points-of-view. In virtually every implementation of ring networks the nodes have been fail-safe in that, in the event of loss of power, electrical continuity is preserved. The second aspect of the reliability considerations is the provision of redundant ring structures which would avert system failures.[21-22] We shall return again to the consideration of reliability when we consider optical fiber.

In both of the seminal works on the ring topology by Farmer and Newhall and by Pierce a hierarchical ring structure was proposed. This concept is perhaps most graphically illustrated in Figure 3.4 where a hierarchy local, regional and national rings are shown.[11] In the original

proposed by Pierce flow on each of the rings have the same frame structure, with different speeds allowed. The access technique from one ring to another is the same preemptive technique discussed above.

Comparison Token Passing and Buffer Insertion

In this section we shall compare the Token Passing Strategy with that of Buffer Insertion on the basis of the average delay of messages. Unfortunately, at this writing analytical models for a comparison based on higher moments of delay are much less tractable and further study is required.

In this and in all subsequent studies in this report we shall assume that messages arrive at a Poisson rate with an average rate of λ to each of the N stations sharing the common line. We shall assume that the durations of the messages follow an arbitrary distribution with mean $\bar{m}$ and mean square $\overline{m^2}$.

Now in order for messages to be transmitted without ambiguity to the proper destination certain overhead information must accompany the information bits. We recognize that the overhead for addressing and for message delimiting is contained in existing protocols such as HDLC. However we shall include it in our analysis in order to account for the required overhead in the most efficient possible manner. If successive messages arriving at a terminal have independent destinations then on the order of $\log_2 N$ address bits must accompany each message. Furthermore,

since messages have an arbitrary length distribution, and since there are no restrictions on the patterns of the actual data, some method is required to indicate the end of a message and the beginning of a new message. There are currently two techniques for doing this - flags and blocks. In the flag technique the end of the message is signalled by a unique bit pattern; e.g., 1000. If the true data sequence replicates the flag a bit is stuffed into the sequence so that 10010 is transmitted. The augmentation of a message that is due to flag and stuff bits has been analyzed for general distributions of message lengths. For the case of geometrically distributed message lengths it can be shown that the average duration of the augmented message is[23,24]

$$\bar{m}' = \frac{1}{1-q}\left[1 + \left(\frac{1}{2}\right)^{F-1} q^{F-2}\right] + F \qquad (3.1a)$$

where F is the duration of the flag and $\frac{1}{1-q}$ is the average duration of the message.

Similarly for geometrically distributed messages it can be shown that mean square value of the augmented message is given by

$$\overline{m'^2} = \frac{1+q+\left(\frac{q}{2}\right)^{2F-3} + 4\left(\frac{q}{2}\right)^{F-1}}{(1-q)^2} + \frac{\left(\frac{q}{2}\right)^{F-2}\left[F - \frac{1}{2}\right]}{1-q}$$

$$+ 2(F+1)\left(\frac{1 + \left(\frac{1}{2}\right)^{F-1} q^{F-2}}{1-q}\right) + (F+1)^2 \qquad (3.1b)$$

It has been shown that the optimum duration of the flag in terms of minimizing the average duration of the augmented message is given by

$$F = [\log_2 \bar{m}] + 1 \tag{3.1c}$$

In the blocking technique a message is segmented into fixed length blocks. The first two bits in a block indicate whether or not it is the first of a message and whether or not it is full. In a block that is not completely filled there must be delimiter bits indicating which of the bits are true information bits and which are stuff bits. Again, the length of the augmented message has been worked out for arbitrary message length distributions. The probability of a message being represented by k blocks each consisting of B bits is

$$Q_k \triangleq \Pr[k \text{ blocks}] = \Pr[(k-1)B - [\log_2 B]^+ < B \leqslant kB - [\log_2 B]^+] \tag{3.2a}$$

For geometrically distributed random variables it is a simple matter to work out the mean and the mean square of the augmented message. It has been shown that, for geometrically distributed messages, the optimum block size is given by B which satisfies

$$p^B + p^{B+1} \leqslant 1 < p^B + p^{B-1} \tag{3.2b}$$

In addition to overhead for addressing and message delimiting, which is essentially the same for both Token Passing and Buffer Insertion, there is overhead that is peculiar to each technique. Thus a certain number of bits are required to pass the token. A reasonable way to do this is to repeat the flag at the end of the last flag followed by an address indicating the next station receiving exclusive line access. An alterna-

tive in the case of block encoding is to follow the final block of a message with a non-full block which is not the first block of a message. In this non-empty block is the address of the next station to be granted access.

In the case of the Insertion Buffer, overhead assumes a slightly different form. When a message enters a station there must be a delay to examine the address to determine whether the message should be passed on or not. This delay is in addition to any traffic induced delay. In Token Passing this delay is not required. In compensation the Buffer Insertion technique does not require a token to be passed.

In calculating message delay for Token Passing we shall rely upon an analysis of a polling model by Hashida.[25] This model assumes infinite buffers and exhaustive service; i.e., a station possessing the token transmits all the messages in its buffer before passing the token. There are a number of alternative models* which are not quite appropriate for comparison with the buffer insertion model we have available to us.

Polling models are characterized by a periodically available server which in this case corresponds to exclusive access to the line. Since message arrival to each of the stations is Poisson, dependencies are introduced between the queues at each of the terminals. The longer the token resides at a station the more messages will be present at another

---

*An addendum to this report is a survey of local distribution techniques by the author. In this survey is an exhaustive discussion of polling models.

station when it receives the token. Hashida has accounted for these dependencies. He has shown that the average delay of a message is

$$\bar{D} = \frac{NW(1-\rho)}{2(1-N\rho)} + \frac{N \lambda \bar{m}^2}{2(1-N\rho)} \qquad (3.3)$$

where $\rho = \lambda \bar{m}$ and W, the walktime, is the amount of time that is required to pass the token.

In calculating the performance of the Buffer Insertion by Bux and Schlatter,[2,6] a simplified derivation of their result is given here. This derivation is illustrated by means of the three node ring shown on Figure 3.5. We assume that messages destined for node 3 exclusively arrive at stations 1 and 2 with rates λ_1 and λ_2 respectively. Further it is assumed that station 3 receives but does not generate traffic. In each of the nodes there are two buffers, one for locally generated traffic and one for traffic already on the line. Previous analyses of this kind of network were seriously flawed. For example if $\lambda_2 = 0$ then messages from terminal 1 suffer the same kind of delay at terminal 2 as they do at terminal 1 . However arrival at terminal 1 is Poisson and can be handled as an M/G/1 whereas at terminal 2 messages from terminal 1 can be handled on the fly with a constant processing delay. Further, in previous analyses there was an unjustified assumption of independence between stations 1 and 2 .

An analysis based on a repeated application of Little's theorem[27] leads to the correct value of average message delay. The delay of messages in an M/G/1 queue* is

---

*Poisson arrival rate, general service time and a single server.

$$G(\lambda) = \frac{\lambda \overline{m^2}}{2(1-\rho)} + \bar{m} \tag{3.4}$$

where $\rho \triangleq \lambda \bar{m}$. From Little's theorem the average number of messages residing in such a system is $Q(\lambda) = \lambda G(\lambda)$. Now with respect to stations 1 and 2, notice that the aggregate of stations 1 and 2 as well as station 1 alone behaves like an M/G/1 queue with arrival rate $\lambda_1 + \lambda_2$ and λ_1 , respectively, since there is no lost work. The average message delay in station 2, denoted by D_2 , is not given by the M/G/1 formula to the stream nature of the line traffic. However by using Little's theorem we have for the average number of terminals in both stations

$$(\lambda_1 + \lambda_2)\, G(\lambda_1 + \lambda_2) = \lambda_1\, G(\lambda_1) + (\lambda_1 + \lambda_2)\, D_2$$

and

$$D_2 = G(\lambda_1 + \lambda_2) - \frac{\lambda_1}{\lambda_1 + \lambda_2} G(\lambda_2) \tag{3.5}$$

In node 2 we may distinguish two kinds of delay - locally generated messages and line traffic. If the line traffic has non-preemptive priority, the delay suffered by the local traffic is given by the standard formula

$$D_{2A} = \frac{(\lambda_1 + \lambda_2)\, \overline{m^2}}{2(1-\rho_1-\rho_2)(1-\rho_1)} + \bar{m} \tag{3.6}$$

where $\rho_1 = \lambda_1 \bar{m}$ and $\rho_2 = \lambda_2 \bar{m}$.

If D_{2B} is the average delay suffered by line traffic then from Little's theorem we have for the average number of messages in station 2

$$(\lambda_1 + \lambda_2)\, D_2 = \lambda_1 D_{2A} + \lambda_2 D_{2B}$$

or

$$D_{2B} = \left(\frac{\lambda_1 + \lambda_2}{\lambda_2}\right) D_2 - \frac{\lambda_1}{\lambda_2} D_{2A} \tag{3.7}$$

From equations (3.4) - (3.7) the average in transit queueing delay of a message at station 2 can be calculated.

The result generalizes easily for the case of a data collection ring of N stations where all messages have the same destination. Consider the transient delay for a message passing through the ith station. The first $i-1$ stations act, for all intents and purposes, like a single M/G/1 queue. The transit delay for the ith station is given by equations (3.4) - (3-7) with λ_1 replaced by $\sum_{i=1}^{i-1}$ and λ_2 by λ_i. Here λ_i is the arrival rate to the ith station.

When there is no single source or destination for traffic an exact analysis is not possible. However, Bux and Schlatter have demonstrated by means of simulation that a reasonable approximation works. In the approximation λ_1 in equations (3.4) - (3.7) is replaced by that portion of the line traffic which passes through the station and λ_2 becomes the locally generated traffic. This is illustrated by means of two examples. Suppose that all traffic goes to an adjacent terminal. In this case $\lambda_1 = 0$ in the foregoing equations and from the foregoing equations the only delay suffered by a message is that of an M/G/1 queue. The second example is that of completely symmetric traffic where each terminal transmits

equally to all other terminals. If $\lambda_i = \lambda$; i = 1, 2, ... N , $\lambda_1 = \sum_{i=1}^{N-2} \frac{i\,\lambda}{N-1} = (\frac{N-2}{2})\lambda$. On the average a message passes through $\frac{N-2}{2}$ stations. The average delay due to queueing processes is the transient delay computed from equations (3.4) to (3.7) with $\lambda_1 = (\frac{N-2}{2})\lambda$ and $\lambda_2 = \lambda$ multiplied by $\frac{N-2}{2}$. To this must be added the average delay upon entering the line given by (3.6) and the processing delays at each of (N-2)/2 nodes, on the average.

A comparison of the mean delay versus system load of buffer insertion rings and token rings is shown on Figures 3.6 - 3.10. In these curves, a message is assumed to consist of data bits and all the necessary address and control information required by higher level protocols. The length of a message is assumed to be geometrically distributed with mean TMEAN = 1/(1-q) . Assuming flag bits are used by the physical layer protocol to delimit individual messages, the message length is augmented by the flag bits, the stuff bits needed to maintain the uniqueness of the flag, and one indicator bit necessary to distinguish between the end of an individual packet and the end of the last of a station's packets.

As mentioned above, Hashida's[25] analysis is used here to find the mean delay of a token ring or polling system where all stations have the same Poisson arrival rate, λ_L , and service at each station is exhaustive; i.e., all packets which are present when the station is polled are transmitted, as are all packets which may be generated during the service. Contrast this situation to Kaye's work,[34] where the buffers at each station

hold at most one message. Hashida's analysis assumes that all packets are destined to a central processor, but one may equally well assume that packets are either broadcast to all stations on the ring, with a packet's destination address determining the correct recipient, or relayed station by station to the proper destination. If, however, one assumes that messages are relayed, then, in lieu of modifying the analysis to account for processing delays at each station en route, the message length distribution should be appropriately adjusted. For the purposes of this comparison, assume that packets are broadcast to all stations. Given this interpretation, then, the walk-time between the polling of adjacent stations is equivalent to the transmission time of a control packet (the token) from the station currently transmitting to the next station. The walk-time is therefore assumed to be a constant, arbitrarily assumed equal to the transmission time of 40 bits.

There is no possible ambiguity in the mean delay analysis of buffer insertion rings: the protocol stipulates that packets are relayed station by station from source to destination. Each hop incurs a processing delay, since, at the very least, a station must inspect each packet's destination address to determine if the packet is to be transmitted on to the next station. Possible inspection of, for example, the source address (to prevent packets from perpetually circling the ring) and other control parameters may further increase this delay. The processing delay per packet per station is thus assumed to be the transmission time of 40 bits.

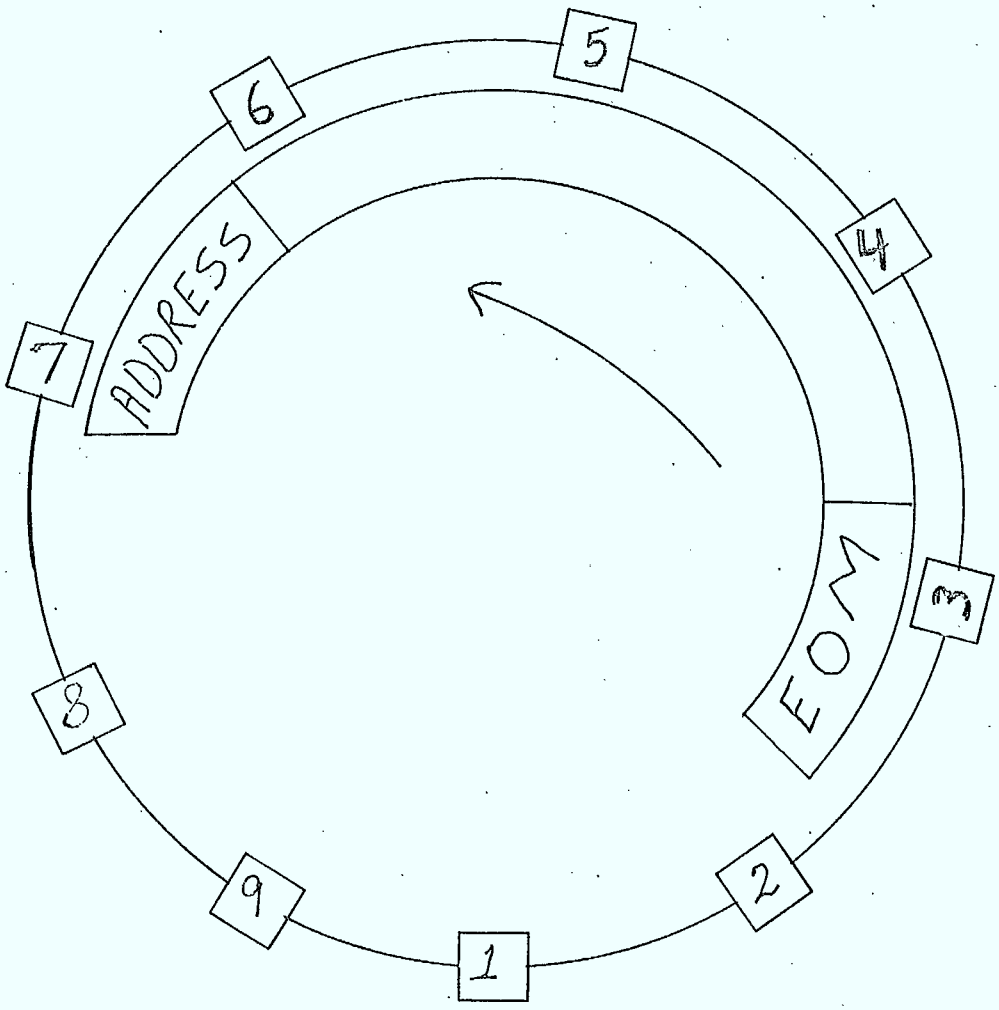
The analysis of buffer insertion ring systems requires explicit knowledge of the ring traffic patterns as defined by the routing probabilities $p(i, j)$ that a packet originating from station i is destined to station j. To compare the delay performance of buffer insertion rings to that of token passing systems, assume that traffic in the buffer insertion ring is symmetric; i.e., the Poisson arrival rate of messages to each station is λ_L and each packet's destination is drawn with equal probability from the stations on the ring, excluding the sender.

The graphs of buffer insertion ring and token ring mean delays are plotted against the system load, defined as the product of the number of stations, the message arrival rate per station, and the mean message duration. The overhead required by the physical level protocol - the flag bits, stuff bits, and indicator bit - is not included. The system load or traffic intensity thus reflects the ratio of the rate at which valid data enters the system to the system's transmission rate.

The most remarkable feature of these curves, then is the fact that buffer insertion rings are able to support loads greater than unity with tolerable mean packet delays. This result is due to the decentralized nature of the buffer insertion ring protocol: according to the ring priority scheme considered here, a station transmits its own generated packets whenever it has no ring traffic to relay and need not wait to be polled. Furthermore, packets are relayed around the ring no farther than their destination. In the case of symmetric traffic, (see Figures 3.6 - 3.9) packets travel on average only halfway around the ring and so the

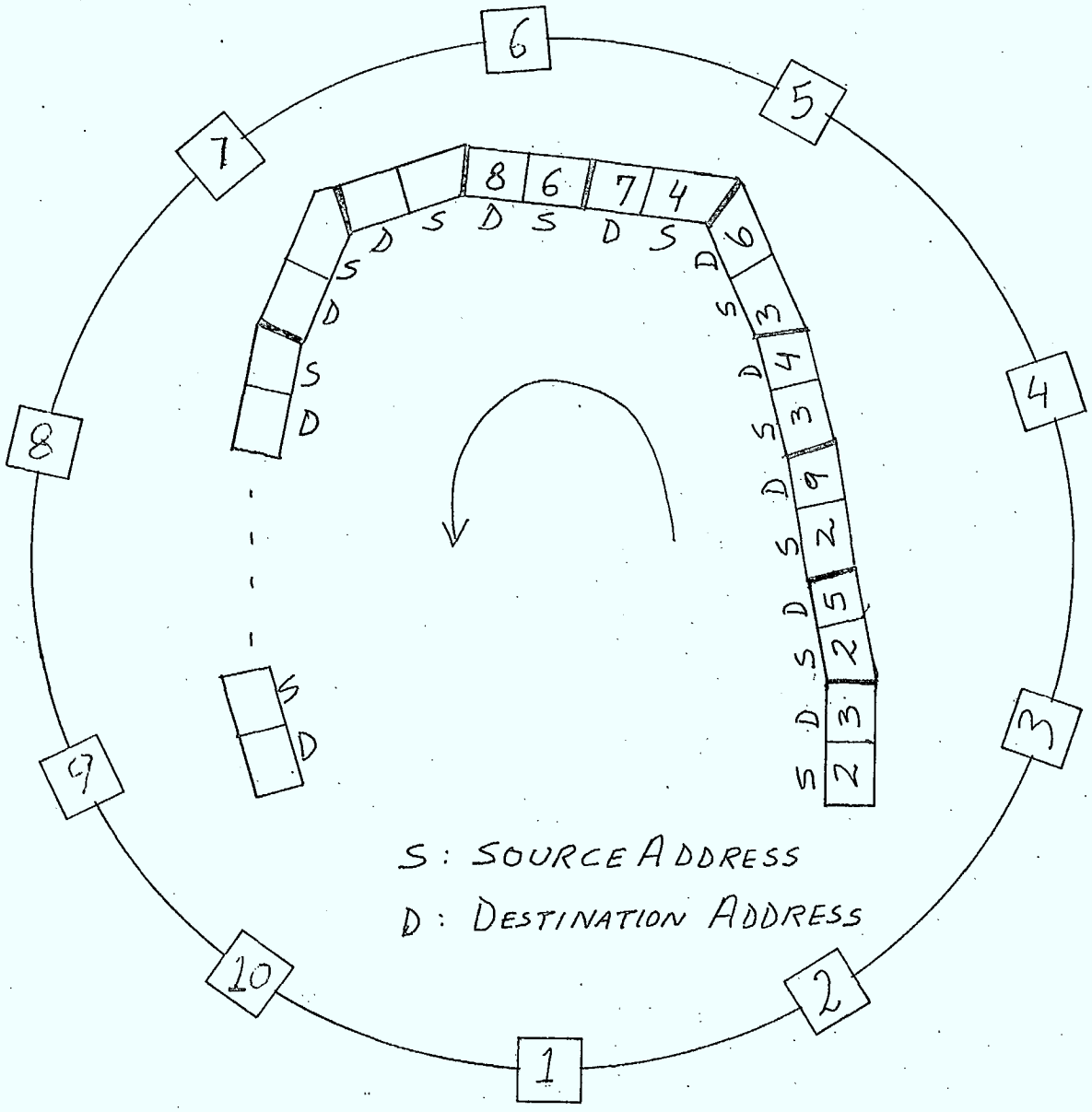
system is able to provide a data rate effectively approaching twice the actual transmission rate. Such behavior is taken to the extreme in the case of nearest-neighbor traffic, where a station generates messages destined only to its neighbor one hop away. As the two graphs of buffer insertion ring performance with nearest-neighbor traffic show, (see Figure 3.10) the system supports data rates approaching N times the actual transmission rate, where N is the number of stations. Effectively, each station has a dedicated line to its neighbor.

As a final note, one would expect that, since a processing delay is incurred at each station, the delay performance of a buffer insertion ring relative to a token ring would degrade as the number of stations increase. That behavior is in fact not observed here since, at low system loads, token rings suffer an equivalent overhead as the token is passed around the ring to those stations with messages to transmit. At higher system loads, a packet's mean delay in either type of system is dominated by its waiting time in a station or stations' queue(s). Of course, a buffer insertion ring in which the majority of each station's traffic is destined to the most relatively remote station will have a significant overhead penalty due to processing delays, but then so will a token ring if one assumes that packets are not broadcast to all stations, but rather relayed station by station. In that case, token ring mean delays would be even greater than the results reported here.



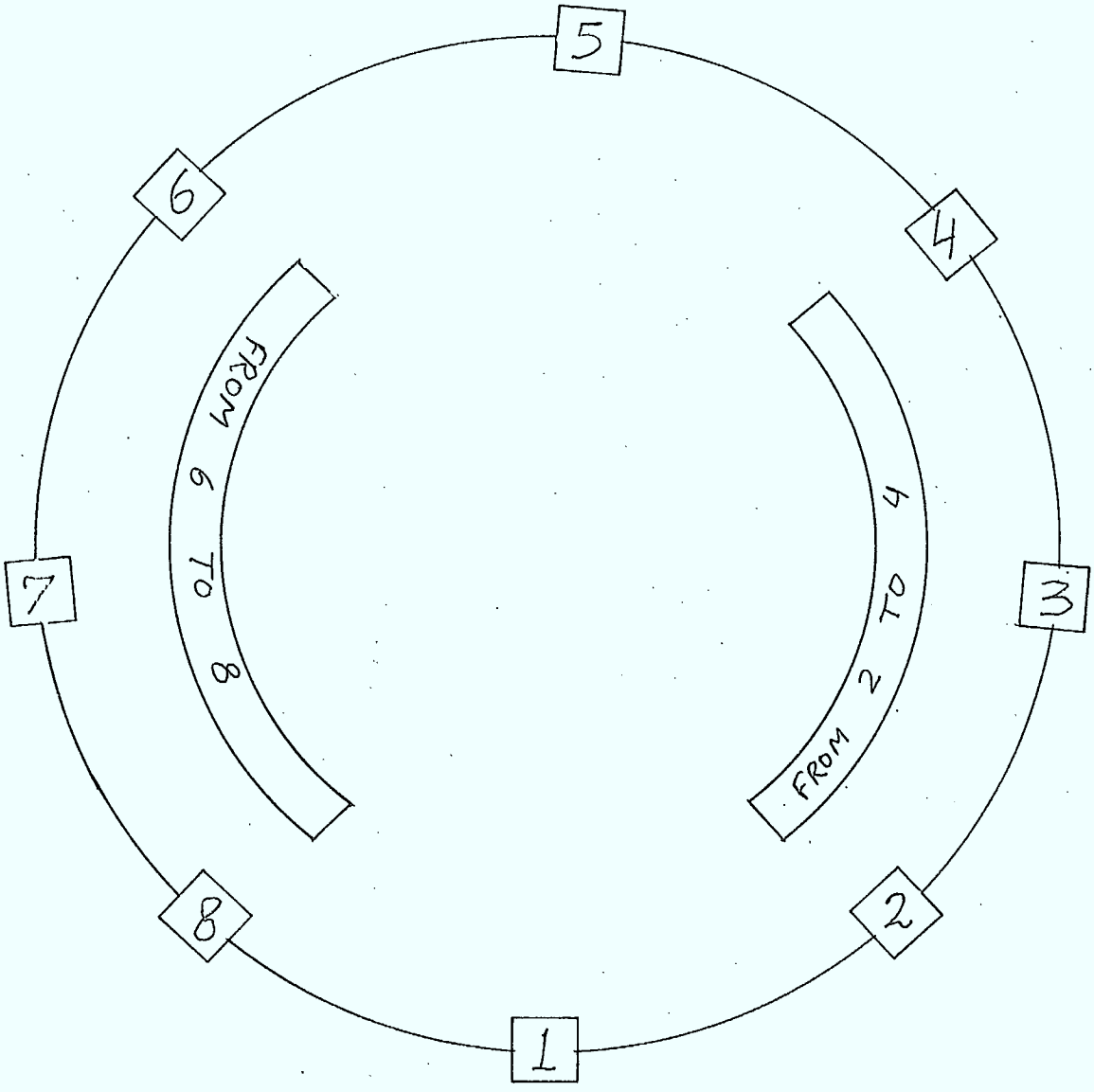
FARMER-NEWHALL LOOP

FIGURE 3.1



PIERCE LOOP

FIGURE 3.2



BUFFER INSERTION

FIGURE 3.3

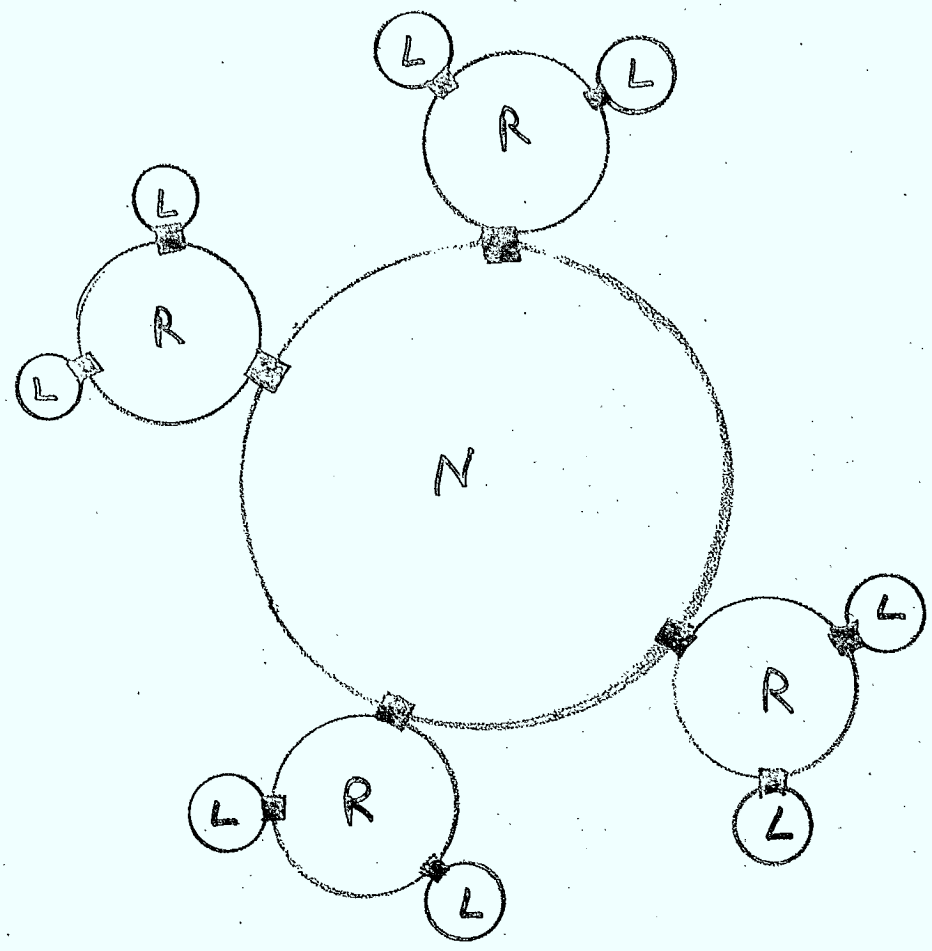


FIGURE 3.4 HIERARCHY OF RINGS

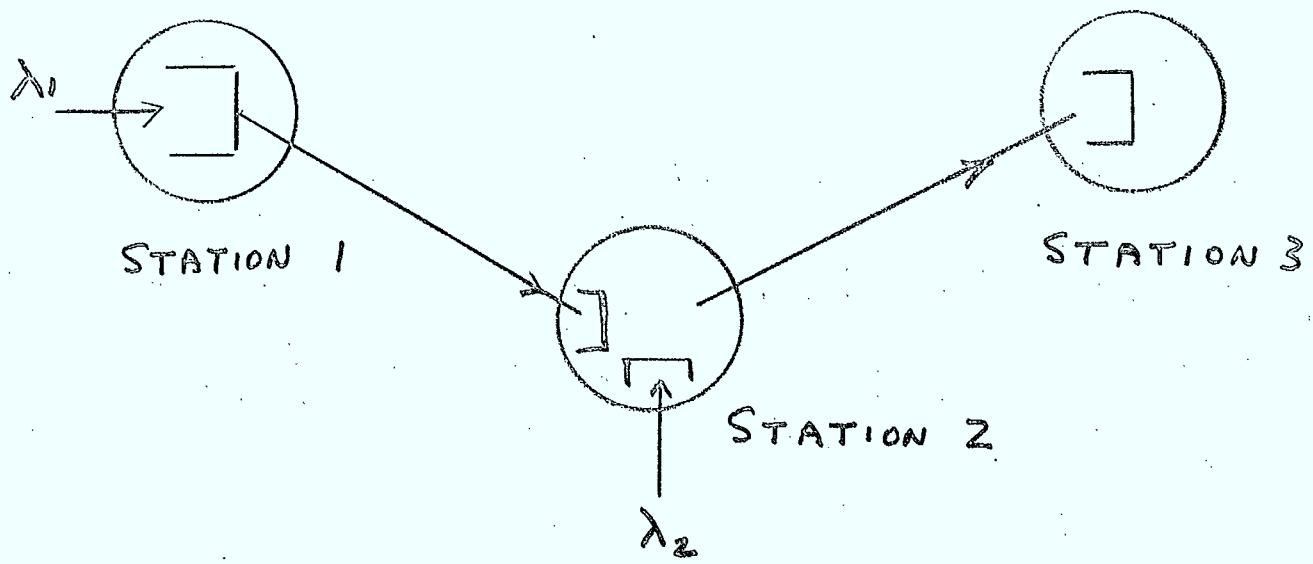


FIGURE 3.5 BUFFER INSERTION MODEL

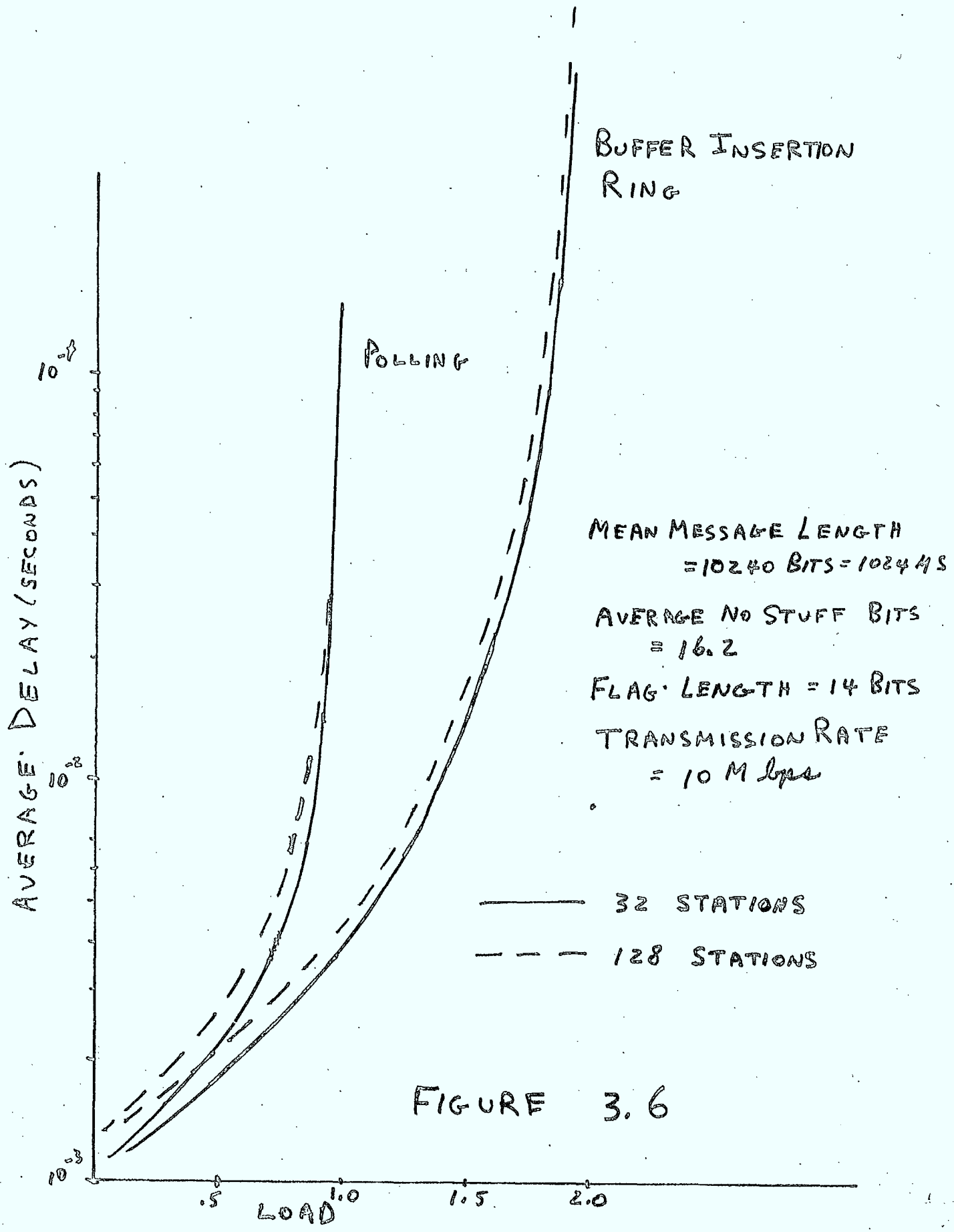

BUFFER INSERTION RING
POLLING
MEAN MESSAGE LENGTH
=1024 BITS = 1024 μS
AVERAGE NO STUFF BITS
= 16.2
FLAG LENGTH = 14 BITS
TRANSMISSION RATE
= 10 M bps
32 STATIONS
128 STATIONS
AVERAGE DELAY (SECONDS)
10^{-1}
10^{-2}
10^{-3}
.5
1.0
1.5
2.0
LOAD

FIGURE 3.6

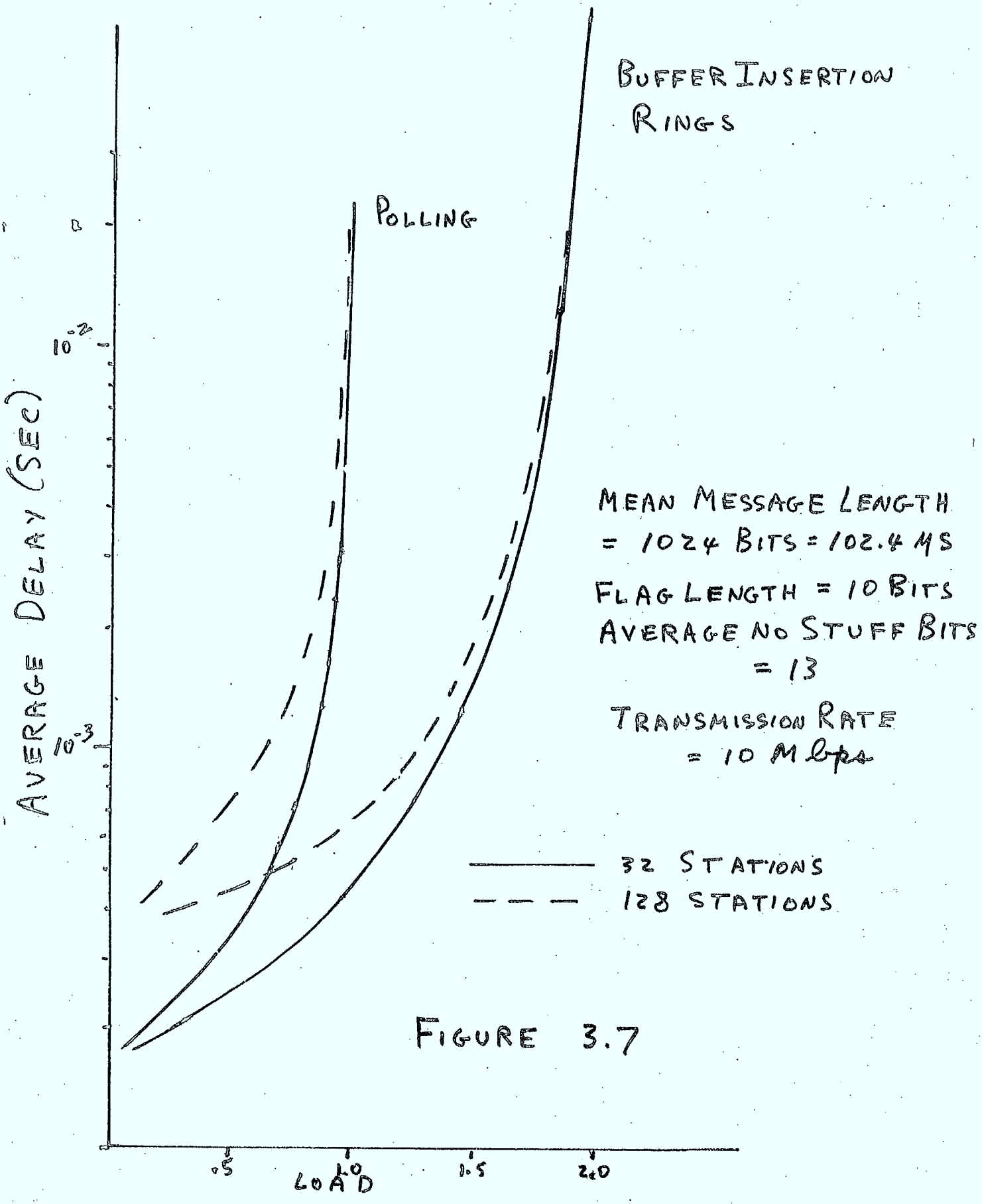
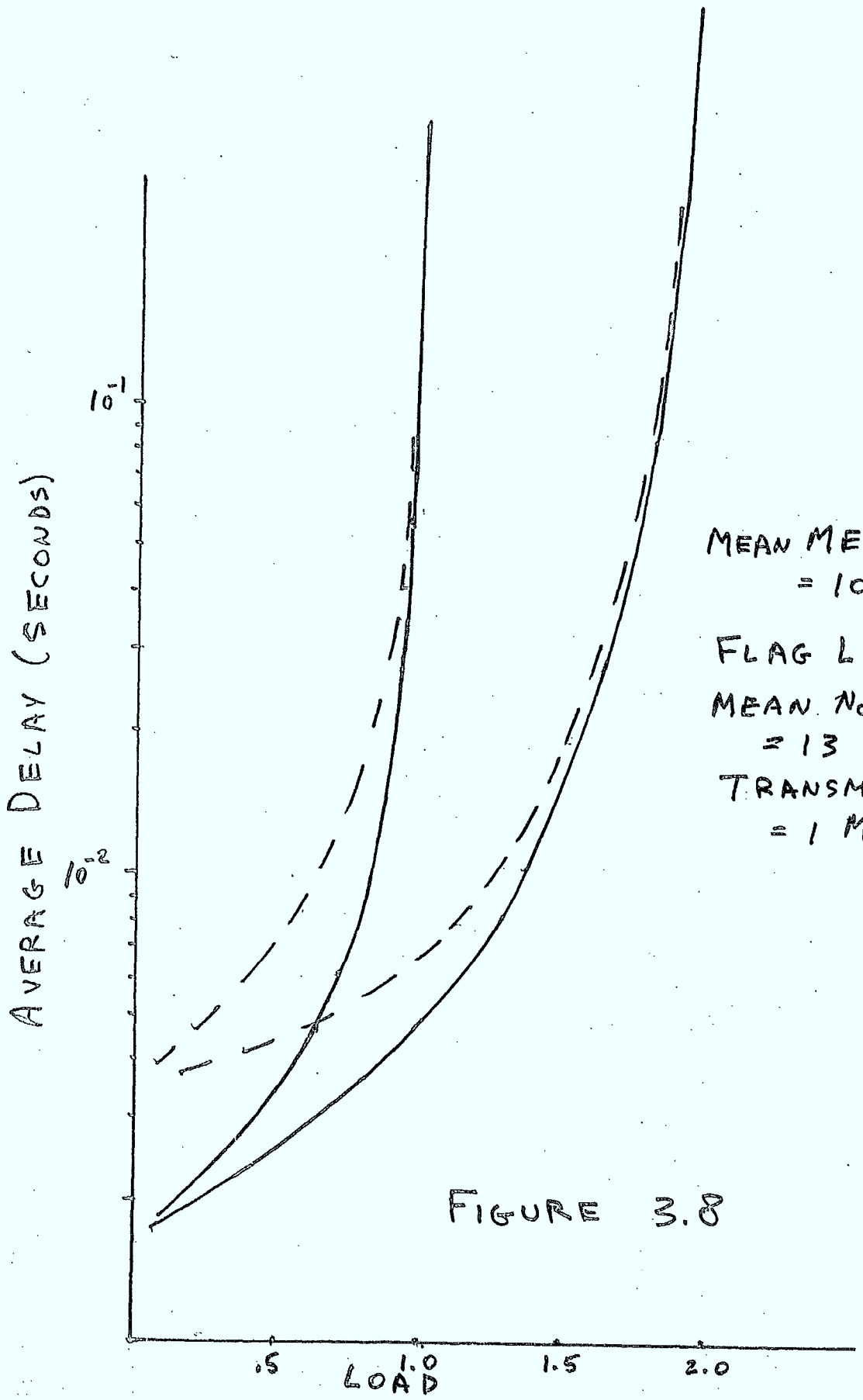

BUFFER INSERTION RINGS
POLLING
AVERAGE DELAY (SEC)
10^{-2}
10^{-3}
.5
1.0
1.5
2.0
LOAD
MEAN MESSAGE LENGTH = 1024 BITS = 102.4 MS
FLAG LENGTH = 10 BITS
AVERAGE NO STUFF BITS = 13
TRANSMISSION RATE = 10 Mbps
32 STATIONS
128 STATIONS

FIGURE 3.7



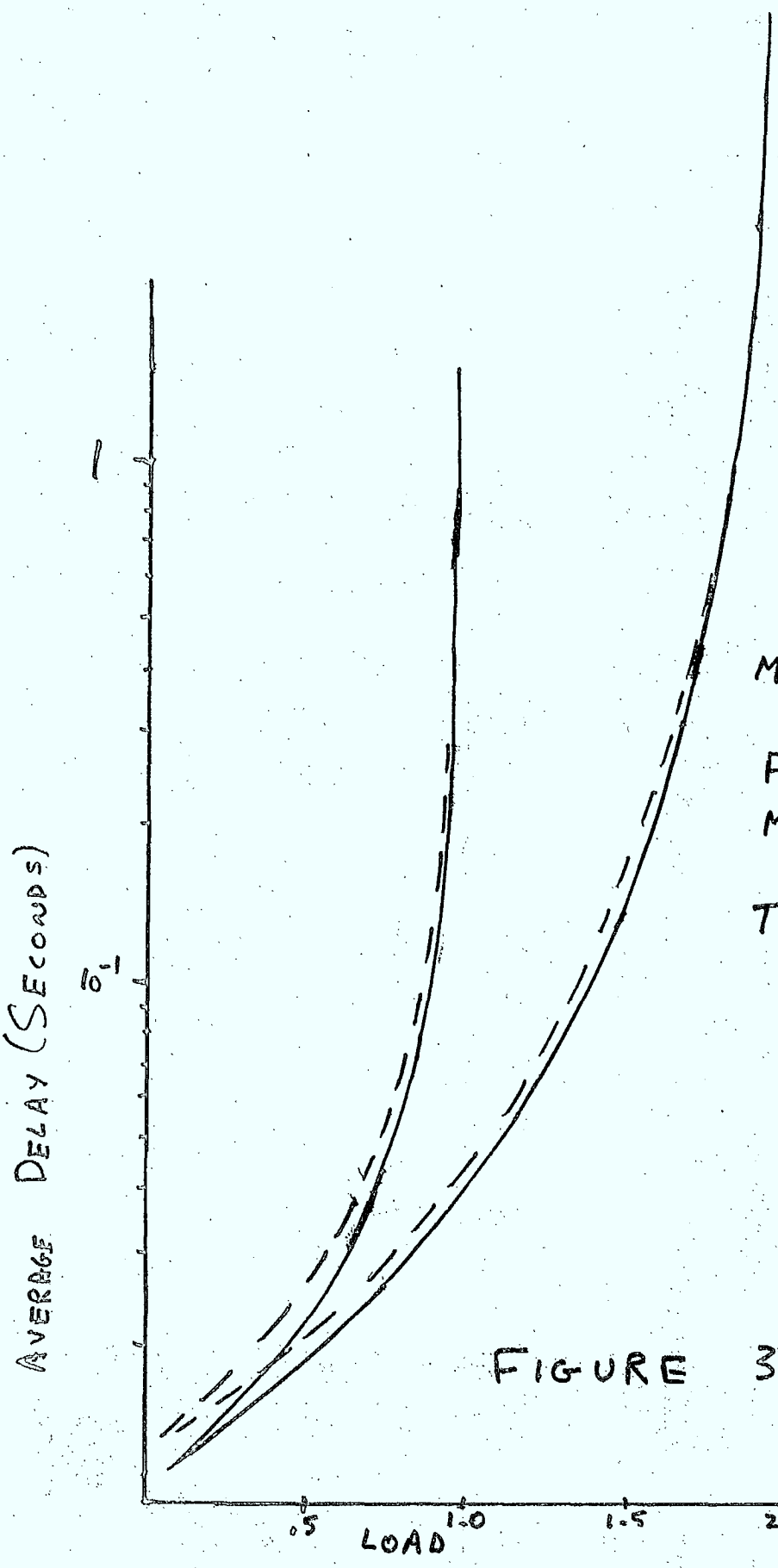
MEAN MESSAGE LENGTH
= 1024 BITS

FLAG LENGTH = 10 BITS

MEAN NO STUFF BITS
= 13

TRANSMISSION RATE
= 1 Mbps

FIGURE 3.8

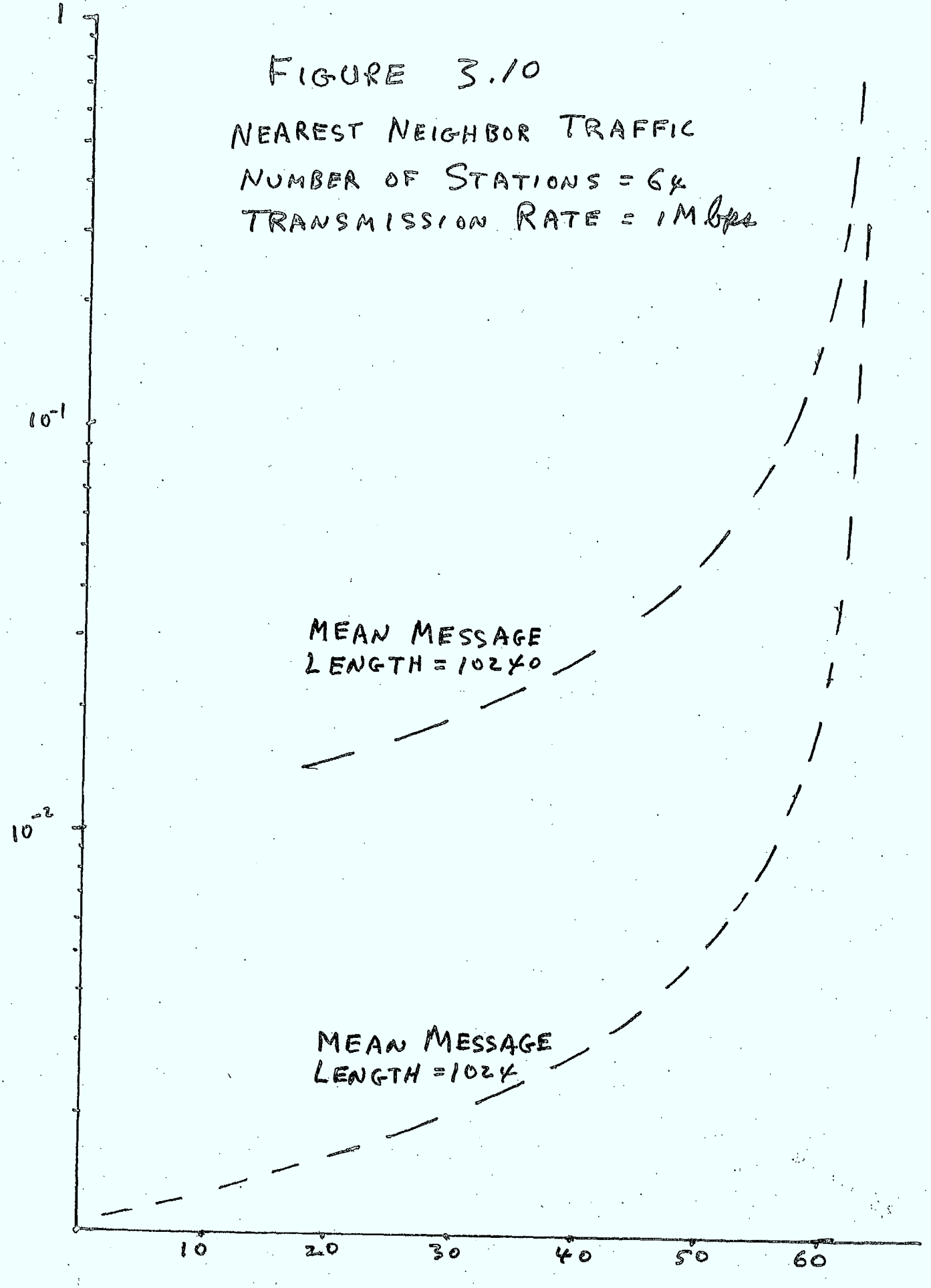


MEAN MESSAGE LENGTH
= 1024 BITS
FLAG LENGTH = 14 BITS
MEAN No STUFF BITS
= 16.2
TRANSMISSION RATE
= 1 Mbps

FIGURE 3.9

FIGURE 3.10

NEAREST NEIGHBOR TRAFFIC
NUMBER OF STATIONS = 64
TRANSMISSION RATE = 1 Mbps



IV. Bus Systems

The current alternative to the ring architecture is the bus shown in Figure 4.1. The essential ingredient is that the medium is of the broadcast type as identified by the 802 committee. Signals transmitted on the line are received by all terminals at times differing only by propagation times on the medium. It seems that coaxial cable is the idea medium for use in the bus architecture. Stations can be bridged onto the line without disturbing the flow or affecting transmission. This does not seem to be the case for the other media. As we shall see in section VI in our discussion of optical fiber, this has important implications in connection with topology.

Two protocols have been proposed for use on the bus architecture - Token Passing and Carrier Sense Multiple Access (CSMA) . The Token Passing Protocol operates in very much the same fashion as in ring systems. A single station at a time has exclusive access to the medium. At the end of its transmission a sequence indicating the end of transmission and the address of the next terminal are appended. The techniques for doing this, flags or blocks, are the same as in ring systems. The performance analysis of Token Passing for the bus architecture is the same as in the ring system. The important parameters are the mean and the mean square message length, the message arrival rate, the number of stations and the time required to pass a token from one station to another (see equation (3.3)).

As we have seen in connection with token passing in ring systems, Token Passing has an overhead which is proportional to the number of stations in the system. Furthermore for the bus architecture, the startup sequence can be fairly complicated since unlike ring systems there is no natural ordering among the terminals. An alternative protocol which has neither of these drawbacks (but drawbacks of its own) is Carrier Sense Multiple Access with Collision Detection CSMA/CD .

CSMA/CD is the latest in a sequence of random access techniques which began with the ALOHA radio system.[28] Although random access techniques have made advances there is a common behavioral pattern which is manifest in all classes of random access systems. This can be illustrated by the behavior of ALOHA in its simplest form - unslotted. In unslotted ALOHA a message is transmitted as soon as it is received by a station. If the message is received by the destination on the common channel some form of acknowledgement is returned to the transmitting terminal. When two or more stations transmit messages simultaneously collisions will occur. A station that has been involved in a collision will retransmit after a randomly selected timeout interval. Traffic on the line will then consist of new attempts and retransmissions. A classical analysis, begins with the assumption that the total flow on the line is Poisson. Under the additional assumption that the duration of messages is constant this leads to a simple relationship between newly generated load at all stations ρ and combined load new and retransmitted, R . We have

$$\rho = R e^{-R} \tag{4.1}$$

As in the previous section of this report the load is the product of message arrival rate and message duration. Equation (4.1) is plotted on Figure 4.1. Notice that ρ is the independent variable since it is the load that is offered to the system. We see that for very light loading there is a linear relationship between ρ and R since the number of retransmission is negligible. However as the offered load increases there is a point of saturation at an offered load of 18% . In fact as shown on Figure 4.1 there is instability in that there are two values of total traffic for each offered load.[29-30] At the higher values of total traffic transmission is dominated by retransmissions. Although the model presented here is quite simple and the Poisson assumption on line flow is a bit dubious, a large number of analyses and simulations have verified the result.

By the introduction of slotting, i.e., only allowing transmission at periodically spaced points in time, it has been shown that this capacity is doubled. The performance is as shown on Figure 4.1. Notice that the same basic instability is present.

Random access systems have developed along the line of utilizing a sensing capability at the station.[32] Thus in Carrier Sense Multiple Access a station senses the line before transmission. In the P-persistent implementation transmission takes place at the end of the current transmission with probability P . With probability $1-P$ transmission is delayed by τ seconds which is the maximum propagation delay between any pair of terminals. Due to propagation delay there may be

more than one terminal transmitting at the same time in which case terminals are retransmitted after random timeout intervals. An optimum value of P for a given load can be chosen so as to balance the probability of retransmission with channel utilization. In the non-persistent version of CSMA, transmission is rescheduled with a random timeout interval when line is sensed to be busy. The detection of carrier on the line is treated as though there were a collision in so far as message transmission is concerned. Studies have shown that the optimum P-persistent and the non-persistent protocols have similar performance characteristics.

The latest development of random access protocols which have particular application in local area networks is Carrier Sense Multiple Access with Collision Detection (CSMA/CD).[33] In the CSMA protocols messages from stations spaced by t seconds will collide if they transmit within t seconds of one another onto a clear line. In CSMA/CD this collision is detected and transmission is aborted. There may be a certain reinforcement interval after a collision which assures that all stations on the line detect the collision. After the collision transmissions are rescheduled after a random timeout interval.

A number of studies of CSMA have been carried out by means of analysis and simulation. From our point-of-view the difficulties with these have been the obtaining of a consistent model for the comparison of different protocols. For example in the previous section the work of Hashida[25] gives the message delay for a Token Passing in the case of the Poisson Arrival of arbitrary length messages to infinite capacity buffers.

No analysis or simulation based on the same assumptions has been done for CSMA/CD . In order to utilize existing analyses of the same model for both Token Passing and CSMA it is necessary to examine a more restrictive model. Accordingly we shall assume that messages are all of the same constant length and that each station can hold only a single message at a time. We assume that the time until arrival at an empty buffer follows a Poisson Distribution. Based on these assumptions, Kaye's[34] analysis of Token Passing and Lam's[35] analysis of CSMA/CD are appropriate to our study. The details of calculations based on these models were presented in the interim report to the project. A typical result of these calculations are shown on Figure 4.2 where average message delay is shown as a function of the load offered to the system for a system with fifty stations. We see for CSMA the same sort of pattern that was indicated for the ALOHA system, good performance for light loading but a rapid deterioration with increased loading. In contrast the Token Passing system has slightly worse performance at light loading but a more graceful degradation as load increases. These results are in conformity with the findings of other workers.[36]

A good deal of effort was expended on developing a more general performance model for CSMA/CD in the sense of general message distributions, large station buffers and higher moments of delay. At this writing only partial success can be reported. We developed a complex but accurate model of the system. The difficulty is that the complexity of the model have led to numerical problems which have prevented a complete evaluation of the model. We feel that these problems present no real obstacle and

results will be soon forthcoming. A simulation program designed to model the system is in the final stages of development as well. The goal is to evaluate the accuracy of both the simple and the complex model by means of simulation. In the next section of this report we shall discuss a simulation of the HDLC protocol. Our objective is to link the CSMA simulation program with that of HDLC . In fact software was developed with this objective in mind.

In order to evaluate the CSMA/CD protocol on the same basis as Token Passing we attempted to develop a model based on the M/G/1 queue. The effort was not successful however the source of our difficulties may be of interest. The model assumes that messages of arbitrary distribution arrive at an infinite queue at a Poisson rate. We shall assume the non-persistent discipline so that stations sensing the line to be busy reschedule transmission. Stations involved in a conflict also reschedule transmission in the same fashion. Once a station gains access to the line, it transmits all of the messages including new arrivals until its buffer is empty. Let the mean and the mean square value of the time required to gain access to the line be denoted by $\bar{s}$ and $\overline{s^2}$. We shall assume for the moment that these quantities are known. The average delay of a message can be found from the standard analysis of the M/G/1 queue with a minor modification. The service time of the first message to gain access, i.e., the first message in a busy period, is augmented by the time requiied to gain access to the line. The resulting delay is:

$$D = \frac{\lambda \overline{m^2}}{2(1 - \bar{m}\lambda)} + \bar{m} + \bar{s} + \lambda(\bar{m}\,\bar{s} + \overline{s^2}/2) \qquad (4.2)$$

where $\bar{m}$ and $\overline{m^2}$ are respectively the mean and the mean square values of the messages. Notice that if $\bar{s} = \overline{s^2} = 0$ we have the delay for the M/G/1 queue. It is not difficult to find higher moments of delay provided that higher moments of the message length and of the access time are known.

The real problem then is finding the distribution of this access time. Each of the stations with messages contend for exclusive access to a free line. Once access is gained the line is occupied for a time interval equal to a k-fold busy period where k is the number of messages that have accumulated in the station buffer since the station was last emptied. After the k-fold busy period, the contention begins again. It appears to be a safe assumption that the successful station is chosen at random from those contending. From all of these elements it is necessary to form estimates of the distribution of the moments of the access time. In the next phase of our work this will comprise part of our effort.

Tree Search Techniques

In the introduction to this section it was pointed out that due to retransmissions the slotted ALOHA technique was limited to a maximum line capacity of 36% . Furthermore there is the more serious problem of instability in the channel. Recently there has been something of a breakthrough in random access protocols. By the application of a technique called Probing[36,37] which is essentially a tree search, the capacity of

the channel has been increased to 43% for the first application[38] and to over 50% of capacity for subsequent refinements.[39-40] Moreover instability in the operation of the channel has been eliminated.

In the slotted ALOHA context the probing technique grants access to a group of stations simultaneously. If any of the stations have messages they transmit immediately. If there is a conflict between two or more stations having messages, the initial group is split in two branches and access is granted to each branch in turn. This stands in contrast to the random retransmission technique which had been used for conflict resolution. In the event of continued conflict splitting into branches continues. The process continues until all messages are isolated within a branch. Probing can be made adaptive in that the search of a group of terminals begins not by granting access to the entire group but to subgroups (see below).

The tree search technique can be used to resolve conflicts in CSMA . Stations sensing the line to be free transmit in the same fashion as previously. If two or more terminals conflict, the conflict is resolved by splitting the entire group in two parts and granting access to each group in turn rather than by a random rescheduling of transmission. After a very long transmission the probability of a station having a message is large and if access were granted to all stations simultaneously conflict would almost surely ensue. In the adaptation of the tree algorithm the total number of stations are split into groups at the end of a long transmission and each group is given access to the line in turn.

In order to evaluate tree search for conflict resolution in CSMA, calculations based on simple models were carried out. It was assumed that for each of N stations the probability of having a message is P . For both tree search and random retransmission we compute the average amount of time required to resolve all conflicts and to have each message transmitted. While the conflict resolution is going on it is assumed that there are no new arrivals to the system. It is recognized that this is a great oversimplification since the conflict resolution in real systems is carried out amid continual arrivals. Nevertheless we feel that the computation will give a valid comparison of the collision resolution capability of the two approaches.

The adaptive probing technique is here used in the random access context where an inquiry is answered by either silence, a successful message transmission, or a garbled transmission resulting from a conflict between two or more users. Each inquiry requires one slot, the inquiries effecting a binary search of, for example, 2^j users. It has been shown that a recursive relationship for the generating function $Q_j(z)$ of a random access probing cycle is given by

$$Q_j(z) = zQ_{j-1}^2(z) + (z-z^2)\,[N_{0j-1}^2 + N_{0j-1}\,Q_{j-1}(z)] + N_{0j-1}\,N_{1j-1}\,(2z-z^2-z^3)z^m \; ; \qquad (4.3)$$

where

where

2^j = the number of users in a group,

N_{0j} = Pr {no messages in 2^j stations} = $(1-p)^{2^j}$,

N_{1j} = Pr {one message among 2^j stations} = $2^j p(1-p)^{2^j-1}$,

m = the number of slots/messages ;

and

$$Q_1(z) = (1-p)^2 z + 2p(1-p)z^{m+1} + p^2 z^{2n+3} ,$$

where

P = Pr {a station has a message} .

From the above it is a simple matter to find the mean and mean-square of the time needed to transmit all messages:

$$E[\text{cycle}_j] = Q'_j(1) = 1 + 2Q'_{j-1}(1) + N_{0j-1}\,[N_{0j-1} + 3N_{1j-1} + 1] ,$$

with

$$E[\text{cycle}_{j=1}] = Q'_1(1) = 1 + 2p^2 + 2mp; \qquad (4.4)$$

The probing technique is adaptive in the sense that the initial 2^n users may be divided into 2^{n-k} groups of 2^k , each group to be probed separately. The number of groups is chosen so as to minimize the duration of a cycle. The mean total cycle time is thus $2^{n-k}\,Q'_k(1)$. To minimize the mean cycle time, one chooses $k^* = \max\{0, 1, \ldots , j, \ldots , n\}$ such that

$$N_{0j} + N_{0j-1} + 3N_{0j-1} N_{1j-1} - 1 > 0 \ . \qquad (4.5)$$

Note that $k^* = 0$ implies, as a result of the relatively high probability that each station has a message, that conventional polling, sequentially querying each user, will minimize the mean cycle time. Adaptive probing here reverts to polling whenever the probability p that a user has a message exceeds $1/\sqrt{2} = .707$.

The foregoing analysis assumes that a single inquiry is needed to probe the system irrespective of the number of messages in the system. Thus even if the system is empty the duration of a cycle is one inquiry time. Also in the case of a single message in the system a single inquiry time is allocated. Now suppose we take the point-of-view that the tree search begins only after there has been a conflict. The average cycle time is reduced by the probability of there being zero or two messages in the system. In order to be complete we shall show the results of both computations.

In contrast to the systematic procedure of adaptive probing, the decentralized non-deterministic scheme of slotted non-persistent CSMA operates in the following fashion:

(1) at the beginning of each time slot, stations with messages ready to transmit sense the channel and transmit if it is sensed idle ;

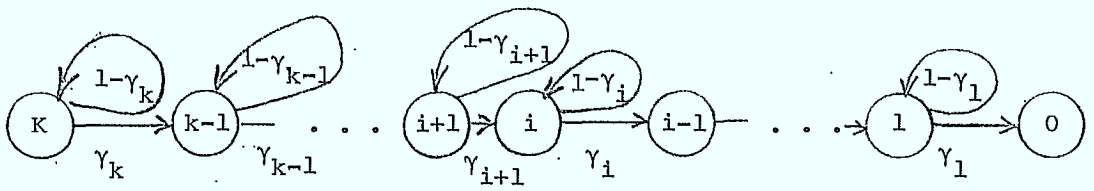
(2) if the channel is sensed busy or if a transmission conflict has occurred, then the user, now backlogged, schedules retransmission according to the retransmission delay distribution.

Assume here that the retransmission delay is geometrically distributed with mean $1/\nu$ slots; i.e., each backlogged user senses the channel with probability ν. Messages are of fixed length m slots, where one slot, being the length of a probing inquiry, is assumed to be at least as long as the maximum propagation delay τ between stations. In accordance with the probing analysis, assume that the message length m incorporates τ and thus in itself represents the full amount of time that the channel is sensed busy after a transmission. No collision detect and abort capability is assumed - - conflict between two or more users lasts one message length - - yet the acknowledgement indicating a foiled transmission is assumed to arrive in a negligible amount of time. If only one station has a message, then its transmission is immediate and successful. If $k > 1$ stations have a message, then all k transmit and collide in the first m slots. All k stations are then backlogged and must resolve their conflict through the random retransmission policy.

Let γ_i be the probability of a successful message transmission given that i stations are backlogged. From the above description, γ_i must equal the probability that only one of the i backlogged stations senses the channel:

$$\gamma_i = i\,\nu(1-\nu)^{i-1} . \tag{4.6}$$

The resolution process is represented by the accompanying transition diagram.



The time spent in each state of the transition diagram is geometrically distributed and statistically independent of the time spent in other states; the probability of $(n-1)$ collisions, each of length m slots, finally followed by a successful transmission, given that there are i backlogged stations, is $\gamma_i(1-\gamma_i)^{n-1}$. Thus the generating function of the number of slots spent in state i is

$$\sum_{n-1}^{\infty} \gamma_i(1-\gamma_i)^{n-1} z^{nm} = \frac{\gamma_i z^m}{1 - (1-\gamma_i)z^m} \tag{4.7}$$

The generating function $G_k(z)$ of the number of slots required until all k packets are transmitted without collision is simply the product of the above generating functions:

$$G_k(z) = \prod_{j=1}^{k} \frac{\gamma_i z^m}{1 - (1-\gamma_i)z^m} = z^{km} \prod_{j=1}^{k} \frac{\gamma_i}{1 - (1-\gamma_i)z^m} . \tag{4.8}$$

From $G_k(z)$, one can obtain $E[T|k] = G_k'(1)$, the mean number of slots required given k stations are backlogged:

$$E[T|k] = km + m \sum_{j=1}^{k} \frac{1-\gamma_i}{\gamma_i} \quad i \quad k \geq 2 \ . \tag{4.9}$$

Similarly, $E[T|k] - E[T|k] = G_k''(1)$ is given by the expression

$$E[T^2|k] - E[T|k] = km(km-1) + 2km^2 \sum_{j=1}^{k} \frac{1-\gamma_i}{\gamma_i} + \sum_{j=1}^{k} \frac{\{\gamma_i(1-\gamma_i)m(m-1)+2m^2(1-\gamma_i)^2\}}{\gamma_i^2}$$

$$+ \sum_{j=1}^{k} \frac{(1-\gamma_i)}{\gamma_i} \sum_{\substack{j=1 \\ j\neq 1}}^{k} \frac{(1-\gamma_j)}{\gamma_j} m^2 \quad ; \quad k \geq 2 \ . \tag{4.10}$$

With p equal to the probability that each station has a message, the probability that k of N stations have a message is given by

$$\Pi_k = \binom{N}{k} p^k (1-p)^{N-k} \ . \tag{4.11}$$

From the discussion of the CSMA scheme, it is clear that the mean cycle time $E[T]$ (in slots) required to clear the system is

$$E[T] = [1 - \pi_0]m + \sum_{K=L}^{N} E[T|k]\pi_k \ , \tag{4.12}$$

and the variance of the cycle time is given by

$$Var(T) = (1 - \pi_0)\pi_2 m^2 + \sum_{K=L}^{N} Var(T|k)\pi_k \tag{4.13}$$

The mean and variance of the cycle time depend on the value of ν, the probability that a backlogged user senses the channel. Just

as the adaptive probing algorithm takes advantage of knowledge of p, the probability that a station initially has a message, to minimize the mean probing cycle time, so too can the mean CSMA cycle time be minimized by judicious selection of the parameter ν. While an analytic expression for such a value ν^* is not readily obtained, it is easy to see that ν^* hinges on the value of p. To make the desired comparison with adaptive probing, then, ν was chosen as that value minimizing $E[T]$ within a resolution of .005.

Discussion

The results obtained from this comparison of random retransmission and adaptive probing are exhibited in Figures 4.3 - 4.5 of the mean and variance of the number of slots required to clear the N terminals versus p, the probability each of the N terminals having a message. A feature common to all the graphs is the superiority of adaptive probing over CSMA for large values of p. This result is to be expected since the mean time to clear for adaptive probing is always less than or equal to that required by conventional polling. When all stations are likely to have messages, however, the contention scheme of random retransmission comparatively wastes time as stations compete for exclusive use of the channel.

On the other hand, when the system is lightly loaded, i.e., for low levels of p, random retransmission often exhibits an advantage over adaptive probing due to the fact that it is very likely that only a single station has a message and there is no competition for the channel.

The adaptive probing algorithm may waste inquiries on groups of stations that have no messages at all. This characteristic is most vivid in the set of curves for messages of one slot length: adaptive probing requires at least one inquiry to determine whether there are any messages in the system at all, while stations without messages do not do anything under the random retransmission protocol. The advantage of random retransmission, however, diminishes for fixed values of p as the number of stations increases, since then the probability of more than one message in the system, and hence the need for contention resolution, increases. Similarly, CSMA's performance relative to adaptive probing's deteriorates as the message length increases - - adaptive probing's one slot inquiry overhead then becomes less significant with respect to the total delay.

Now if we take the point-of-view of comparing probing and random retransmission only when true contention is taking place the advantage of random retransmission disappears. We assume that the contention resolution interval begins when two or more terminals have collided. In this case, as shown on the curves, there is very little difference between probing and CSMA at light loading while probing retains its advantage at heavy loading.

It should be noted that this comparison has cast the CSMA protocol in the best possible light in the sense that the mean retransmission delay $1/\nu$ has been adjusted for each value of p, N, and message length m to minimize the mean time required to successfully transmit all the messages initially present. Typically, CSMA protocols such as that implemented in Ethernet establish an initial value of $1/\nu$ and then progressively in-

crease its value in the event of a subsequent collision. Of course, the actual CSMA protocol is such that the initial condition assumed here, the presence of messages being at each of N terminals with probability p , is somewhat contrived since, as mentioned previously, CSMA is an ongoing process in which such a situation is not routine. A more thorough evaluation of adaptive probing in CSMA would consider their performances in terms of, say, message delay as a function of system load where one accounts for successive arrivals and the possibility of a message arriving during a fraction of a probing cycle. The point here, however, has been to compare the relative efficiencies of random retransmission and adaptive tree search as contention resolution processes.

Propinquity

In the foregoing comparison of probing and random retransmission it is assumed that the time required to make an inquiry, i.e., grant access to a group of terminals is the full propagation delay in the system since this is presumed to be the time that is required to decide whether there is a conflict between two or more stations. Now the essence of the tree search technique is to split groups of stations. It is not unreasonable to compose subgroups of stations according to propinquity, i.e., terminals in the same area in the same subgroup. Thus the time required to decide whether a conflict has occurred within a subgroup is not the propagation delay for the whole system and the time required to resolve conflicts is reduced. The number of inquiries may be the same but the duration

of inquiries is reduced in proportion to the size of subgroups. This effect can be illustrated by a straightforward example. We assume that stations are equally spaced along a bus. This is something of a worst case since it is likely that in actual systems stations will be clustered. As in previous cases we assume that messages arrive at a Poisson rate. Again we use as the measure of performance the time required to transmit all messages. An expression for the probability generating function of this cycle time similar to that in equation (4.3) has been derived. From this expression the average time required to probe 2^j stations can be found. Faced with 2^N terminals each of which have a message with probability P this calculation for average cycle time can be used to segment the group in an optimum fashion. This is exactly the same procedure as in the previous case the only difference being that the duration of an inquiry is proportional to the size of the group of stations being probed. The results of illustrative calculations are shown on Figure 4.6 for 32 stations. The topmost curve shows the duration of a cycle when the full roundtrip delay is the duration of each inquiry. In computing this curve the optimum starting groupings as given in equation (4.5) is used. In contrast by varying the inquiry time according to the size of groups one obtains the lower curve. As is evident there is considerable reduction in the cycle time.

The idea of taking advantage of propinquity is not appropriate to random retransmission but only to Probing or Tree search. The results of this and the previous section indicate that tree search offers potential advantages over random retransmission as a means of resolving con-

flicts among stations transmitting messages simultaneously. In order to fully evaluate the relative performance of the two techniques a full scale simulation would be necessary, since a complete analytical evaluation is not analytically tractable.

Assuming, as seems likely at this point, tree search yields superior performance then the relative complexity of the two techniques should be assessed. It seems that the tree search requires more complex logic at the stations. However, in the era of VLSI, more complex logic is no obstacle.

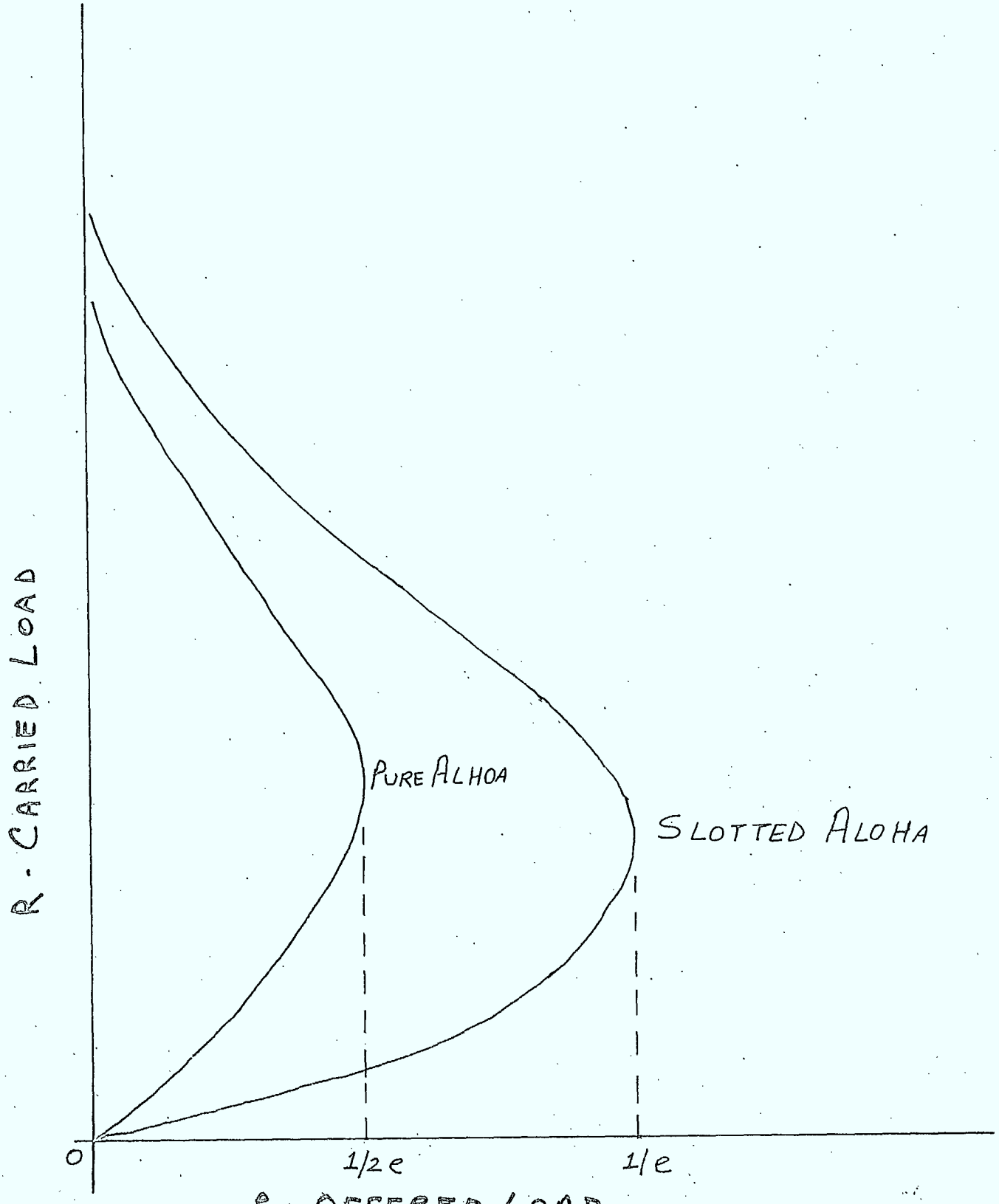

R - CARRIED LOAD
PURE ALHOA
SLOTTED ALOHA
0
1/2e
1/e
S - OFFERED LOAD

FIGURE 4.1

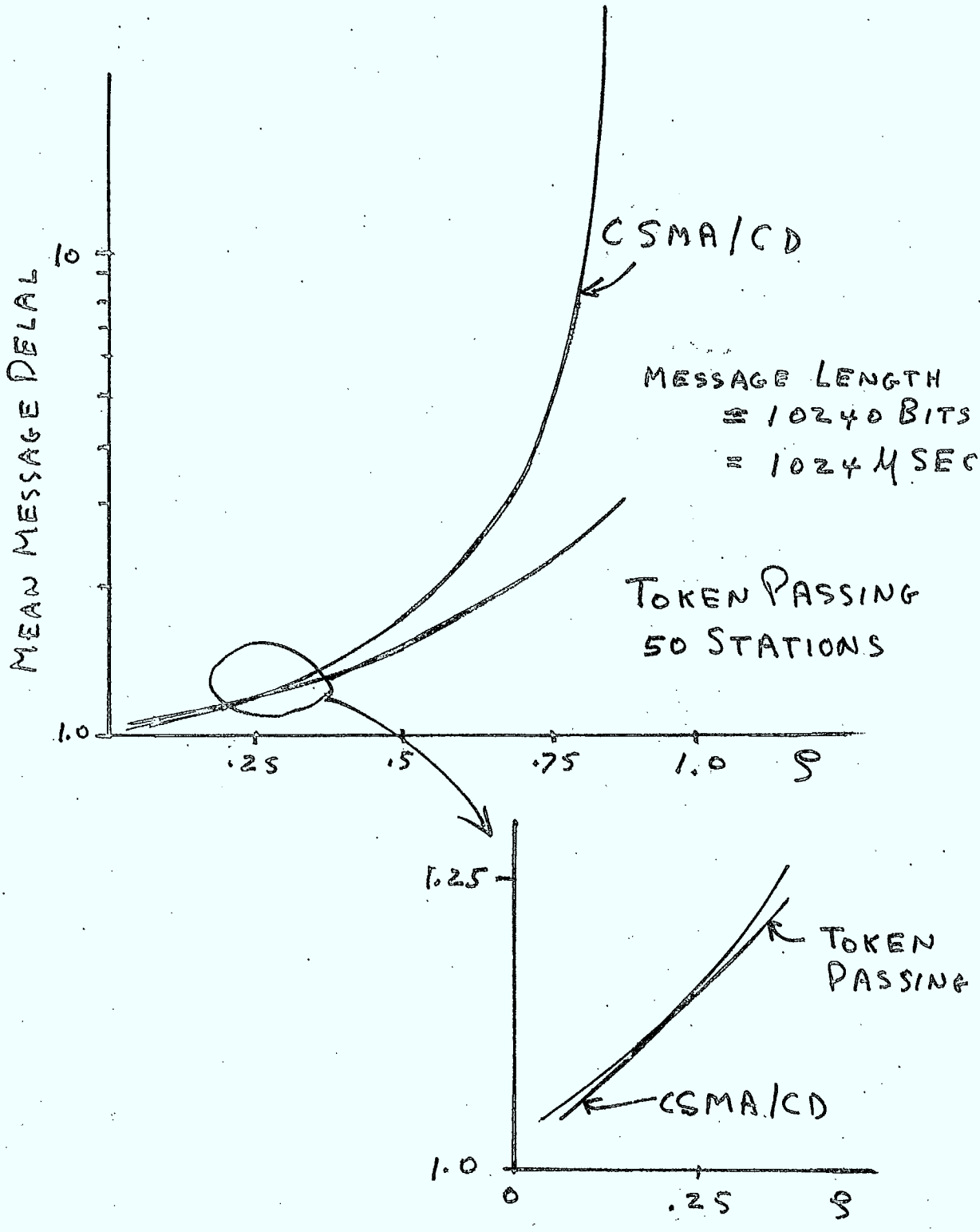

MEAN MESSAGE DELAL
10
1.0
.25
.5
.75
1.0
S
CSMA/CD
MESSAGE LENGTH
= 10240 BITS
= 1024 μSEC
TOKEN PASSING
50 STATIONS
1.25
1.0
0
.25
S
TOKEN
PASSING
CSMA/CD

FIGURE 4.2

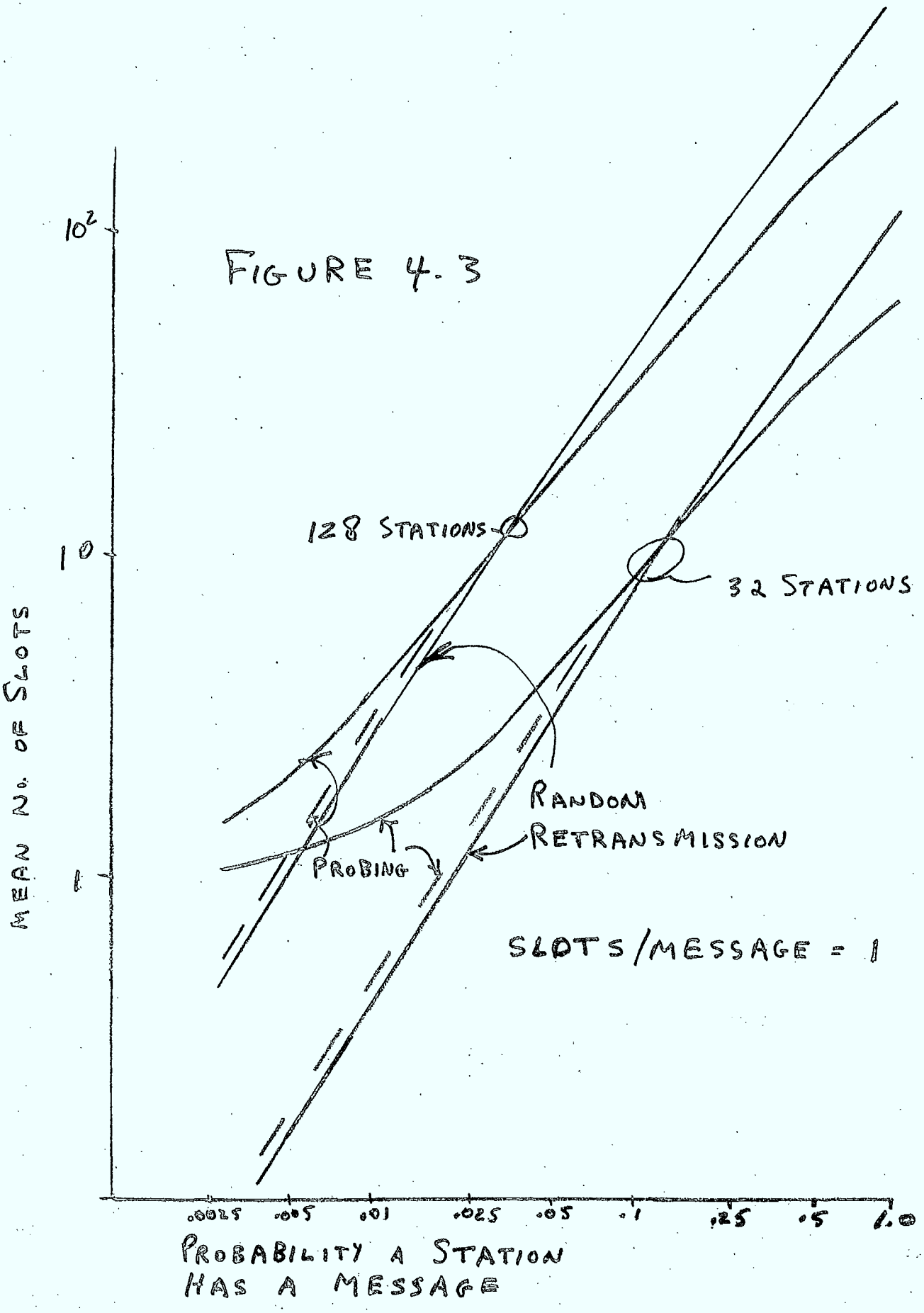
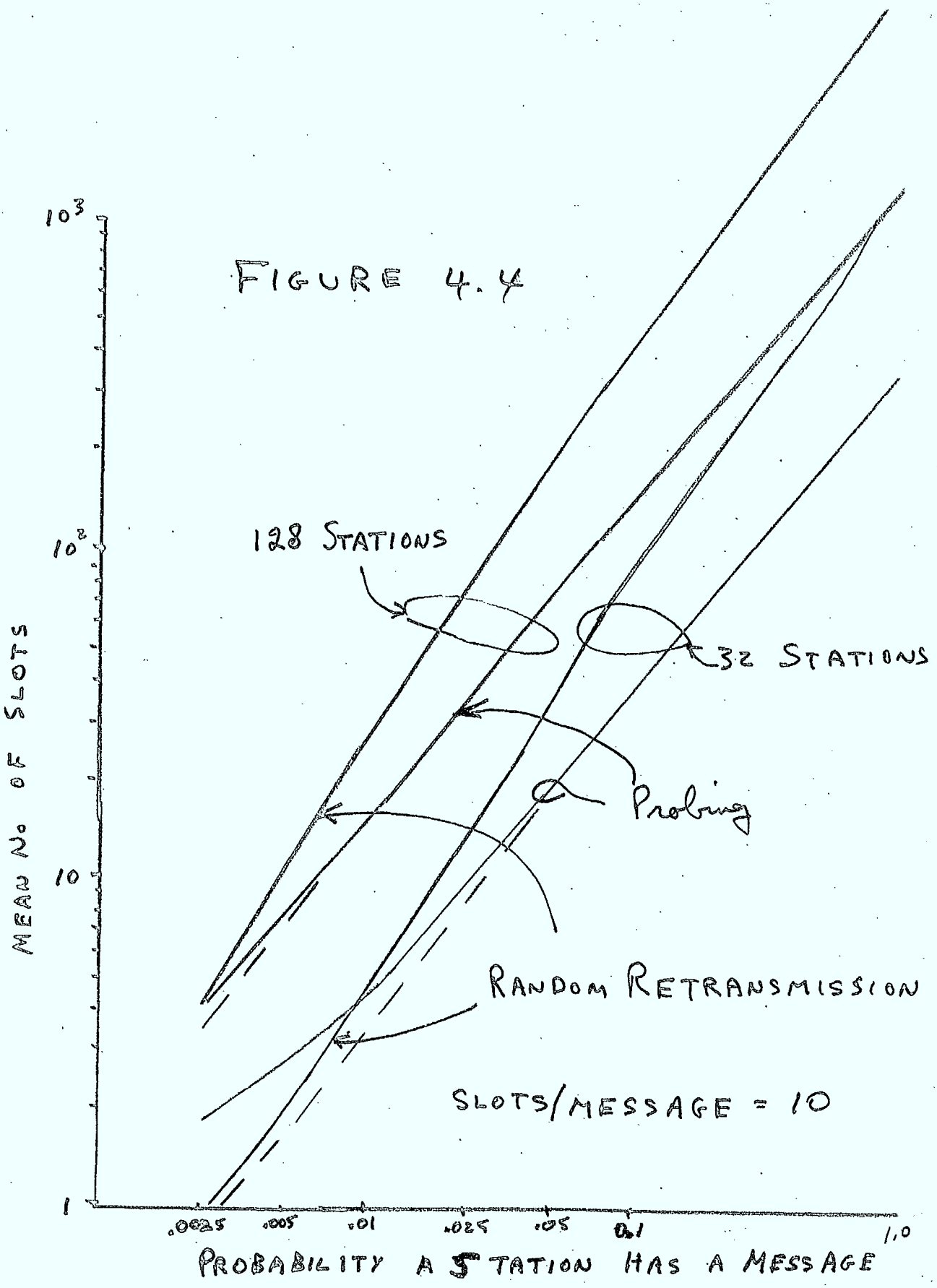


FIGURE 4.3
10^2
10
1
MEAN No. OF SLOTS
128 STATIONS
32 STATIONS
RANDOM
RETRANSMISSION
PROBING
SLOTS/MESSAGE = 1
.0025
.005
.01
.025
.05
.1
.25
.5
1.0
PROBABILITY A STATION
HAS A MESSAGE

FIGURE 4.4


10^3
10^2
10
1
MEAN No. OF SLOTS
128 STATIONS
32 STATIONS
Probing
RANDOM RETRANSMISSION
SLOTS/MESSAGE = 10
.0025
.005
.01
.025
.05
0.1
1.0
PROBABILITY A STATION HAS A MESSAGE

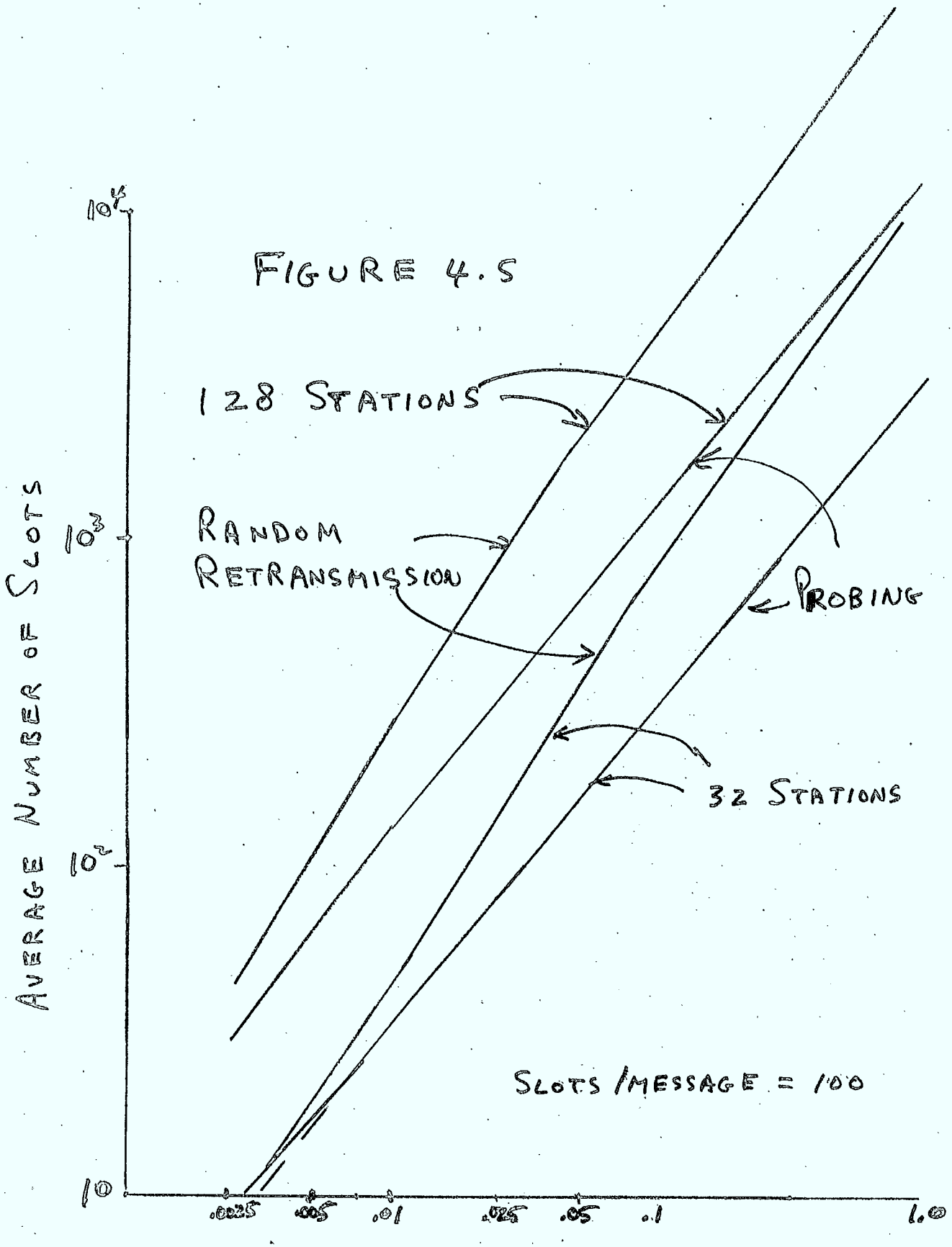
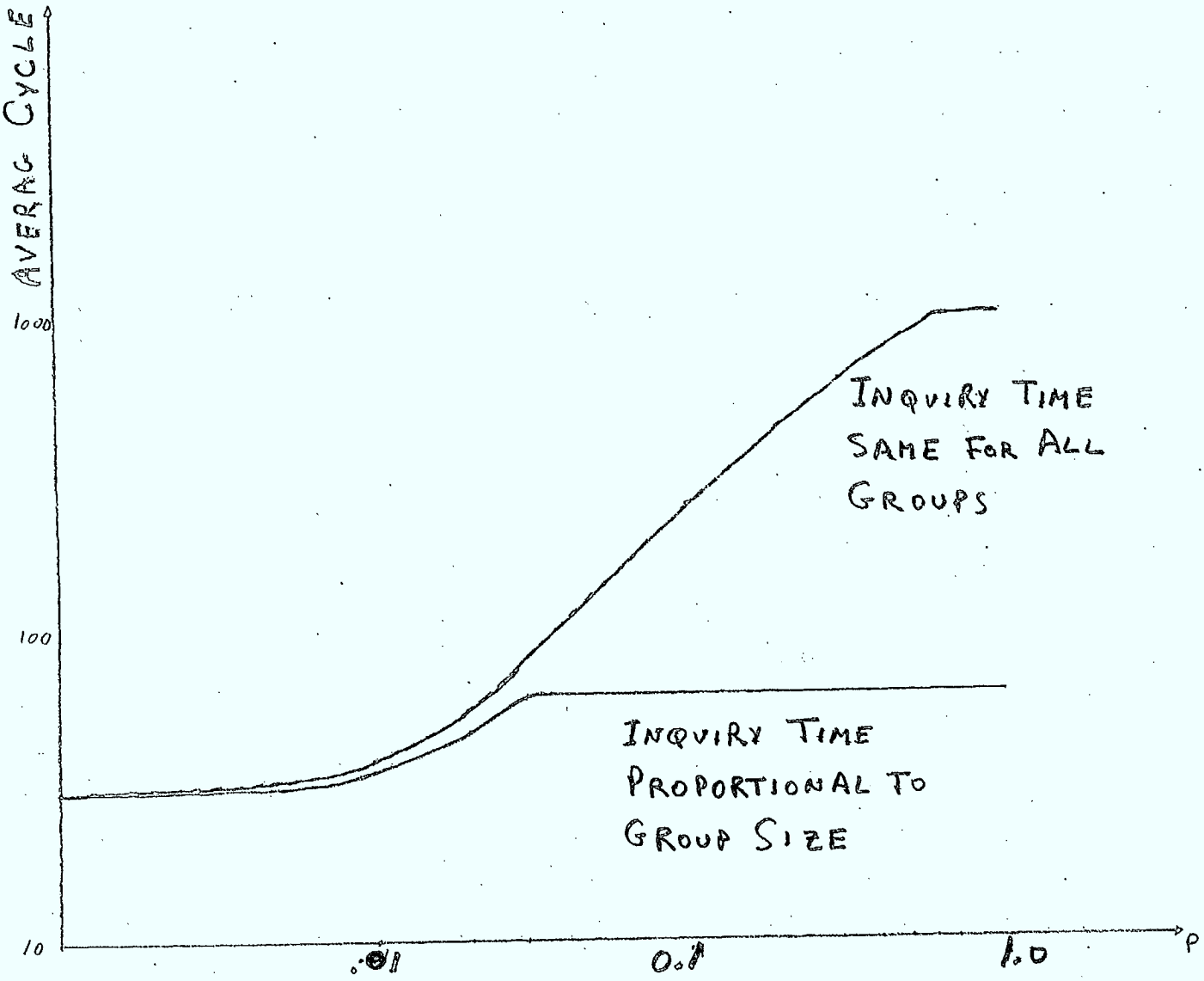


FIGURE 4.5
128 STATIONS
RANDOM
RETRANSMISSION
PROBING
32 STATIONS
SLOTS/MESSAGE = 100
AVERAGE NUMBER OF SLOTS
10^4
10^3
10^2
10
.0025
.005
.01
.025
.05
.1
1.0



PROBABILITY A TERMINAL HAS A MESSAG

FIGURE 4.6

V. High Level Data Link Control (HDLC)

In section II of this report the Open Systems Interconnection concept was discussed. The key here is the layering of protocols with carefully defined interfaces. Of immediate interest is the operation of the sublayers within layer 2 the Link layer. In this layer we have a flow control protocol overlaid on the line access protocols such as CSMA and Token Passing. a key issue in the design of Local Area Networks the interaction of the two protocols within the Link Layer. The effect of the variability of the different random access techniques is of particular interest. The analytical models in the literature take no account of this variability due to obstacles involving mathematical tractability.[41,42] At this juncture it seems that simulation is the best course of action. Insight will be developed on the operation of the protocol and it is hoped that mathematical models will emerge. As mentioned in the previous section of this report simulation will be part of the study of CSMA . So that we shall eventually have a complete system model the programs for HDLC and CSMA are designed to be compatible with one another.

Before describing simulator details and results obtained from the simulation, essential features of HDLC are first presented. As well as being part of the link level protocol proposed by ISO, CCITT Recommendation X 2.5 Layer 2 is one of the permissible options of HDLC. It is currently the most used protocol for computer networks and distributed processing. The main objective of HDLC is to help provide a communication mechanism for a user to send any number and any pattern of bits in a fault-

tolerant manner without being dependent on the topology of the network. Moreover the efficiency of the protocol is not to be affected by such features of transmission links as full or half duplex mode, propagation delays and transmission rates.

A frame (that is, a packet) in HDLC usually consists of 3 header bytes, zero and more information bytes and 3 trailer bytes (see Figure 5.13).

Header Bytes:

Byte 1: The flag, 01111110, which indicates the start of a frame.

Byte 2: An 8-bit address capable of addressing up to 256 units on a network.

Byte 3: A control byte which is described later.

Information Bytes:

Any number including zero of bytes can be carried between the header and the trailer.

Trailer Bytes:

Bytes 1 and 2: Error detection code.

Byte 3: The flag, 01111110, which indicates the end of the frame.

To avoid the mixup between the flag byte and an identical information byte, a scheme called bit stuffing is used in which the 0 bit is inserted after every five consecutive 1 bits in the data. The receiving unit deletes any 0 bit following five consecutive 1 bits in order to retrieve the original data.

The frame makeup as described above is flexible in that the address byte, the control byte and the 2 error detection bytes can be extended to be more than four bytes. These extensions however are not considered the simulation study.

The control byte specifies one of the three frame types possible in HDLC protocol (see Figure 5.13).

Case 1: If the first bit of the control byte is 0 , then the corresponding frame is referred to as an I-frame (information transfer frame). An I-frame is used to send the data. The Ns field of 3 bits in the control byte refers to the sequence number (between 0 and 7) of the I-frame being sent. It implies that no more than 8 different frames can be simultaneously in-transit from one unit to any other units. This is also referred as the window of a transmitting unit. One advantage of this window is that up to 8 frames may be sent before an acknowledgement is received. In case an acknowledgement or a reply of some sort is needed for data of less than 8 frames in size, the P/F bit is set to 1 in the last frame.

The acknowledgement for frames received at one unit can be sent to the transmitting unit in one of the two ways. One way is to use an S-frame described in the sequel. The other is to piggyback the acknowledgement information on an I-frame. The Nr field of 3 bits is used for this purpose. More specifically, if say unit 1 is sending an I-frame to unit 2, then the Nr value informs unit 2 that unit 1 has received all the frames numbered 0, 1, 2, ... , Nr-1, from unit 2, and is now ready to receive the I-frame with sequence number Nr .

Case 2: If the first two bits of the control byte refer to the 10 pattern, the corresponding frame is referred to as an S-frame (supervisory frame). No information bytes are supplied in this frame as its main function is to provide such supervisory control functions as acknowledgements, requesting transmissions and requesting a temporary suspension of transmission. The next two bits (bits 3 and 4) of the control byte indicate the type of the frame.

Case 2a: 00: RR (receive ready)

This type of S-frame is sent by say unit 1 to unit 2 to acknowledge that it has received frames numbered up to Nr-1 correctly from unit 2. It can also be used by a primary unit (such as a computer) to poll a secondary unit (such as a terminal).

Case 2b: 01; REJ (reject)

This type of S-frame is sent to request transmission or retransmission of I-frames numbered Nr and higher.

Case 2c: 10: RNR (receive not ready)

Whenever a unit is temporarily busy and cannot accept any I-frames, it sends out an RNR S-frame. The end of a busy condition may be signalled with any other valid S-frame.

Case 2d: 11: SREJ (selective reject)

This type of S-frame is sent to request transmission or retransmission of a single I-frame numbered Nr .

Case 3: If the first two bits of the control byte are 11, then the corresponding frame is referred to as a U-frame (unnumbered frame). It is used to provide additional link control functions which are not directly relevant to our discussions here.

For control purposes it is often convenient to establish a hierarchy among various units on the same physical link. For instance, a unit is termed primary if it assumes responsibility for the organization of data flow and for error recovery operation on the link. A non-primary or secondary unit i is then under the control of a primary unit. The most general form of hierarchy is achieved when a unit assumes the role of both a primary and a secondary; i.e., it is a combined unit. All units then have the same set of protocols and any unit can send and receive information on its own initiative. Such a mode of operation is called asynchronous balanced mode ABMM or simply as balanced mode.

The efficiency of an HDLC protocol depends on the number of units in a network, message traffic at each unit, error and retransmission probability and on the physical level protocols, CSMA/CD or Token Passing, used in the link. The design of the HDLC simulator described below is motivated by the desire to study the interrelationship among these various factors.

HDLC Simulator

The simulator named sim is written in PASCAL to fully exploit the advantages of the data structure flexibility of PASCAL over FORTRAN. The main emphasis of sim is to be upward compatible with future simulations of higher level protocols in local area networks. Thus in future, sim could become a component of a complete local area network simulator. On its own level which is the link level and physical level, sim is being designed to simulate variables such as number of terminals, different message traffic at each terminal, various error distributions, CSMA/CD and Token Passing protocols, and then determine their effect on such parameters as round trip delay, window size, send and receive buffer size, and number of S-frames in transit.

Sim is event-driven and is constantly executing one of the following events:

1. A message arrival at any terminal.
2. A frame arrival at any terminal.

3. Transmission of an S-frame from a terminal.

4. Transmission of an I-frame from a terminal.

5. Time out.

Messages arrive at each terminal with a Poisson arrival rate, and are converted into I-frames and stored into a transmission buffer. Whenever the window is open and no acknowledgement is due, the I-frames from the transmission buffer are sent out, that is they are put in an in-transit queue. The error distribution and physical level protocol is applied to frames in this queue, and when the transmission is possible, frames are next put into the receive buffer of t the destination terminal. If no errors are detected, an acknowledgement of each frame is sent either using the piggybacking or a separate S-frame. In case of an error, the transmitting terminal waits until a certain time out period and if no acknowledgement is received in that time, the frames are transmitted again. Figures 5.1 to 5.9 contain the flow charts of this procedure. The complete program for sim appears in Appendix A . Every event as it occurs is logged in sim and various graphs are plotted using this logged data. The next section describes some such graphs and results obtained from them.

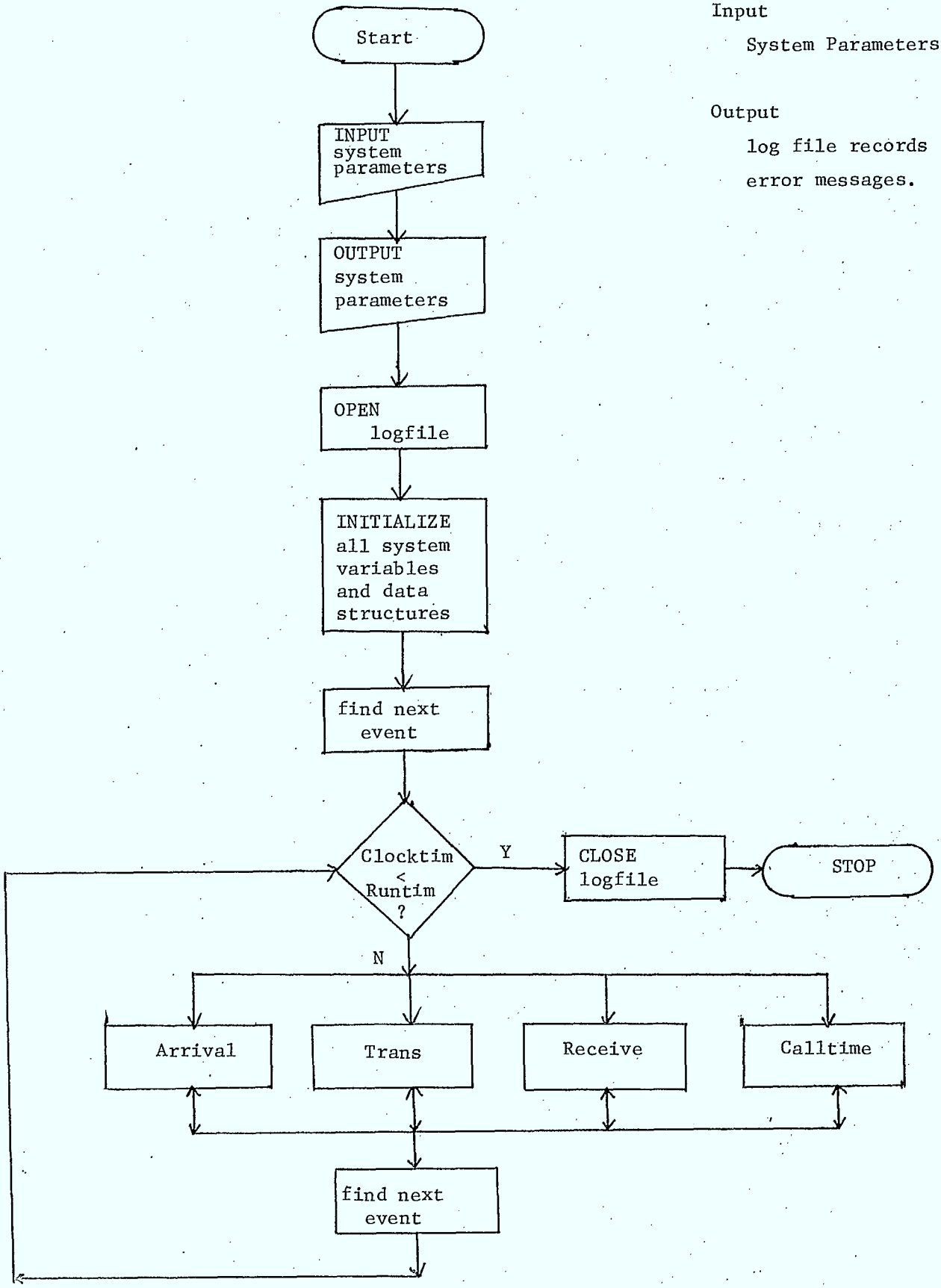
In this early stage of software development only two stations communicating with one another are simulated. However we have in mind systems where a number of stations share the same line through CSMA or through Token Passing. We model this sharing through the probability distribution of the access delay of a message. For CSMA we take the message delay to be a geometrically distributed sum of geometrically

distributed random variables. This roughly models the processes of sensing the line and randomly rescheduling transmission. As the analysis of the CSMA protocol progresses (see section IV), more refined models of message access delay can be used. For Token Passing the probability distribution of access delay was modeled as a constant time followed by a geometrically distributed random variable. The constant term models the overhead required to pass control from one station to another and the geometrically distributed random variable models message transmission. Again as the work progresses this distribution can be refined.

Typical results of simulation are shown on Figure 5.10 for CSMA and on Figure 5.11 for Token Passing. In these curves we show average delay as a function of load with window size as a parameter. The vertical line through points indicates the variability between stations. It is assumed in the simulation that the transit time for the S-frame is one-fifth that of an information frame. The results show the effect of varying the window size. The maximum window size shown is seven since increasing beyond this has no effect. In fact the curve for W = 7 is the same as that for the M/G/1 queue for both CSMA and Token Passing. As the window size is decreased there is a significant deterioration of performance. There is an interesting comparison between Figures 5.10 and 5.11. We see that the model for Token Passing (Figure 5.11) shows much less deterioration of performance as the window size is decreased. In both cases the mean access time is the same. However variance of the access time for the CSMA model (Figure 5.10) is larger. Thus it would seem that a less variable access scheme would prove superior.

On Figure 5.12 we show the average number of S-frames transmitted as a function of load. The decrease in this average with increasing load is entirely expected since as load increases more information frames are available for piggybacking acknowledgements. What was unexpected was that the decrease was very nearly linear.

5.1. Network Main



Input

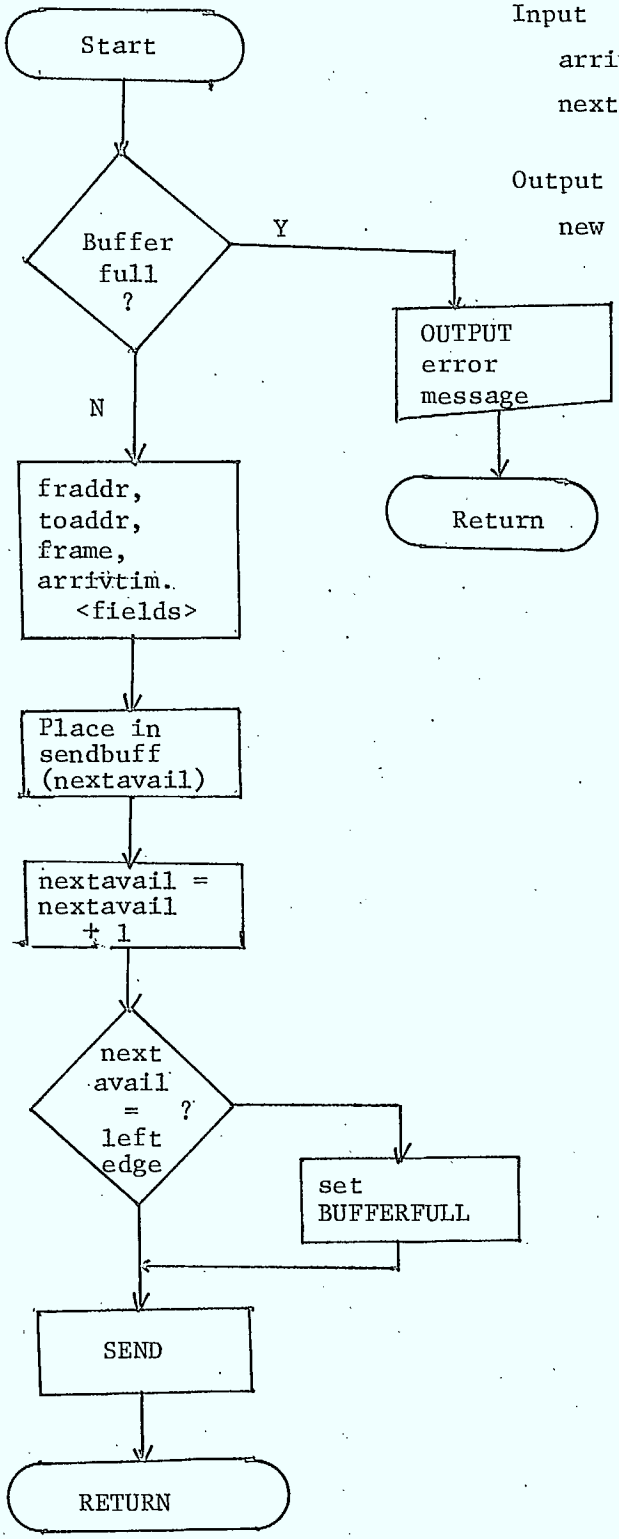
System Parameters

Output

log file records

error messages.

5.2. Arrival


Start
Input
arrivaltime
nextavail
Output
new send buffer
Buffer full ?
Y
N
OUTPUT error message
Return
fraddr, toaddr, frame, arrivtim. <fields>
Place in sendbuff (nextavail)
nextavail = nextavail + 1
next avail = ? left edge
set BUFFERFULL
SEND
RETURN

5.3. Trans

Input

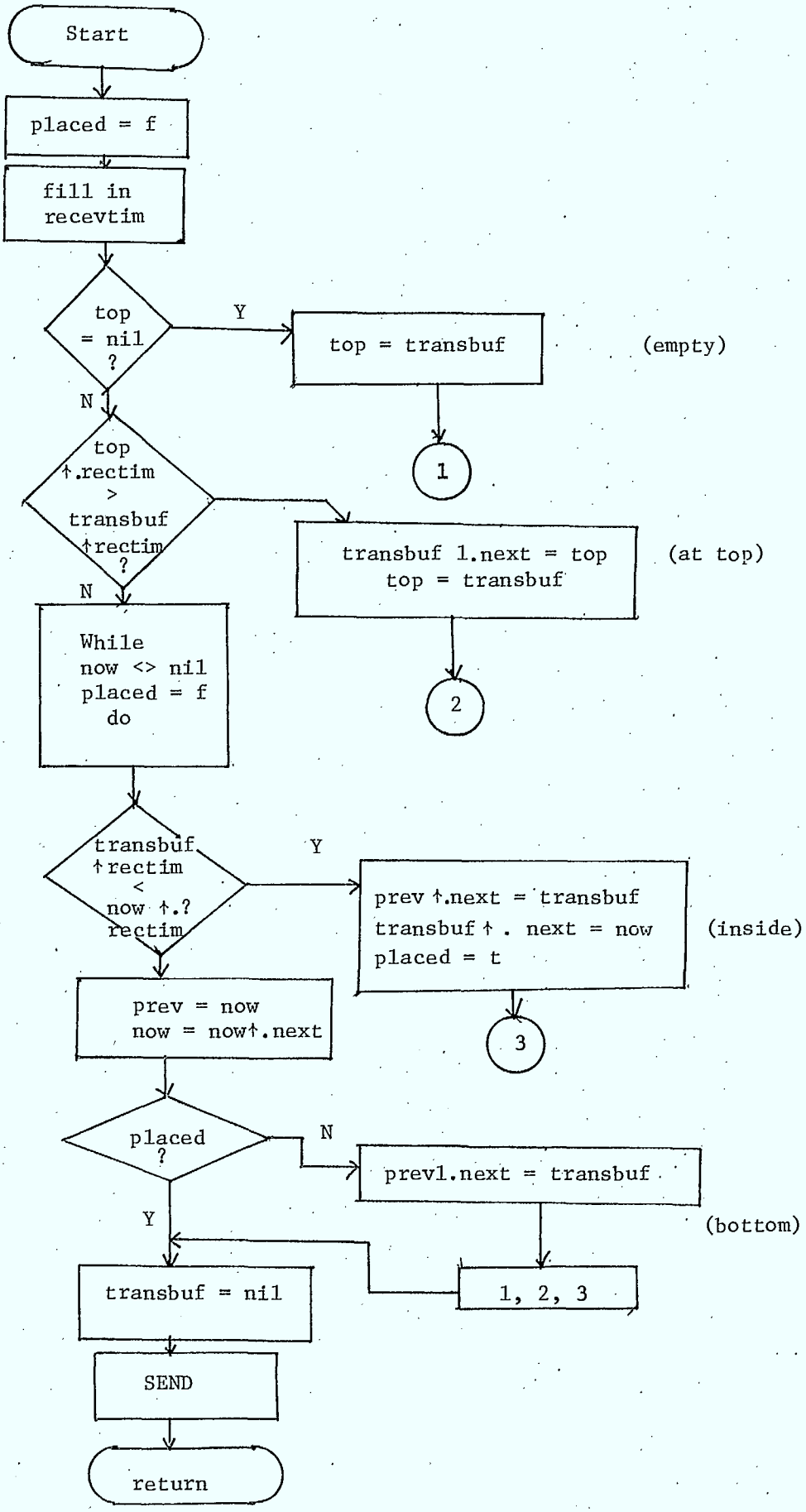
transbuf

Output

updated

in-transit

queue



5.4 Receive

Input:

transbuf

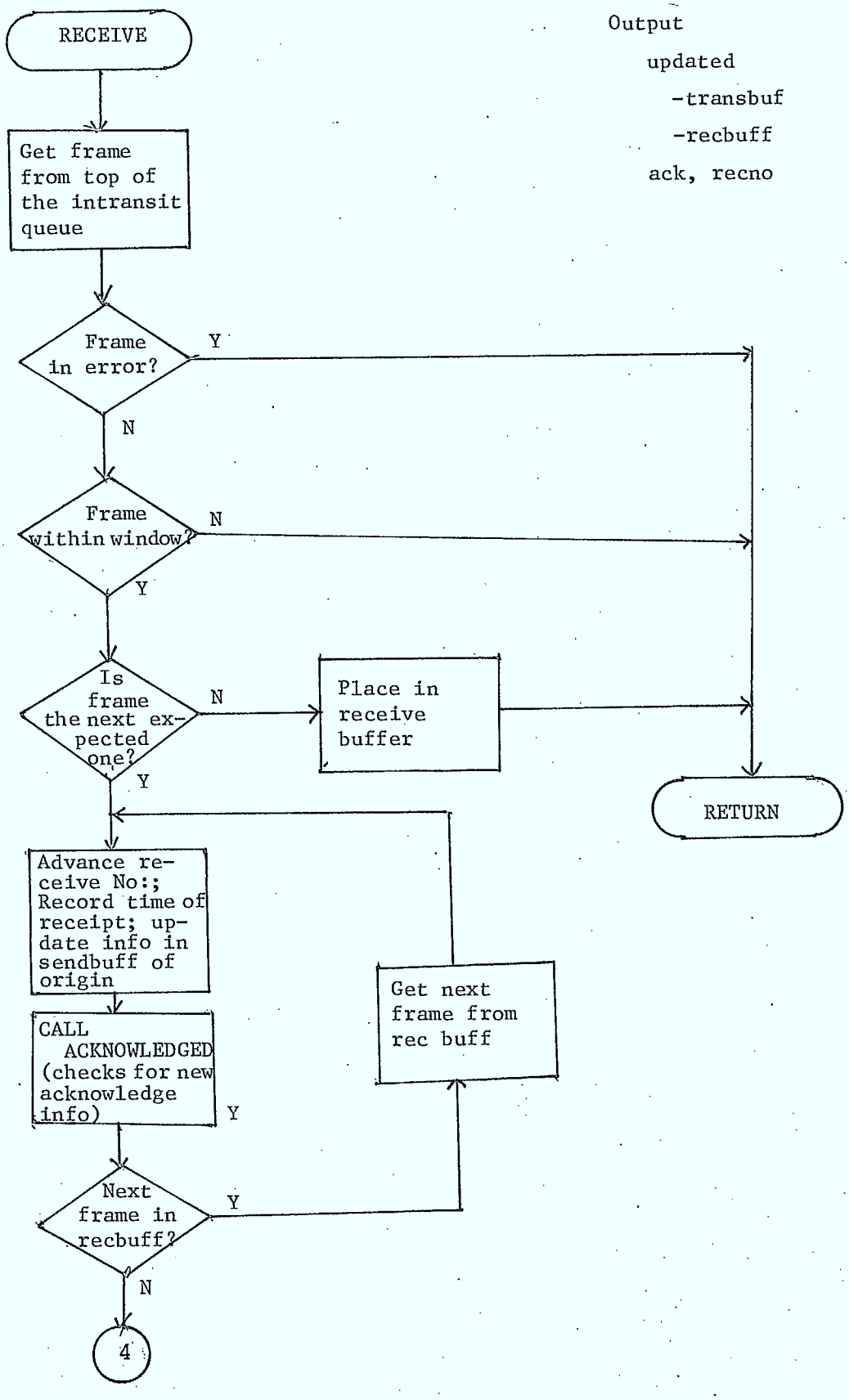
Output

updated

-transbuf

-recbuff

ack, recno

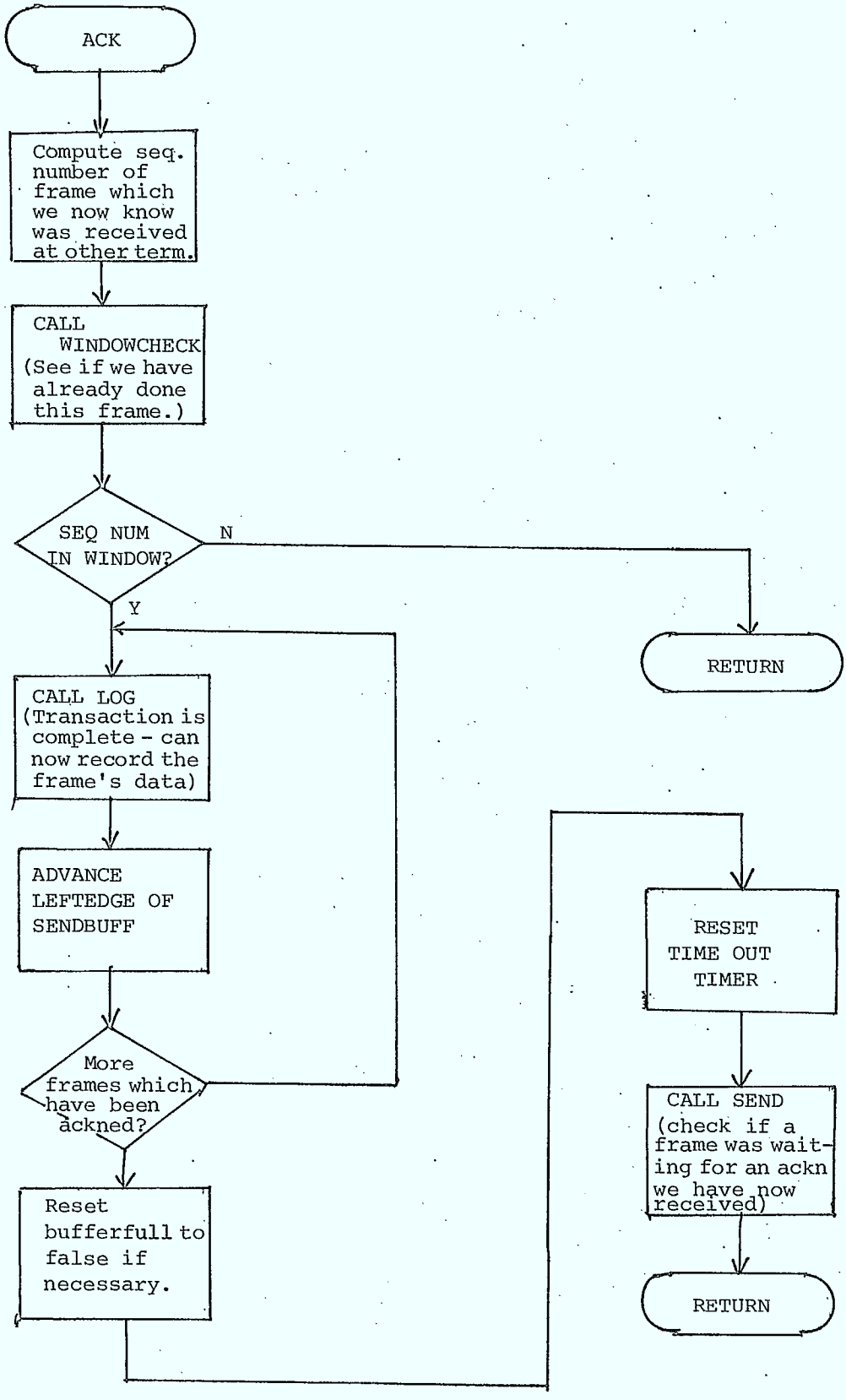


5.5. Calltime

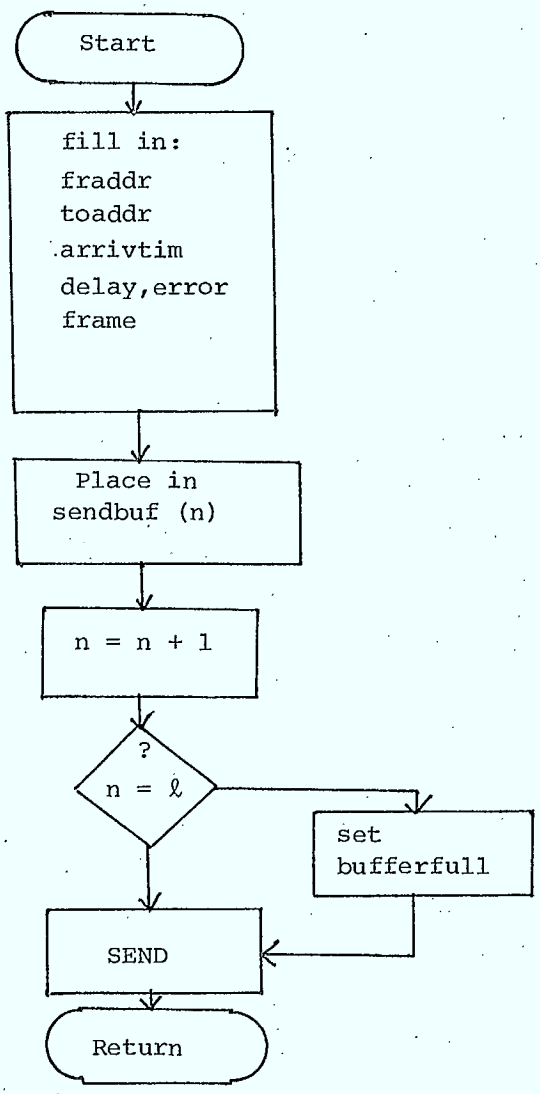
Input
terminal
electtime

Output
new sendseq
new timout time

5.7. Ack


ACK
Compute seq. number of frame which we now know was received at other term.
CALL WINDOWCHECK (See if we have already done this frame.)
SEQ NUM IN WINDOW?
N
Y
RETURN
CALL LOG (Transaction is complete - can now record the frame's data)
ADVANCE LEFTEDGE OF SENDBUFF
More frames which have been ackned?
Reset bufferfull to false if necessary.
RESET TIME OUT TIMER
CALL SEND (check if a frame was waiting for an ackn we have now received)
RETURN

5.8. SSframe



Input

term

sendbuffer

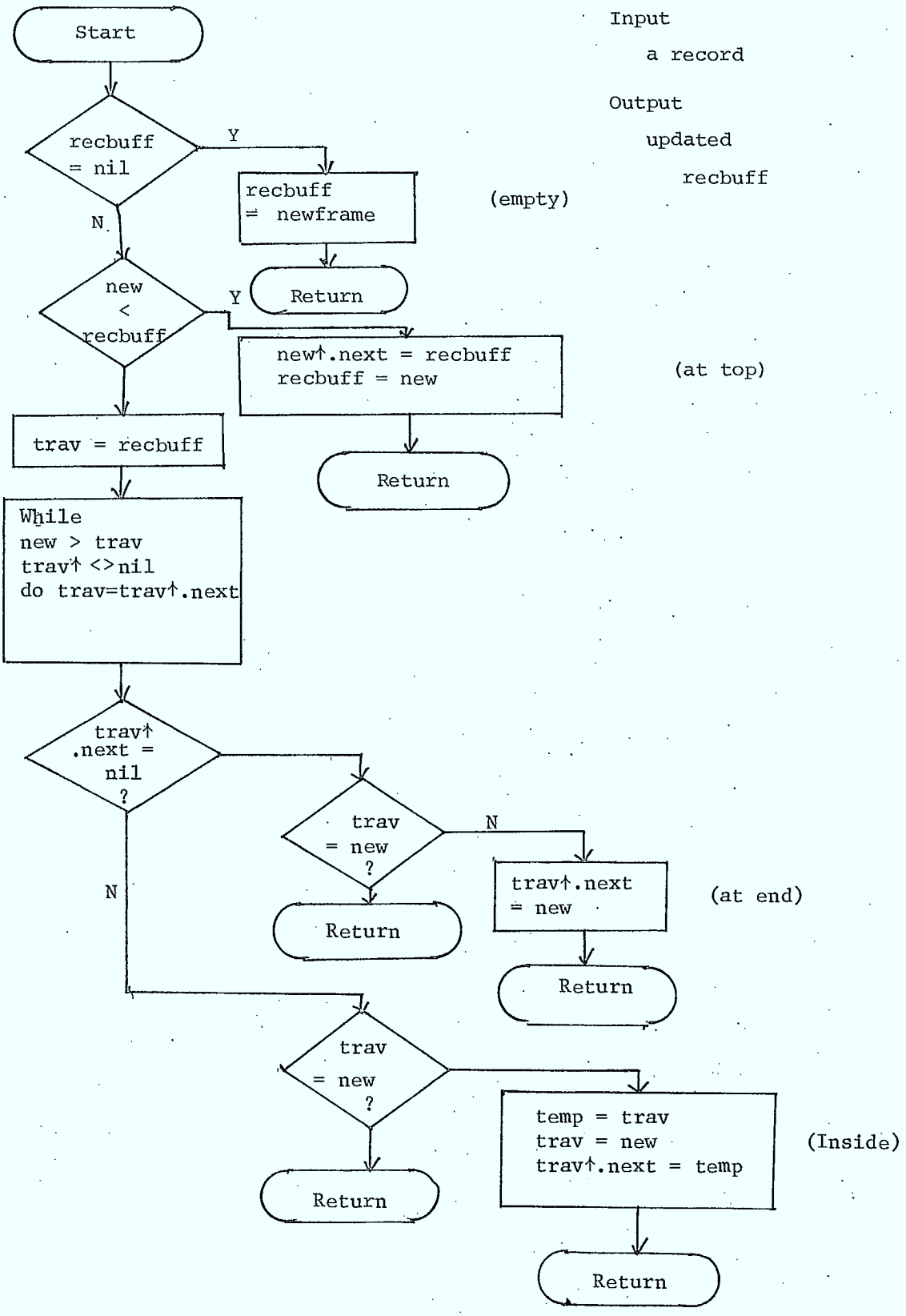
Output

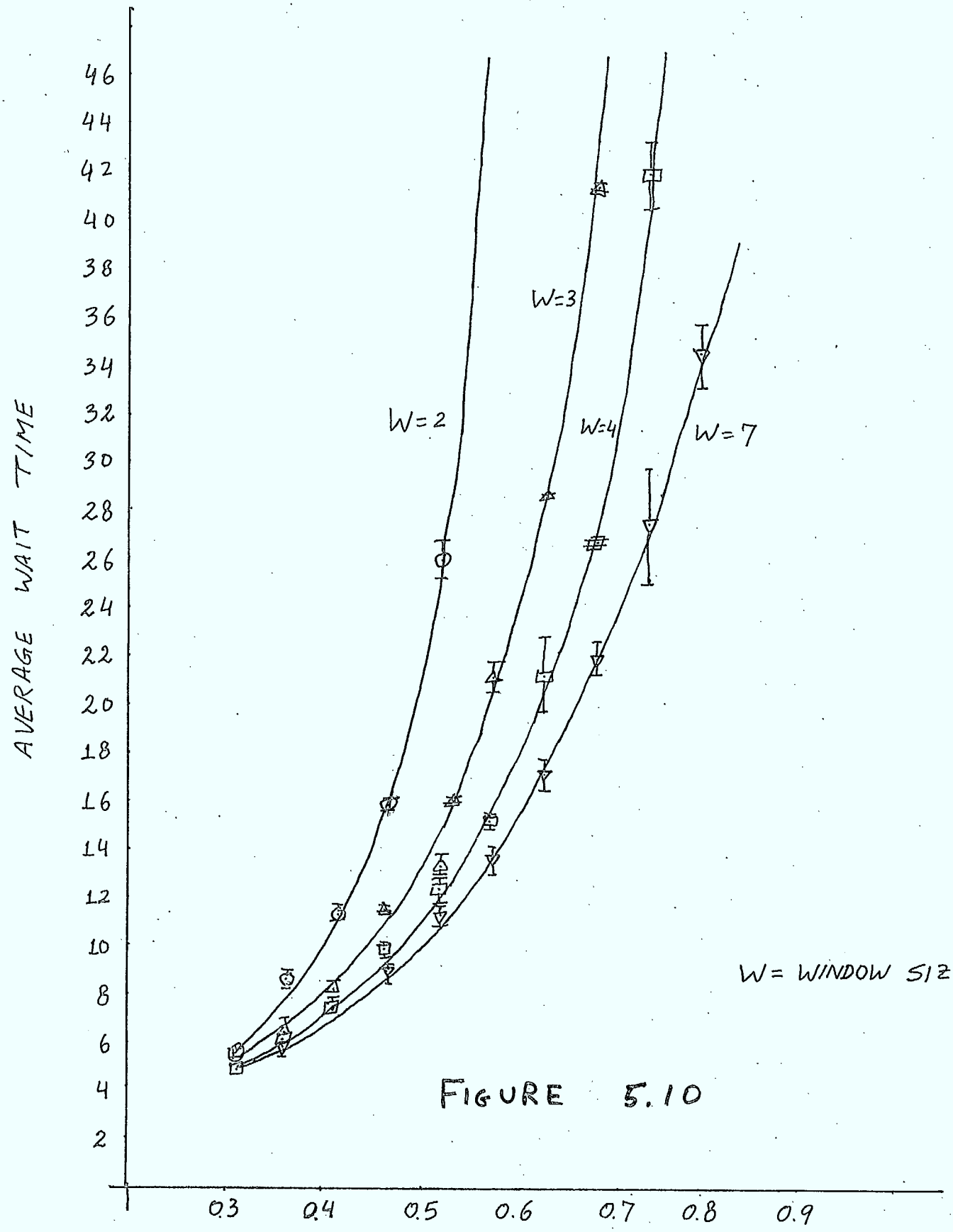
updated

- sendbuffer

nextavail (n)

5.9. Buffer


Start
Input
a record
Output
updated
recbuff
recbuff
= nil
Y
recbuff
= newframe
(empty)
N
new
<
recbuff
Y
Return
new↑.next = recbuff
recbuff = new
(at top)
Return
trav = recbuff
While
new > trav
trav↑ <>nil
do trav=trav↑.next
trav↑
.next =
nil
?
trav
= new
?
N
trav↑.next
= new
(at end)
Return
Return
N
trav
= new
?
temp = trav
trav = new
trav↑.next = temp
(Inside)
Return
Return



W = WINDOW SIZE

FIGURE 5.10

CSMA: DOUBLE SERVER
10,000 SAMPLES

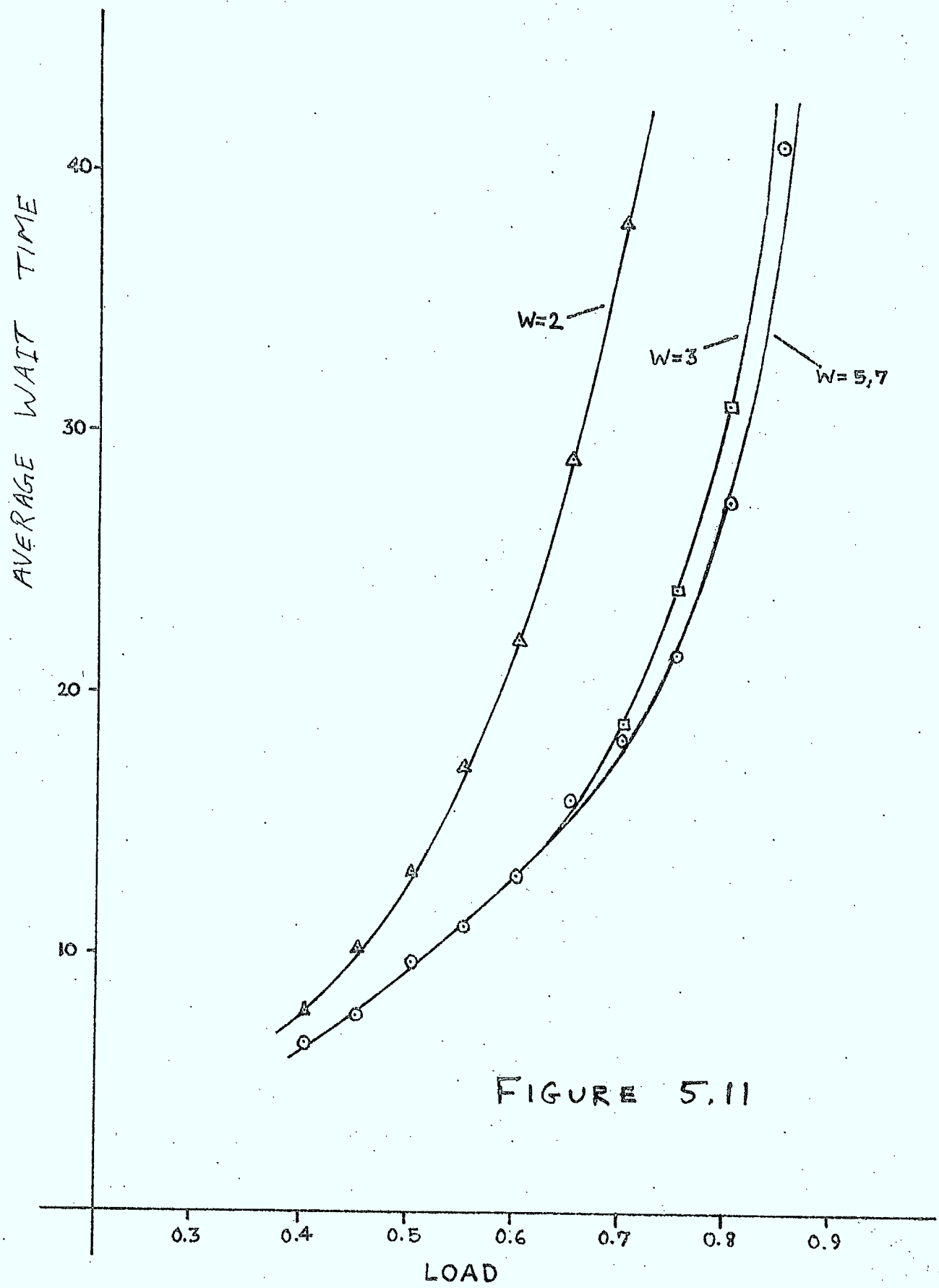


FIGURE 5.11

TOKEN PASSING

10,000 SAMPLES

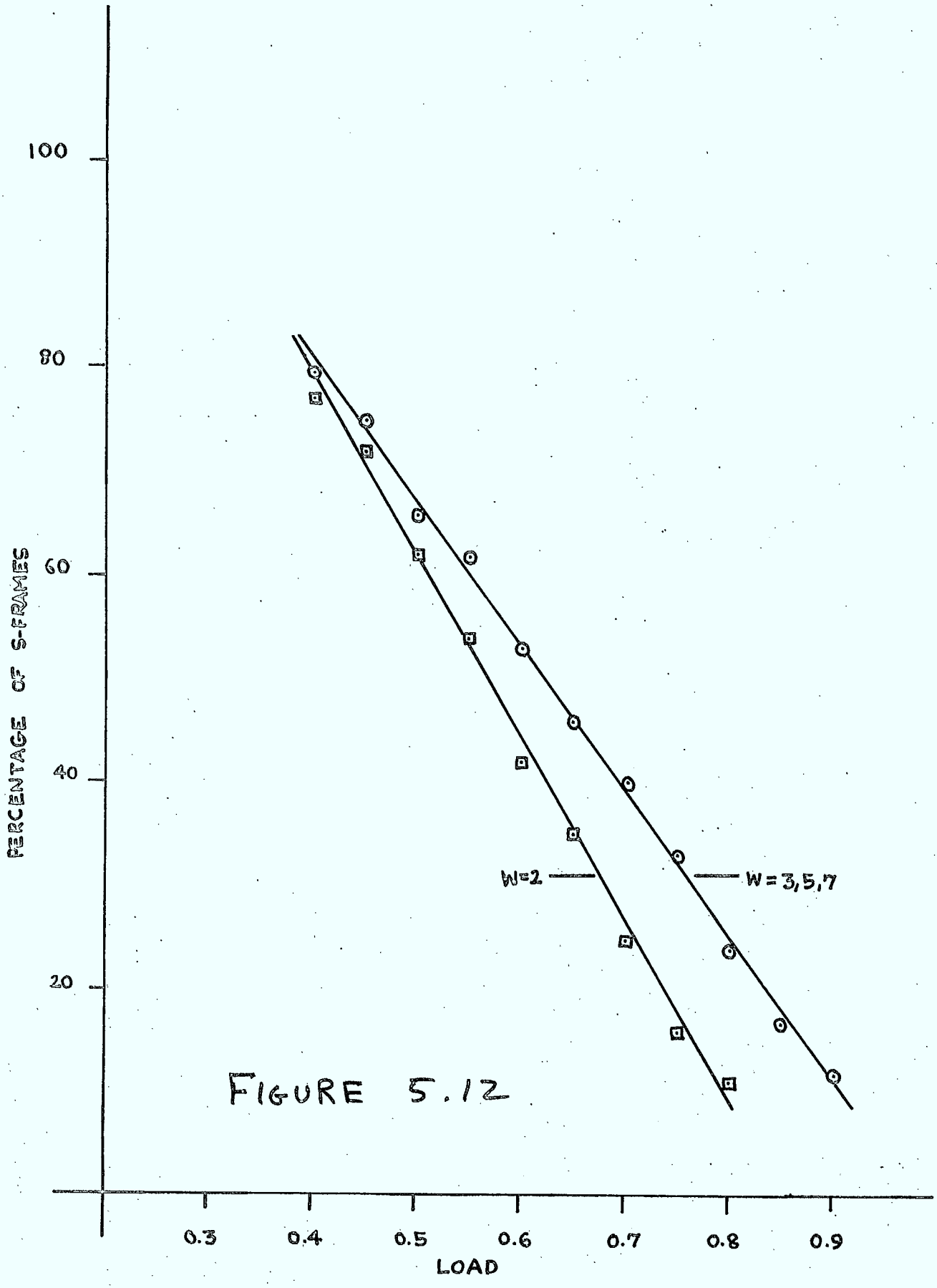
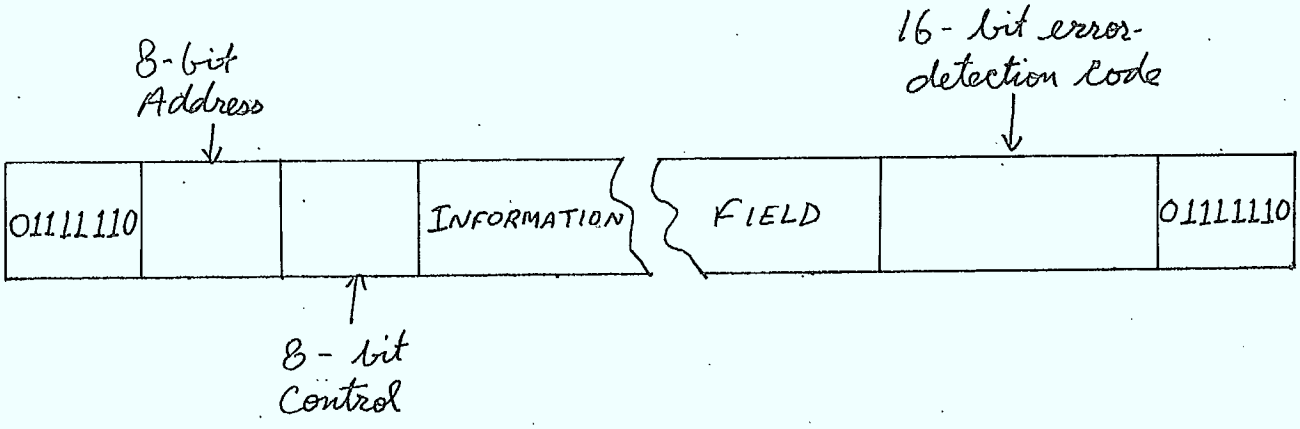

100
80
60
40
20
PERCENTAGE OF S-FRAMES
W=2
W = 3,5,7
0.3
0.4
0.5
0.6
0.7
0.8
0.9
LOAD

FIGURE 5.12



AN HDLC FRAME

I-FRAME	0	SEND SEQUENCE NUMBER, N_S	P/F	RECEIVE SEQ. NUMBER, N_R	
S-FRAME	1	0	SUPERVISORY COMMANDS	P/F	RECEIVE SEQ. NUMBER, N_R
U-FRAME	1	1	UNNUMBERED COMMANDS	P/F	UNNUMBERED COMMANDS

CONTROL BYTE

FIGURE 5.13

VI. Optical Fiber

The existing Local Area Network techniques have been developed in connection with metallic media, either twisted pair or coaxial cable. The Token Passing technique, which originated with twisted pairs in the ring configuration is easily adapted to coaxial cable. The CSMA technique which had its origins in the ALOHA radio systems is also well suited to coaxial cable. With the rapid development of optical fiber technology it is natural to study the kinds of access techniques that are appropriate.

Optical fiber[43,44] possesses a number of features that make it attractive as a transmission medium. It has low transmission loss: a realizable standard is 4db per kilometer. It appears that for single mode operation losses of 1db per kilometer are attainable. Thus distances in the order of kilometers are supportable for point-to-point operation. A second large advantage of optical fiber is high bandwidth. Data rates up to 20 M bps seem to be easily attainable. Recently we have learned of a fully operational experimental system which supports a 50 M bps data stream using LED as a source. Other properties which are of value in certain applications are immunity to electromagnetic interference. Due to fiber's insulating property there is no need for electrical insulation as a safety precaution. Finally optical fibers are small. A bundle of fibers with enormous capacity can be placed in a relatively small space.

The salient disadvantage of fiber for LAN's is that it is inherently a point-to-point medium. In order for fiber to be a multiple

access medium such as coaxial cable or radio, low loss taps on the line would be required. However with current technology the loss of a passive T is from 3 to 6db . As we shall see this loss severely limits the number of allowable access points. Also in order to bring a new station on the system it is necessary to interrupt service. In a sense the nonconductivity of fiber is something of a disadvantage from the point-of-view of reliability. It is desirable, as in ordinary telephone service, to power stations through the communications medium. In order to do this in fiber systems a separate copper wire should parallel the optical fiber.

The fact that only a limited number of access points can be put on the fiber is illustrated by the following example. As specified by the 802 standard the average power should be greater than 10 μW(-20dbm). In order for reception to be reliable, i.e., 10^{-10} bit error rate; the average received power must be 0.1 μW(-40dbm). Thus with six passive taps almost all of the margin is used up in the best case. There is almost nothing left for fiber attenuation and attenuation due to splicing and coupling. Taps with attenuation in the range .1 - .2db have been discussed as a future possibility. This together with parallel development of the other components would change the current assessment.

Apart from the attenuation of passive taps there is another significant disadvantage. As we have seen there is considerable improvement in random access techniques if the line can be continuously monitored for collisions. With the present technology this continuous

monitoring during transmission is not possible since there is large reflected energy on transmission.

In applying CSMA to fiber systems the roundtrip delay in the system comes into play. Suppose that the roundtrip delay is due to 2km of cable. If the light velocity is .8 of the free space velocity the roundtrip delay is greater than 8 μsec. This is the minimum time required to transmit a message since it is necessary to see if a collision has occurred. Note at a rate of 50 M bps a 400 bit message can be transmitted in 8 μ seconds. Any shorter length messages are transmitted inefficiently.

From these considerations it seems that fiber systems with a bus topology and the CSMA access protocol are not feasible with current technology. A ring topology with digital demultiplexing at each of the nodes is certainly feasible since it is really a series of point-to-point connections. The idea of an optical fiber ring has been implemented in a system which is aptly called HALO .

The topology which seems to be of greatest current interest is the star connection. With the star topology the number of connections and regeneration points is minimized. Reliability is improved since only the center of the star may need a guarded power supply for system reliability. If the terminals at the point of the star fail presumably system operation would be unimpaired. This stands in direct contrast to the ring topology since failure of ring nodes may bring

the entire system down. Reliability may be further improved by the use of a passive component at the center of the star. Such a component, for example, may be a reflector.

In summation it appears that the role of optical fiber in LAN is confined to high speed, limited access applications. This may be ideal for connecting LAN's to one another, to high speed processors and to the public switched network. In the interim report we spoke of such configurations as Networks of Local Area Networks. It seems that for connecting a large number of low speed users to a common line a bus configuration with some form of CSMA protocol may be optimum. Two suggested configurations are shown on Figures 6.1 and 6.2, where respectively the ring and star topologies are used to tie together LAN's . Obviously a key element in either configuration is the black box interfacing the LAN and the inner star or ring. As discussed in the interim report it seems that it would be appropriate to place higher level protocols such as HDLC at this point.

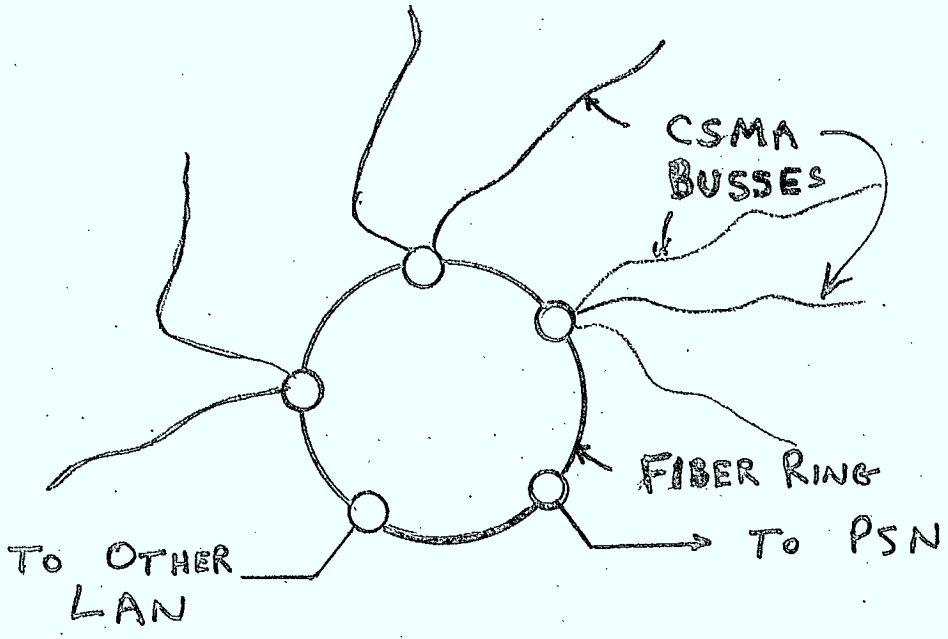


FIGURE 6.1

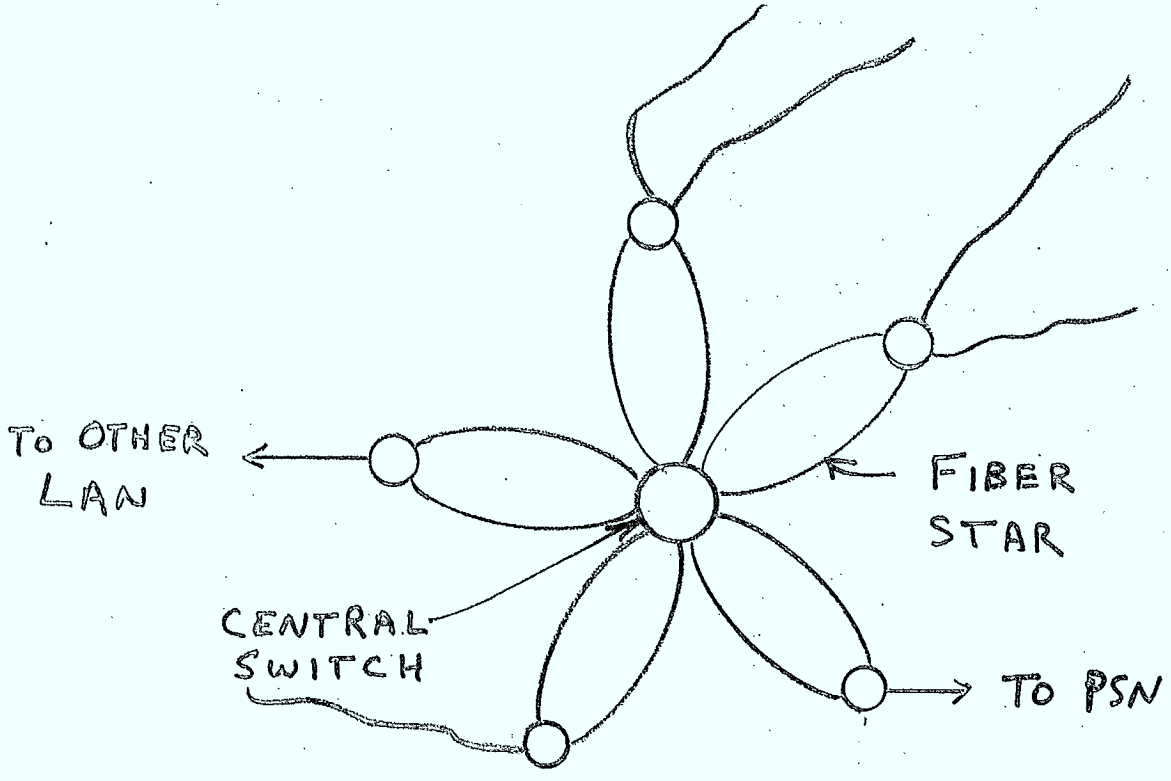


FIGURE 6.2

VII. Summary and Future Directions

In this last section of the report we shall summarize the work and we shall chart the direction of our effort in the coming year. Section III of the report dealt with ring systems. In this section we considered alternatives to Token Passing. In particular we compared the mean delay of Token Passing with the buffer insertion technique. The results show that buffer insertion shows significant improvement in performance. The analysis that is available is limited to average delay. During the coming period we shall attempt to find higher moments and the probability distribution of delay for buffer insertion.

In section IV we considered line accessing techniques which are appropriate to bus networks. We began with a summary of a previous study comparing CSMA with Token Passing. The difficulty is that the model is rather limited. Substantial effort has been expended in finding more general models for CSMA. We expect this effort to bear fruit shortly. We have also done a good deal of work on a simulation program which will be yielding results very soon. In section IV we considered the application of a tree search technique for conflict resolution in CSMA systems. Comparisons with random retransmission systems show the technique to be promising particularly if distance between stations are taken into account. In the coming period this work will be continued by means of simulation and analytical techniques. The objective will be a full evaluation of alternatives to random retransmission.

The HDLC protocol is the subject of section V of the report. A simulation program has been written to evaluate the performance of HDLC in Local Area Networks. Results have been obtained on window size for models of CSMA and Token Passing systems. The program deals with only two terminals. In the coming period the simulation program will be extended. A basic thrust here is to form a link with simulation programs modeling line access techniques.

In section VI of the report is a qualitative discussion of the role optical fiber in Local Area Networks is assessed. The salient result here is that the star configuration may be the most appropriate. We shall continue work in this area in two directions. An analysis of the behavior of star systems and comparison with the ring configuration would be of interest. Secondly we shall attempt to build, on a modest scale, a laboratory model of an optical fiber data communications system.

Acknowledgements

It is with pleasure that the author expresses his appreciation to those who participated in the work presented in this report. V. Agarwal acted as a consultant for computer matters. A. Jacobsen performed most of the computations in sections III and IV, and participated in formulating the models. P. Pownall and T. Venster wrote the HDLC simulation program discussed in section V . The results on propinquity in section IV were obtained by N. Karimi. Finally the material in section VI was researched by G. Legaria from the National Technical University of Mexico, who is spending a sabbatical at McGill.

References

1. L.G. Roberts and B.D. Wessler, "Computer Network Development to Achieve Resource Sharing," Proc. SJCC 1970, pp. 543-549.

2. W.D. Farmer and E.E. Newhall, "An Experimental Distributed Switching System to Handle Bursty Computer Traffic," Proc. ACM Symp., Problems Optimization of Data Commun. Syst., pp. 1-34, Pine Mountain, Ga., October 1969.

3. N. Abramson, "The ALOHA - System - Another Alternative for Computer Communications," in Fall Joint Comput. Conf. AFIPS Conf. Proc., Vol. 37, pp. 281-285.

4. R.M. Metcalfe and D.R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," Commun Ass. Comput. Mach., Vol. 19, pp. 395-403, 1976.

5. J.F. Hayes and D.N. Sherman, "A Study of Data Multiplexing Techniques and Delay Performance," Bell System Tech. J., Vol. 51, pp. 1985-2011, November 1972.

6. H. Zimmerman, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," IEEE Trans. on Comm., Vol. Com. 28, No. 4, pp. 425-433, April 1980.

7. IEEE 802 Local Network Standard and Draft 1, July 4, 1981.

8. W. Bux et al., IEEE Project 802 Local Area Networks Traffic Handling Characteristics Committee Report Working Draft.

9. N.M. Denkin, IEEE Standard and Specifications for Local Network Man Medium Interface and Medium Optical Fiber Baseband System, IEEE Media Subcommittee, April 27th, 1981.

10. J.R. Pierce, "Network for Block Switches of Data," Bell System Tech. J., Vol. 51, No. 6, pp. 1133-1145, July/August 1972.

11. J.R. Pierce, "How Far Can Data Loops Go?" IEEE Trans. on Comm., Vol. Com. 20, No. 3, pp. 527-530, June 1972.

12. W. Bux et al., "A Reliable Token-Ring-System for Local Area Communication," National Telecommunications Conference, pp. A2.2.1-2.2.6, New Orleans, La., December 1981.

13. A.G. Fraser, "Spider - A Data Communications Experiment," Computing Science Tech. Rep., No. 23, Bell Labs., Murray Hill, U.S.A. (1974).

14. M.T. Liu and C.C. Reams, "Message Communication Protocol and Operating Systems Design for the Distributed Loop Computer Network (DLCN)," Proc. 4th Annual Symposium on Computer Architecture, pp. 193-200, March 1977.

15. M.V. Wilkes, "Communication Using a Digital Ring," PAC NET Conf., Sendai, Japan, August 1975.

16. E.R. Hafner et al., "A Digital Loop Communication System," IEEE Trans. Comm., Vol. Com. 22, No. 6, pp. 877-881, June 1974.

17. Z.G. Vranesic et al., Tornet: A Local Area Network, Proc. 7th Data Communications Symposium, Mexico City, October 1981.

18. W.J. Kropfl, "An Experimental Data Block Switching System," Bell Syst. Tech. J., Vol. 51, No. 6, pp. 1147-1165, July/August 1972.

19. J.F. Hayes and D.N. Sherman, "Simulated Performance of a Ring Switched Data Network," Bell Syst. Tech. J., Vol. 50, No. 9, pp. 2947-2978, November 1971.

20. R.R. Anderson et al., "Simulated Performance of a Ring Switched Data Transmission System," IEEE Trans. Comm., Vol. Com. 20, No. 3, pp. 576-591, June 1972.

21. P. Zafiropulo, "Reliability Optimization in Multiloop Communication Networks," IEEE Trans. Comm., Com. 21, No. 8, pp. 898-907, August 1973.

22. P. Zafiropulo, "Performance Evaluation of Reliability Improvement Techniques for Single Loop Communication Systems," IEEE Trans. Comm., Vol. 22, No. 6, pp. 742-751, June 1974.

23. R.J. Camrass and R.G. Gallager, "Encoding Message Lengths for Data Transmission," IEEE Trans. Inform. Theory, Vol. IT-24, July 1978.

24. HK-W. Lau, "Data Multiplexing: Many Lightly Loaded Sources," Master's Thesis, McGill University 1981.

25. O. Hashida, "Analysis of Multiqueue," Rev. Elect. Comm. Lab., NTT, Vol. 20, Nos. 3 and 4, pp. 189-199, March/April 1972.

26. W. Bux and M. Schlatter, "An Approximate Method for the Performance Analysis of Buffer Insertion Rings," to be published IEEE Trans. on Comm.

27. R.B. Cooper, Introduction to Queueing Theory, Macmillan, 1972, p. 156.

28. N. Abramson, "The ALOHA System - Another Alternative for Computer Communications," in 1970 Fall Joint Comput. Conf. AFIPS Conf. Proc., Vol. 37, pp. 281-285.

29. S.S. Lam and L. Kleinrock, "Packet Switching in a Multiaccess Broadcast Channel: Performance Evaluation," Vol. Com. 23, No. 4, pp. 410-423, April 1975.

30. A.B. Cardeal and M.E. Hellman, "Bistable Behavior of ALOHA - type Systems," IEEE Trans. Comm., Vol. 23, No. 4, pp. 401-410, April 1975.

31. L.G. Roberts, "ALOHA Packet System with and without Slots and Capture," Computer Comm. Rev., Vol. 5, pp. 28-42, April 1975.

32. F.A. Tobagi and L. Kleinrock, "Packet Switching in Radio Channels Part 1: Carrier Sense Multiple Access Modes and Their Throughput Delay Characteristics," Vol. Com. 23, No. 12, pp. 1400-1416, December 1975.

33. F.A. Tobagi and V.B. Hunt, "Performance Analysis of Carrier Sense Multiple Access with Collision Detection," Computer Networks, Vol. 4, No. 5, pp. 245-259, November 1980.

34. A.R. Kaye, "Analysis of a Distributed Control Loop for Data Transmission," in Proc. Symp. Comput. Comm.: Network Teletraffic, Polytech. Inst. of Brooklyn, Brooklyn, N.Y., April 1972.

35. S.S. Lam, "A Carrier Sense Multiple Access Protocol for Local Networks," Computer Networks, Vol. 4, No. 1, pp. 21-32.

36. J.F. Hayes, "An Adaptive Technique for Local Distribution," IEEE Trans. Comm., Vol. Com. 26, No. 8, pp. 1178-1186, August 1978.

37. A. Grami and J.F. Hayes, "Delay Performance of Adaptive Local Distribution," Proc. ICC'80, Seattle, Wa., pp. 39.4.1 - 39.4.5, June 1980.

38. J. Capetanakis, "Tree Algorithms for Packet Broadcast Channels," IEEE Trans. Inform. Theory, Vol. IT-25, pp. 1178-1186, September 1979.

39. N. Pippenger, "Bounds on the Performance of Protocols for a Multiple Access Broadcast Channel," IEEE Trans. Inform. Theory, Vol. IT-27, No. 2, March 1981.

40. P.A. Hamblet and J. Mosely, "Efficient Accessing of a Multiaccess Broadcast Channel," IEEE Conf. Decision and Control, Albuquerque, December 1980.

41. J. Labetoulle and G. Pujolle, "HDLC Throughput and Response Time for Bidirectional Data Flow with Nonuniform Frame Sizes," IEEE Trans. Computers, Vol. C-30, No. 6, pp. 405-413, June 1981.

42. W. Bux et al., "Balanced HDLC Procedures: A Performance Analysis," IEEE Trans. Comm., Vol. Com. 28, No. 11, pp. 1889-1898, November 1980.

43. "Guided Optical Communications," Proc. Society of Photo-Optical Instrumentation Engineers, Vol. 63, August 19-20, 1975, San Diego.

44. E.G. Rawson and R.M. Metcalfe, "Fibernet: Multimode Optical Fibers for Local Computer Networks," IEEE Trans. Comm., Vol. Com. 26, No. 7, pp. 983-990, July 1978.

```
program network (input, output, stats, sumdat, busy);

{ ------------------------------------------------- }
{ computer network simulation using hdlc protocol  }
{ for abstract see P. Pownall or T. Ventser        }
{ ------------------------------------------------- }

const
{ ************************************************** }
{                                                    }
{ The following are the system parameters which determine  }
{ the performance of the system. At the moment, we have a  }
{ two terminal system.  Each of the terminals has a send   }
{ buffer and a receive buffer.  The send buffer is controlled}
{ using the pointers sendbuff, sendseq, and nextavail ..   }
{ The two terminals have independent arrivals, and will    }
{ piggyback acknowledgements using the nr field on the info }
{ frames they send.  If their send buffer is empty, an     }
{ s-frame is constructed and transmitted.                  }
{                                                    }
{ ************************************************** }

   t = 2;                                  { no. of terminals }
   buffsiz = 8;                            { send buffer size }
   n = 99;                          { length of arr/del/err q's }

{ ************************************************** }
{                                                    }
{ The following describes the types of variables that are  }
{ used in the program.  The main data structure is:        }
{                                                    }
{   record : This is the standard record (analogous to a   }
{            frame) which is passed through the simulation  }
{            system.  The fields are described below.      }
{                                                    }
{ ************************************************** }

type
   frametype = ( sframe, iframe );                  { info or s-frame  }
   queue = array [1..t, 0..n] of real;              { queue type       }
   lists = array [1..t] of integer;                 { pointers         }
   eventype = ( arr, tra, rec, tim );               { 4 event types    }
   string = packed array [1..11] of char;
   link = ↑object;                                  { pointer          }
   object = record                            { the record fields :   }
      toaddr : integer;                       { to address            }
      frame : frametype;                      {          kind of frame }
      ns, nr : integer;                       {  send and rec seq nos }
      delay, error : real;                    {    delay, error times }
      arrivtim, sendtim, transtim,
      recevtim, fintim, acktim : real;        { these are the time fields }
      next : link;                            {    next pointer field }
   end;
   card = array[1..t, 1..9] of real;           { summary file line    }
   cardfile = file of card;                    { summary file type    }

{ ************************************************** }
{                                                    }
{ The actual variable names are outlined in this section.  }
{                                                    }
```

```
{ ********************************************************************** }

var
   i, j, mainloop : integer;                    { loop counters          }
   stats : text;                                { the output stats       }
   busy : text;                                 { the dumpout file       }
   lastct, hlcount : integer;                   { counts for dumpout     }
   alambda, dlambda : real;                     { arrival, delay means   }
   elambda : real;                              { error probability      }
   tiproc, tsproc, tout, framenum : real;       { processing and timout  }
   runtim : real;
   seed, sizwin : integer;                      { random seed, window    }
   clocktim : real;                             { the current time       }
   term : integer;                              { the current terminal   }
   arrivq, proq, errorq : queue;                { the queues             }
   nextarr, nextpro, nexterr : lists;           { pointers to queues     }
   sendbuff, nextsend, bottom : array [1..t] of link;
                                                {  ptrs for send buffer  }
   bufferfull : array [1..t] of boolean;        { buffer full flag       }
   transbuf, recbuff : array[1..t] of link;     { pointers to buffers    }
   recno, sendseq : lists;                      { rec & send #s,         }
   timout : array [1..t] of real;               { timout timer           }
   event : eventype;                            { current event type     }
   intranstop : link;                           { pointer to in-trans q  }
   todays_date, current_time : string;
   isum : array[1..t, 1..8] of real;            { stats running sums     }
   ssum : array[1..t, 1..4] of real;            { s-frame sums           }
   scount, count : array[1..t] of real;         { frame counts  }
   buff : card;                                 { summary file buffer    }
   sumdat : cardfile;                           { the summary file       }
   opt1, opt2, opt3, opt4 : integer;            { simulation options     }
   ropt1, ropt2, increments, load : real;
   freelist : link;                             { free record list ptr   }


function mth$random (seed : integer) : real; extern;
{ ****************************************** }
{ this is the system random number generator }
{ yields a uniform distribution in [0,1)     }
{ ****************************************** }


procedure histogram( term : integer ; event : eventype );
{**************************************************************************}
{                                                                          }
{  This procedure prints a histogram!                                      }
{                                                                          }
{**************************************************************************}

var
   i, j, timeless1, count : integer;
   traverse : link;

begin
   timeless1 := round(clocktim - 1.0);
   if (timeless1 < lastct) then timeless1:=lastct;
   for i := lastct to timeless1 do
   begin
      if (hlcount > 49) then
      begin
```

```
      page(busy);
      hlcount := 0;
      writeln(busy,'TERMINAL 1':25,'TERMINAL 2':40);
      writeln(busy,'----------':25,'----------':40);
      writeln(busy);
      writeln(busy,'CLOCKTIME':9,'SENDBUFF':10,'TRANSBUFF':11,
              'INTRANSIT QUEUE':17,'TRANSBUFF':11,'SENDBUFF':10);
      for j := 1 to 70 do write(busy,'-');
      writeln(busy)
   end;
   write(busy,i:7,'!':5);
   traverse := sendbuff[1];
   count := 1;
   while (traverse <> nil) and (count < 12 ) do
   begin
      if (traverse↑.sendtim < clocktim) and (traverse↑.sendtim > -1)
      then
         if (traverse↑.ns = -1)
         then write(busy,'S')
         else write(busy,traverse↑.ns:1)
      else write(busy,'*');
      traverse := traverse↑.next;
      count := count + 1
   end;
   for j:= count to 12 do write(busy,' ');
   if transbuf[1] = nil
   then write(busy,'!':12)
   else if (transbuf[1]↑.ns = -1)
      then write(busy,'S','!':11)
      else write(busy,transbuf[1]↑.ns:1,'!':11);
   traverse := intranstop;
   count := 1;
   while (traverse <> nil) and (count < 17) do
   begin
      if (traverse↑.toaddr = 1)
      then write(busy,'1')
      else write(busy,'2');
      traverse := traverse↑.next;
      count := count + 1
   end;
   for j := count to 17 do write(busy,' ');
   if transbuf[2] = nil
   then write(busy,'!':8)
   else if (transbuf[2]↑.ns = -1)
      then write(busy,'S','!':7)
      else write(busy,transbuf[2]↑.ns:1,'!':7);
   traverse := sendbuff[2];
   count := 1;
   while (traverse<>nil) and (count < 12) do
   begin
      if (traverse↑.sendtim < clocktim) and (traverse↑.sendtim > -1)
      then
         if (traverse↑.ns = -1)
         then write(busy,'S')
         else write(busy,traverse↑.ns:1)
      else write(busy,'*');
      traverse := traverse↑.next;
      count := count + 1
   end;
   writeln(busy);
```

```
      hlcount := hlcount + 1
   end
end;


procedure create (var ptr : link);
{*****************************************************************}
{ This procedure keeps a list of discarded records and gives them  }
{ out when needed.                                                 }
{*****************************************************************}
begin
   if freelist = nil
      then new(ptr)
      else
         begin
            ptr := freelist;
            freelist := freelist^.next
         end
end;


procedure release (ptr : link);
{*****************************************************************}
{ This procedure places a discarded record back into the list of   }
{ free records.                                                    }
{*****************************************************************}
begin
   if (ptr <> nil) then
   begin
      ptr^.next := freelist;
      freelist := ptr
   end
end;


procedure emptybuffer (var rec : link);
{*****************************************************************}
{* This procedure is used at the end of a run of a load to release the   *}
{* memory which happened to be in use when time ran out.                 *}
{*****************************************************************}

var
   temp : link;

begin
   while (rec <> nil) do
   begin
      temp := rec;
      rec := rec^.next;
      release(temp)
   end
end;



procedure fillarr ( term : integer );
{ ********************************************************** }
{ this procedure uses a Poisson interarrival time distribution to  }
{ refill the arrival queue. The formula used for the seed value is  }
{ arbitrary. Note that absolute times are calculated from the inter }
```

```
{ arrival times.                                                    }
{ ************************************************************** }

var
   i : integer;
   lasttime,randnum : real;

begin
   lasttime := arrivq[term,n];                    { last value offset    }
   if ( term=2 ) and ( opt4 = 0 ) then            { for single server,   }
      for i := 0 to n do arrivq[term,i]:=runtim   { no arrivals for 2    }
   else
   for i := 0 to n do
   begin
      randnum := mth$random(seed);                { get random no.       }
      arrivq[term,i] := lasttime + abs( ln( 1-randnum ) ) * alambda ;
                                                  { absolute arr time    }
      lasttime := arrivq[term,i]                  { advance last value   }
   end
end; { fillarr }


procedure fillpro ( term : integer );
{ ************************************************************** }
{ This proc fills the tiproc queue according to the              }
{ distribution specified by option 1 (opt1).                     }
{ opt1 = 0      :  constant (zero) distribution                  }
{ opt1 = 1      :  CSMA distribution                             }
{ opt1 = 2      :  Token Passing distribution                    }
{ NOTE: AT THIS TIME THIS IS A CRUDE APPROX.  WE WILL            }
{ IMPROVE THIS LATER.                                            }
{ ************************************************************** }

var
   i,j,sumq,trys,tryq : integer;
   p0,q0,randnum : real;

begin
   if (opt1=0) then            { for constant server, the processing }
      for i := 0 to n do       { time is just tiproc or tsproc, and  }
         proq[term,i] := 0.0   { is done in SEND.                    }
   else
   begin
      if (opt1=1) then         { for csma server, we generate a var  }
      for i := 0 to n do       { which is geometric sum of geometric }
      begin                    { random variables. For the exact     }
         p0 := sqrt(0.01);     { formula, see the literature         }
         randnum := mth$random(seed);   { note that we generate a    }
         trys := 1;                     { distribution in 0..100     }
         while not (randnum < p0) do    { and then scale down        }
         begin
            randnum := mth$random(seed);
            trys := trys + 1
         end;
         sumq := 0;
         for j := 1 to trys do
         begin { geometric - inter-trial time }
            q0 := sqrt(0.01);
            randnum := mth$random(seed);
            tryq := 1;
```

```
            while not (randnum < q0) do
            begin
               randnum := mth$random(seed);
               tryq := tryq + 1
            end;
            sumq := sumq + tryq
         end;
         proq[term,i] := sumq*0.08      { scale down to 8.0 mean  }
      end
      else
      for i := 0 to n do
      begin                             { for token passing, we generate }
         q0 := 0.01;                    { a geometric r.v. in 0..100,    }
         randnum := mth$random(seed);   { and add a constant plus tiproc }
         tryq := 1;                     { or tsproc respectively.        }
         while not (randnum < q0) do    { Again, see lit for details     }
         begin
            randnum := mth$random(seed);
            tryq := tryq + 1
         end;
         proq[term,i] := tryq*0.02 + 6.0       { scale to 8.0 mean again }
      end
   end
end; { fillpro }


procedure fillerr ( term : integer );
{ ************************************************************************ }
{ This proc fills the error queue with 0 or 1.  0 indicates OK, 1 is in    }
{ error.  Elambda is 1 minus the error probability .                       }
{ ************************************************************************ }
var
   i : integer;

begin
   for i := 0 to n do
      if (mth$random(seed) > elambda) then errorq[term,i] := 1
      else errorq[term,i] := 0
end; { fillerr }


function windowcheck (newr,left : integer):boolean;
{ ************************************************************************ }
{ Checks if a number is within a window of size sizwin.                    }
{ ************************************************************************ }

var
   i : integer;

begin
   windowcheck := false;                                  { assume false }
   if (left <> -1) then
      for i := 0 to (sizwin-1) do
         if ( (left + i)mod buffsiz = newr ) then windowcheck := true
end; { windowcheck }


procedure findnextevent (var term : integer; var event : eventype );
{ ************************************************************************ }
```

```
{ This proc searches the lists in order to find the event which occurs  }
{ next.  It does this by comparing the times; obviously the next event  }
{ time is less than the runtim, so we assume the next time is runtim to }
{ begin with.  We check each terminal consecutively, so terminal one has }
{ passive priority on events that occur simultaneously.                 }
{ ********************************************************************* }

var
   i : integer ;
   high : real;

begin
   high := runtim;                          { assume next is runtim      }
   for i := 1 to t do begin
      { -- ARRIVAL QUEUE -- }
      if ( arrivq[i,nextarr[i]] < high ) then
      begin
         high := arrivq [i,nextarr [i]];    { set high to arr time       }
         term := i;                         { set term to i              }
         event := arr                       { set event to arrival       }
      end;
      { -- TRANSMIT buffER -- }
      if ( transbuf[i] <> nil ) then        { check nonempty             }
         if ( transbuf[i]↑.transtim < high ) then begin
            high := transbuf[i]↑.transtim;  { set high to trans time     }
            term := i;                      { set term to i              }
            event := tra                    { set event to trans         }
         end;
      { -- TIME-OUT -- }
      if ( timout[i] < high ) then
      begin
         high := timout[i];                 { set high to timout time    }
         term := i;                         { set term to i              }
         event := tim                       { set event to timout        }
      end
   end;
   { -- IN-TRANSIT QUEUE -- }
   if ( intranstop <> nil ) then            { check nonempty             }
      if ( intranstop↑.recevtim < high ) then
      begin
         high := intranstop↑.recevtim;      { set high to recevtim       }
         term := intranstop↑.toaddr;        { set term to destination    }
         event := rec                       { set event to receive       }
      end;
   clocktim := high                         { adjust clocktim            }
end; { findnextevent }



procedure log (term : integer; var rec : link );
{ ********************************************************************* }
{ This procedure updates the running sums which are being kept to       }
{ track the performance of the system.                                  }
{ ********************************************************************* }

begin
   if (rec↑.frame = iframe) then
   with rec↑ do
   begin
      isum[term,1] := isum[term,1] + (sendtim - arrivtim);
```

```
      isum[term,2] := isum[term,2] + (transtim - sendtim);
      isum[term,3] := isum[term,3] + (recevtim - transtim);
      isum[term,4] := isum[term,4] + (fintim - recevtim);
      isum[term,5] := isum[term,5] + (fintim - arrivtim);
      isum[term,6] := isum[term,6] + (acktim - fintim);
      isum[term,7] := isum[term,7] + (acktim - arrivtim);
      isum[term,8] := arrivtim;
      count[term] := count[term] + 1.0
   end
   else
   with rect do
   begin
      ssum[term,1] := ssum[term,1] + (sendtim - arrivtim);
      ssum[term,2] := ssum[term,2] + (transtim - sendtim);
      ssum[term,3] := ssum[term,3] + (recevtim - transtim);
      ssum[term,4] := ssum[term,4] + (fintim - arrivtim);
      scount[term] := scount[term] + 1.0
   end
end; { log }


procedure send ( term : integer );
{ ********************************************************************** }
{ this procedure takes a record from the send buffer, if any exists }
{ and places it into the transmission buffer. At the same time it   }
{ calculates the sendtim at which the record will be completely     }
{ transmitted. Note that a new physical record is formed in order   }
{ to keep physical records in the send buffer.                      }
{ Note that we only send if the transbuff is empty.                 }
{ ********************************************************************** }

var
   p, temp, traverse : link;

begin
   if (transbuf[term] = nil) then
   begin
     if ( nextsend [term] = nil )
     then                                              { nothing to send }
        if ( sendbuff[term] = nil )                    { if empty,       }
        then timout[term] := clocktim + tout           { extend timout   }
        else { do nothing }
     else
     begin
        if ( windowcheck ( nextsend [term]↑.ns, sendbuff [term]↑.ns ) )
           or ( nextsend[term]↑.ns = -1 ) then
        begin                                          {   inside window }
           create (p);                                 { create new physical loc }
           p↑ := nextsend [term]↑;                     {     copy buffer }
           nextsend[term]↑.sendtim := clocktim;        {t}
           if (nextsend[term]↑.frame = sframe ) then   { del sframe buff }
           begin
              if (nextsend[term] = sendbuff[term]) then
              begin                                    { at top          }
                 if (nextsend[term] = bottom[term]) then
                    bottom[term] := nil;
                 temp := sendbuff[term];
                 sendbuff[term] := sendbuff[term]↑.next;
                 nextsend[term] := nextsend[term]↑.next;
                 release(temp)
```

```
            end
            else                                    { within buffer    }
            begin
               traverse := sendbuff[term];
               while ( traverse↑.next <> nextsend[term] ) do
                  traverse := traverse↑.next;
               traverse↑.next := nextsend[term]↑.next;
               if ( nextsend[term] = bottom[term] ) then
               begin
                  release(nextsend[term]);
                  nextsend[term] := nil;
                  bottom[term] := traverse
               end
               else
                  begin
                  temp := nextsend[term];
                  nextsend[term] := nextsend[term]↑.next;
                  release(temp)
                  end
            end
         end
         else                                    { i-frame                  }
            nextsend [term] := nextsend [term]↑.next;  { move pointer }
         p↑.nr := recno [term];                 { fill rec no field        }
         p↑.delay := dlambda;                   { delay field              }
         p↑.error := errorq[term,nexterr[term]];   { error field           }
         if (nexterr[term] = n) then fillerr(term);
         nexterr[term] := (nexterr[term] + 1) mod (n+1);
         p↑.sendtim := clocktim;                { send time field          }
         if (p↑.frame = iframe) then
            p↑.transtim := clocktim + proq[term,nextpro[term]]+tiproc
         else
            p↑.transtim := clocktim + proq[term,nextpro[term]]+tsproc;
         if (nextpro[term] = n) then fillpro(term);
         nextpro[term] := (nextpro[term] + 1) mod (n+1);
         transbuf[term] := p                    { assign pointer           }
      end
      else                                  { do nothing - outside window }
   end
   else                                  { do nothing - transbuf occupied }
end; { send }


procedure arrival (term : integer );
{ ************************************************************************ }
{ this procedure takes an arrival time from the arrival queue, builds a }
{ record and places it into the send buffer for that terminal.           }
{ ************************************************************************ }

var
  p : link;
begin
   create(p);                                   { create new rec     }
   with p↑ do                                   { next avail record  }
   begin
      toaddr := (term mod t) + 1;               { to terminal        }
      ns := sendseq[term];                      { sendseq no         }
      sendseq[term] := (sendseq[term] + 1) mod buffsiz;
      frame := iframe;                          { frame type         }
```

```
      arrivtim := arrivq[term,nextarr[term]];      { arrival time      }
      sendtim := -1;                               {t}
      next := nil
   end;
   if (nextarr[term] = n) then fillarr(term);      { refill if necc }
   nextarr[term] := (nextarr[term] + 1) mod (n+1);    { advance ptr }
   if (bottom[term] = nil) then bottom[term] := p
   else
   begin
      bottom[term]↑.next := p;                     { add to end of the  }
      bottom[term] := p
   end;
   if (sendbuff[term] = nil) then sendbuff[term] := p;
   if (nextsend[term] = nil) then nextsend[term] := p;
   send (term)                              { check if we can send      }
end; { arrival }


procedure ssframe (term : integer ; stype : integer);
{ ************************************************************************ }
{ this proc places an s-frame into the send buffer of the terminal       }
{ specified and fills in the appropriate fields.                         }
{ Note that sendseq does not change since that field is used to          }
{ identify the s-frame command (stype).                                  }
{ ************************************************************************ }

var
   p : link;

begin
   create(p);                                   { create new record      }
   with p↑ do                                   { fill in the fields     }
   begin
      toaddr := (term mod t) + 1;               { to other terminal      }
      arrivtim := clocktim;                     { arrival time           }
      frame := sframe;                          { frame type is s-frame  }
      ns := stype;                              { ns field is s type     }
      nr := recno[term];                        { request recno retrans  }
      next := nil
   end;
   if (bottom[term] = nil) then bottom[term] := p
   else
   begin
      bottom[term]↑.next := p;                     { add to buffer          }
      bottom[term] := p
   end;
   if (sendbuff[term] = nil) then sendbuff[term] := p;
   if (nextsend[term] = nil) then nextsend[term] := p;
   send(term)                                   { check if we can send   }
end; { ssframe }


procedure trans (term : integer );
{ ************************************************************************ }
{ this procedure places the record from the transmission buffer into     }
{ the in-transit queue in order of recevtim at the other terminal        }
{ we assume the record is not placed to begin with and use a             }
{ standard bubble-down algorithm.                                        }
{ ************************************************************************ }
```

```
var
   placed : boolean;
   prev,now : link;

begin
   placed := false;                          { assume not placed          }
   transbuf[term]↑.recevtim := clocktim + transbuf[term]↑.delay;
   transbuf[term]↑.next := nil;              { set pointer to nil         }
   if (intranstop = nil ) then intranstop := transbuf[term]
                                { if in-trans q is empty, then place }
   else
   begin                                  { the in-trans q is not empty }
      if (transbuf[term]↑.recevtim < intranstop↑.recevtim) then
      begin                              { place at top if lower recev tim }
         transbuf[term]↑.next := intranstop;
         intranstop := transbuf[term]
      end
      else
      begin                                 { insert into intrans queue }
         prev := intranstop;                { set prev pointer          }
         now := prev↑.next;                 { set next pointer          }
         while ( now <> nil ) do begin
            if ( transbuf[term]↑.recevtim < now↑.recevtim ) then
            begin
               prev↑.next := transbuf[term];    { correct; insert      }
               transbuf[term]↑.next := now;
               now := nil;
               placed := true                   { set indicator value }
            end
            else
            begin
               prev := now;                     { advance             }
               now := now↑.next
            end
         end;
         if not (placed) then prev↑.next := transbuf[term]  { at end }
      end
   end;
   transbuf[term] := nil;                        { empty transbuff     }
   send(term)                                   { check if we can send }
end; { trans }


procedure ack (term : integer; var rec : link );
{ ********************************************************** }
{ This proc acknowledges the receipt of a frame.  The info is  }
{ contained in the nr field; nr-1 is the number of frames the }
{ other terminal has received                                  }
{ ********************************************************** }

var
   num : integer;
   temp : link;

begin
   if ( sendbuff[term] <> nil ) then
   begin
      num:= (rec↑.nr + buffsiz - 1) mod buffsiz;          {    nr-1 using MOD }
      if ( windowcheck ( num,sendbuff[term]↑.ns )) then  {    window check    }
      begin                                              { advance left edge }
```

```
        while ( sendbuff[term]↑.ns <> num ) and
              ( sendbuff[term] <> nextsend[term] ) do
        begin
           sendbuff[term]↑.acktim := clocktim;
           temp := sendbuff[term];
           sendbuff[term] := sendbuff[term]↑.next;
           log (term, temp);
           release (temp)                          { reassign memory to freelist  }
        end;
        sendbuff[term]↑.acktim := clocktim;
        temp := sendbuff[term];
        sendbuff[term] := sendbuff[term]↑.next;       { advance ptr              }
        log (term, temp);
        release (temp);                            { reassign memory to freelist  }
        timout[term] := clocktim + tout;                 {   reset timout        }
        if ( sendbuff[term] = nil ) then
           bottom[term]:= nil;
        send(term)
     end
  end
end; { ack }


procedure buffer (term : integer; var newframe : link);
{ ************************************************************ }
{ This procedure inserts a new frame into the receive        }
{ buffer for that terminal. The process used is again the    }
{ standard bubble-down algorithm, with a check for frames    }
{ that are already in the receive buffer ( This could        }
{ occur after a timout )                                     }
{ ************************************************************ }

var
   temp, traverse : link;

begin
   create (temp);                                  { create temp location }
   if (recbuff[term] = nil) then recbuff[term] := newframe       { empty }
   else
   begin
      if (newframe↑.ns < recbuff[term]↑.ns) then      { top of queue ?     }
      begin
         newframe↑.next := recbuff[term];             { place at top       }
         recbuff[term] := newframe
      end
      else
      begin                                          { traverse the queue }
         traverse := recbuff[term];                  { set traverse pointer }
         while (newframe↑.ns > traverse↑.ns) and
               (traverse↑.next <> nil) do traverse := traverse↑.next;
         if (traverse↑.next = nil) then
         begin
            if (traverse↑.ns = newframe↑.ns) then    { already there       }
            else traverse↑.next := newframe          { place at end        }
         end
         else                                { the frame is inside the queue }
         begin
            if (traverse↑.ns = newframe↑.ns) then    { already there       }
            else
            begin
```

```
                temp↑ := traverse↑;                    { insert frame      }
                traverse↑ := newframe↑;
                traverse↑.next := temp;
                release(temp)
              end
            end
          end
        end
end; { buffer }


procedure update (term : integer ; var rec : link);
{ ************************************************************ }
{ This proc updates the fields of the frame received with      }
{ the original copy of the frame in the send buffer of the     }
{ other terminal                                               }
{ ************************************************************ }

var
    traverse : link;

begin
    traverse := sendbuff[(term mod t)+1];    { traverse the other term }
    while (traverse↑.ns <> rec↑.ns) and
          (traverse <> nil) do traverse := traverse↑.next;
    if (traverse=nil) then writeln('%%%error in update')
    else                                     { update the fields      }
    with traverse↑ do
    begin
       nr := rec↑.nr;
       delay := rec↑.delay;
       error := rec↑.error;
       sendtim := rec↑.sendtim;
       transtim := rec↑.transtim;
       recevtim := rec↑.recevtim;
       fintim := rec↑.fintim
    end
end;


procedure receive (term:integer);
{ ************************************************************ }
{ this procedure is called when the next event to occur is a   }
{ frame arriving at a terminal from the intransit queue.       }
{ The frame is removed from the intransit queue and passed     }
{ if possible. If the frame is out of sequence, it is placed   }
{ in the receive buffer. We also check for ack using the nr    }
{ field of the received frame.                                 }
{ ************************************************************ }

var
    newframe,traverse,temp : link;

begin
    newframe := intranstop;                  { take frame from in-trans }
    intranstop := intranstop↑.next;          { advance in-trans queue   }
    newframe↑.next := nil;                   { nil the next pointer     }
    if (newframe↑.error = 0) then
    begin                                    { frame is ok              }
       if (newframe↑.frame = iframe) then
```

```
    begin                                    { --- I-FRAME SECTION ---    }
       if ( windowcheck ( newframe↑.ns,recno[term] )) then
       begin                                   { sequence num within window }
          if ( newframe↑.ns = recno[term] ) then
          begin                                { pass out the frame         }
             recno[term] := (newframe↑.ns + 1) mod buffsiz; { advance recno}
             newframe↑.fintim := clocktim;     { fill fintim field          }
             update(term,newframe);            {   update rec at other term }
             ack (term,newframe);                { check for ack info      }
             release (newframe);
             while ( recbuff[term] <> nil ) do   { advance recbuff ?       }
             begin                               { advance buffer          }
               if (recno[term] = recbuff[term]↑.ns) then
               begin                             { expected frame ?        }
                  recno[term] := (recno[term]+1) mod buffsiz;
                  recbuff[term]↑.fintim := clocktim;    { fintim field     }
                  update(term,recbuff[term]);          { update other copy }
                  temp := recbuff[term];
                  recbuff[term] := recbuff[term]↑.next; { advance pointer  }
                  release(temp)
               end
             end;
             if ( nextsend[term] = nil )
             then ssframe (term,-1)            {.place s-frame with RR code }
             else                          { do nothing - no s-frame needed }
          end
          else buffer(term,newframe)    { outside - place in receive buffer }
       end
       else                                     { frame is outside window }
    end
    else                                     { --- S-FRAME SECTION --- }
    begin
       if ( newframe↑.ns = -1 ) then          { RR frame: ack only       }
       begin
          newframe↑.fintim := clocktim;       { fill in fintim field     }
          ack(term,newframe);
          log(term,newframe);                 { log the s frame          }
          release(newframe)
       end
       else                                   { REJ frame: retransmit    }
       begin
          traverse := recbuff[term];          { find the restart point   }
          if (traverse<> nil) then
          while (traverse↑.ns <> newframe↑.nr) and
                (traverse <> nil) do traverse := traverse↑.next;
          if (traverse = nil) then writeln('%%% error in rec')
          else nextsend[term] := traverse;      { reset the next pointer  }
          send(term)                          { check to send            }
       end
    end
  end
  else {* here goes the REJ instr *}    { frame in error; discard }
    ssframe(term,-9)
end; { receive }


procedure calltime (term : integer );
{ ************************************************************************ }
{ this proc deals with a time-out by recycling the send buffer for that   }
{ for that terminal. This means we set s (next to send) back to 1 (left   }
```

```
{ edge) and begin retransmitting the frames.                                }
{ ************************************************************************ }

begin
   writeln(' timeout at',clocktim:4,' for terminal',term:2);
   nextsend[term] := sendbuff[term];                    { set s back to 1 }
   send(term);                                          { send if poss    }
   timout[term] := clocktim + tout                      { reset timout    }
end; { calltime }




{************************************************************************}
{*                                                                      *}
{*------------------------- M A I N ------------------------------------*}
{*                                                                      *}
{************************************************************************}


begin { main }

   { --- interactive section: set parameters for the run --->
   open ( sumdat,'sumdat.dat');                      { the summary stats file }
   rewrite ( sumdat );                               { prepare for write      }

   date ( todays_date );                {t}
   time ( current_time );               {t}
   writeln (' SET SYSTEM PARAMETERS : ');
   writeln;
   writeln (' ENTER : ');
   writeln (' dlambda : Mean propagation delay time ');
   writeln (' elambda : Probability of correct transmission ');
   writeln (' tiproc  : I-frame mean service time ');
   writeln (' tsproc  : S-frame mean service time ');
   writeln (' tout    : Timout time ');
   writeln (' framenum: The running time per trial ');
   writeln (' sizwin  : The window size ');
   writeln (' seed    : A random seed ');
   readln ( dlambda,elambda,tiproc,tsproc,tout,framenum,sizwin,seed );

   writeln (' tiproc constant (0), csma (1), or token passing (2) ? >');
   readln ( opt1 );
   writeln (' load minimum,load maximum (real)                      >');
   readln ( ropt1,ropt2);
   writeln (' increment size (real)                                 >');
   readln ( increments );
   writeln (' busy diagram listing.  NO = 0, YES = 1                >');
   readln ( opt3 );
   writeln (' single station (0) or double station (1) ?            >');
   readln ( opt4 );

{ ********************* here begins the mainloop ********************** }

   writeln; writeln;
   writeln('***** RUNNING - PLEASE STANDBY *****');
   writeln;
   freelist := nil;
   opt2 := round((ropt2-ropt1)/increments);
```

```
for mainloop := 0 to opt2 do
begin
   load := mainloop*increments + ropt1;
   alambda := (tiproc + 8.0) / load;
   runtim := framenum*alambda;

   { --- busyout printing section --- }

   if ( opt3=1 ) then                    { busyout file is required }
   begin
      open ( busy,'busy.dat' );          { open the busyout file    }
      rewrite ( busy );                  { prepare for writing      }
      writeln (busy,' ':7,todays_date);
      writeln (busy,' ':7,current_time);
      writeln (busy);
      writeln (busy,'dlambda:':20,dlambda:6:1,'elambda:':20,elambda:6:2);
      writeln (busy,'tiproc:':20,tiproc:6:1,'tsproc:':20,tsproc:6:1);
      writeln (busy,'tout:':20,tout:6:1,'framenum:':20,framenum:6:1);
      writeln (busy,'buffsiz:':20,buffsiz:6,'window:':20,sizwin:6)
   end;

   { --- statistics printing section --- }

   open ( stats,'stats.dat' );           { open the stats file          }
   rewrite ( stats );                    { prepare for writing         }
   date ( todays_date );
   time ( current_time );
   writeln (stats,todays_date);
   writeln (stats,current_time);
   writeln (stats,'dlambda:':20,dlambda:7:1,'elambda:':20,elambda:7:1);
   writeln (stats,'tiproc:':20,tiproc:7:1,'tsproc:':20,tsproc:7:1);
   writeln (stats,'tout:':20,tout:7:1,'framenum:':20,framenum:7:1);
   writeln (stats,'buffsiz:':20,buffsiz:7,'window:':20,sizwin:7);
   writeln (stats); writeln(stats); writeln(stats);

   { --- initialize system structures --- }

   for term := 1 to t do
   begin                                 { initialize each terminal:   }
      arrivq[term,n] := 0;               { arrival queue last entry    }
      fillarr (term);                    { call fill arrivals          }
      fillpro (term);                    { call fill delays            }
      fillerr (term);                    { call fill errors            }
      recno [term] := 0;                 { set receive no to zero      }
      nextarr [term] := 0;               { next arrival                }
      nextpro [term] := 0;               { next proc time              }
      nexterr [term] := 0;               { next error                  }
      sendbuff [term] := nil;            { nil the left edge pointer   }
      nextsend [term] := nil;            { nil the next to send pointer }
      bottom [term] := nil;              { nil the bottom pointer      }
      sendseq [term] := 0;               { send sequence number        }
      transbuf[term] := nil;             { nil the trans buffer        }
      intranstop := nil;                 { nil the in-trans queue      }
      recbuff[term] := nil;              { nil the receive buffer      }
      timout[term] := arrivq[term,1] + tout;   { set the first timout  }
      for j := 1 to 8 do isum[term,j] := 0.0;  { set running sums      }
      for j := 1 to 4 do ssum[term,j] := 0.0;
      count[term] := 0.0;
      scount[term] := 0.0
   end;
```

```
clocktim := 0;                          { start at zero clocktim          }
lastct := 0;                            {t}
hlcount := 50;                          {t}

{ --- action loop begins here --- }

findnextevent (term,event);                          { find first event   }

if ( opt3 = 1 ) then histogram (term,event);

while ( clocktim < runtim ) do begin                 { begin event loop   }
   case event of
      arr : arrival (term);                          { process arrival    }
      tra : trans (term);                            { process transmit   }
      rec : receive (term);                          { process receive    }
      tim : calltime (term)                          { process timout     }
   end;
   lastct := round(clocktim);                              {t}
   findnextevent(term,event);                        { find next event    }
   if ( opt3 = 1 ) then histogram(term,event)
end; { of cycle loop }

if (opt3=1) then close (busy);

{ --- statistics calculating section --- }

writeln (stats,'RUNNING STATISTICS':40);
writeln (stats);
writeln (stats,'THEORETICAL LOAD':20,load:5:2);
writeln (stats,'EXPERIMENTAL LOAD 1':20,(isum[1,2]/isum[1,8]):5:2);
writeln (stats,'EXPERIMENTAL LOAD 2':20,(isum[2,2]/isum[2,8]):5:2);
writeln (stats);
for i := 1 to 8 do                          { divide by counts to get   }
begin                                       { the mean statistics       }
    if (count[1]<>0) then isum[1,i] := isum[1,i]/count[1];
    if (count[2]<>0) then isum[2,i] := isum[2,i]/count[2]
end;
for i := 1 to 4 do
begin
    if (scount[1]<>0) then ssum[1,i] := ssum[1,i]/scount[1];
    if (scount[2]<>0) then ssum[2,i] := ssum[2,i]/scount[2]
end;
writeln (stats,' ':36,'TERMINAL 1',' ':5,'TERMINAL 2');
writeln (stats);
writeln (stats,'I-FRAMES':30);
writeln (stats,'Number of frames                    :',
         count[1]:7:0,' ':9,count[2]:7:0);
writeln (stats,'Mean send buffer wait time          :',
         isum[1,1]:7:2,' ':9,isum[2,1]:7:2);
writeln (stats,'Mean transmission (server) time     :',
         isum[1,2]:7:2,' ':9,isum[2,2]:7:2);
writeln (stats,'Mean propagation delay              :',
         isum[1,3]:7:2,' ':9,isum[2,3]:7:2);
writeln (stats,'Mean receive buffer wait time       :',
         isum[1,4]:7:2,' ':9,isum[2,4]:7:2);
writeln (stats,'Mean one-way transit time           :',
         isum[1,5]:7:2,' ':9,isum[2,5]:7:2);
writeln (stats,'Mean acknowledge time               :',
         isum[1,6]:7:2,' ':9,isum[2,6]:7:2);
writeln (stats,'Mean two-way transit time           :',
```

```
            isum[1,7]:7:2,' ':9,isum[2,7]:7:2);
    writeln (stats,'Mean interarrival time                :',
            isum[1,8]:7:2,' ':9,isum[2,8]:7:2);
    writeln (stats);
    writeln (stats,'S-FRAMES':30);
    writeln (stats,'Number of frames                      :',
            scount[1]:7:0,' ':9,scount[2]:7:0);
    writeln (stats,'Mean Send buffer wait time            :',
            ssum[1,1]:7:2,' ':9,ssum[2,1]:7:2);
    writeln (stats,'Mean transmit (server) time           :',
            ssum[1,2]:7:2,' ':9,ssum[2,2]:7:2);
    writeln (stats,'Mean one-way propagation delay        :',
            ssum[1,3]:7:2,' ':9,ssum[2,3]:7:2);
    writeln (stats,'Mean one-way transit time             :',
            ssum[1,4]:7:2,' ':9,ssum[2,4]:7:2);
    writeln (stats); writeln (stats);
    time (current_time);
    writeln (stats,current_time);
    close (stats);                          { close the statistics file }

    { --- Write a summary record to sumdat --- }

    for i := 1 to 2 do
       for j := 1 to 8 do
          buff[i,j] := isum[i,j];
    buff[1,9]:=load;
    buff[2,9]:=load;
    sumdat^ := buff;
    put(sumdat);

    { release of memory for next run }


    for term := 1 to t do
    begin
        emptybuffer(sendbuff[term]);
        release(transbuf[term]);
        emptybuffer(recbuff[term])
    end;

    emptybuffer(intranstop);
    writeln( chr(7) );
    writeln('finished run for load =',load:5:2)

  end; { the mainloop loop }

  close ( sumdat );
  writeln ('  ALL DONE! ');
  for i := 1 to 5 do write( chr(7) )      { IT'S FINISHED!  RING THE BELL!!! }
end.  { main }
```

LOCAL DISTRIBUTION IN COMPUTER COMMUNICATIONS

J. F. Hayes

Reprinted from IEEE Communications Magazine
0163-6804/81/0300-0006 $00.75

LOCAL DISTRIBUTION IN COMPUTER COMMUNICATIONS

JEREMIAH F. HAYES

A sophistication of early queueing theory solves bursty transmission problem.

A significant part of the field of Computer Communications is concerned with providing transmission facilities for data sources which may be characterized as bursty, i.e., short spurts interspersed with relatively long idle periods. It has been estimated, for example, that terminals in interactive data networks are active from 1 to 5 percent of the time [1],[2]. This burstiness allows one to share channels among a number of sources. In this paper, we shall consider a particular context in which transmission facilities must be provided—local distribution. In local distribution a number of geographically dispersed sources are to be connected to a central facility. The importance of local distribution systems lies in the fact that they are the most common class of computer communication networks. Further, as part of large systems, local distribution consumes a significant portion of the total cost. In this paper, we shall describe the basic approaches to local distribution. Our discussion encompasses certain adaptive techniques which have been discovered recently.

The focus of our discussion is on the fundamental principles of the techniques used in local distribution without dwelling on details of implementation. We distinguish three main categories: polling, random access, and adaptive techniques. A fourth category is techniques which are suited to a particular topology—the ring or loop structure. As well as describing the techniques we shall summarize the results of studies of performance. These results quantify the effect of various system parameters on performance. An extensive review of the literature is given in a final section of the paper.

To a large extent the analyses of performance that we shall discuss are based on queueing theory. Queueing theory began with the work of a Danish mathematician, A. K. Erlang (1878-1929) [3], on telephone switching systems. His first paper on the subject was published in 1909 [4]. Amazingly, the formulas derived by Erlang in a 1917 paper [5] are in constant use in engineering the modern telephone office. Furthermore, although queueing theory finds wide use as one of the basic components of operations research, telecommunications remains as its most successful application. As one might expect, voice traffic was the primary concern in telecommunications applications. However, recently there has been an upsurge of interest in data traffic and a corresponding reapplication of queueing theory in connection with computer communications [6],[7].

Although Erlang's work was concerned with voice traffic, certain of his basic concepts are appropriate to data networks. In the generic queueing model customers randomly arrive at a facility with service requirements that may be random in nature. The theory attempts to find probabilistic descriptions of such quantities as the size of the waiting lines, the delay of a customer and availability of a serving facility. In the voice telephone network, demands for service take the form of telephones going off hook or call attempts. Erlang found that given a sufficiently large population, the random rate of such calls can be described by a Poisson process.[1] The service time of a customer is the duration (holding time) of a call and was found to have an exponential distribution which is closely related to the Poisson distribution. In computer communications applications, the generation of data messages at a terminal is the analog of customer arrival. The service time is the time required to transmit the data message. In many cases of interest, the arrival process is approximated by a Poisson process. The duration of messages is commonly taken to be constant or to be exponentially distributed.

POLLING SYSTEMS

A ubiquitous example of a local distribution system is shown in Fig. 1 where we depict part of the communications facilities in the Bonanza of Bargains Shopping Mall and Family Entertainment Center. A number of terminals situated throughout the B of B are bridged across a common line and connected to a common computer. The common line may be wire or coaxial cable.[2] The terminals are engaged in such commercially useful activities as credit checking and inventory control. However, even in the best of seasons the traffic produced by an individual terminal is bursty and a number of terminals can share the same line. Located at the

[1] A definition of the Poisson process will be given in the sequel.

[2] The required properties of this common line are related to the particular local distribution technique employed and will be discussed in due course.

0163-6804/81/0300-0006 $00.75 © 1981 IEEE

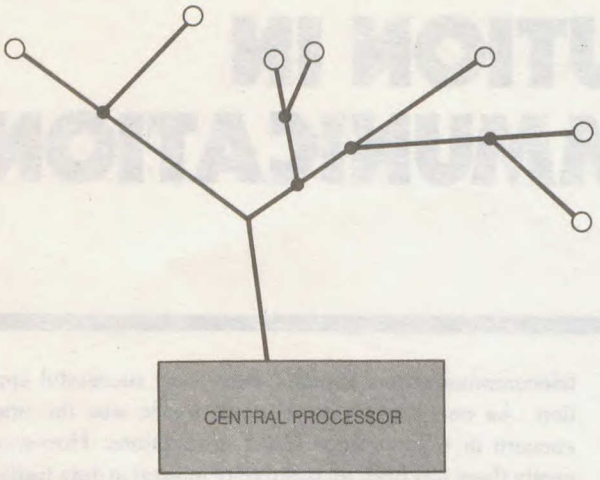


Fig. 1. Geographically dispersed users tree topology.

computer is a controller, one of whose functions is effecting this sharing efficiently and equitably.

A common technique for parcelling out bits per second among users sharing a common line is *roll-call polling*[3]. It is assumed that the common line is such that the controller can broadcast to all terminals simultaneously. Each terminal has an address which is transmitted in sequence by the controller over the common line. After broadcasting a terminal's address the controller pauses for a message from the terminal. If a terminal has a message the polling cycle is interrupted while the message is transmitted.

The ability to achieve economies by sharing transmission facilities is limited by performance requirements usually expressed in the delay experienced by a user in obtaining service. If there are too many terminals on the line, for example, the time required to cycle through all terminals is too large and user dissatisfaction ensues. The parameters of the mathematical models of performance are: the number of terminals, the volume of traffic generated by each terminal, the line speed in bits/s and the line required by the polling protocol. As we shall see in connection with the analysis of polling models a significant factor in performance is overhead, i.e., the time required to poll all terminals even when there are no messages. In the case of terminals equipped with voiceband modems, for example, this may involve equalizer training as well as the phase and timing recovery associated with the transmission of each polling message.

A close relative to roll-call polling is *hub polling* [8],[9]. Again, we have the geographically dispersed terminals of Fig. 1. The controller begins a polling cycle by broadcasting the address of the most distant terminal thereby granting to this terminal exclusive access to the line. After this terminal has transmitted any messages that it might have, it transmits an "end of message" symbol which acts to grant access to the next most distant terminal. Upon receiving this symbol, the next most distant terminal repeats the process, passing on access to the third most distant terminal when its messages have been transmitted. The process continues until all terminals have been given an opportunity to transmit messages whereupon the controller initiates a new cycle. This model contains the same parameters as roll-call polling. The salient difference between the two is the time required to grant access to a terminal. In roll-call polling, the time required to transmit a message and receive a reply is typically much larger than the time required to transmit a symbol from one terminal to another. However, hub polling requires that the line be such that terminals reliably receive transmissions from other terminals.

The hub-polling technique has been implemented in the ring topology depicted in Fig. 2 [10]. Flow around the ring is clockwise as shown. The central processor grants first access to the first terminal downstream. As in the previous implementation, access is passed from a terminal to its nearest neighbor downstream. An end of message character is appended to the data from a terminal. We have the same set of parameters as in the previous cases. The time required to pass access from one terminal to another is the time required to transmit an end of message character.

Before going on to consider other local distribution techniques we pause to consider the performance of polling systems as related to network parameters and to traffic. A useful measure of performance is the cycle time which is the time required to grant access to all terminals in the system at least once and to transmit messages from the terminals. We may view the cycle time as having two components—fixed and random. The overhead or fixed component is the time required to grant access to all terminals. In roll-call polling, for example, it is the time required to broadcast all of the terminal addresses and to listen for replies. In the hub-polling technique, overhead is the total time in a cycle that is required to pass access from one terminal to another.

The random component of a polling cycle is due to the random nature of the message generation process. The number of messages transmitted in a cycle varies from one cycle to the next. The analysis of polling systems is

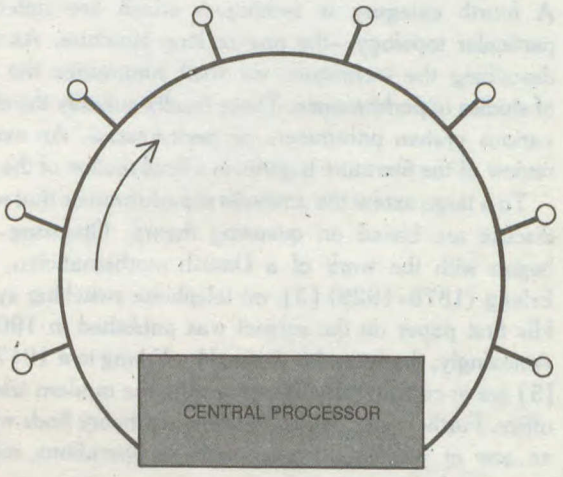


Fig. 2. Ring topology.

[3]For implementation of polling systems, see [8] and [9].

complicated by the fact that there are correlations between the number of messages encountered in successive cycles and in the number of messages in adjacent terminals.

The most studied model assumes Poisson arrival[4] at a terminal having storage facilities which may be regarded as being infinite, i.e., compared to the arrival rate of messages the terminal buffer is so large that the probability of overflow is negligible. The time required to poll n terminals may be written

$$T_c = \sum_{i=1}^{n} t_i + W \qquad (1)$$

where W is the overhead in a cycle, assumed to be constant, and t_i is the time spent at terminal i to read out messages. Even though there are dependencies between buffer contents, the average cycle time is easily found since the expected value of a sum such as shown in (1) is the sum of the expected values. Assuming that the arrival rates and message transmission times are the same for all terminals we find that $\overline{T}_c$ the average duration of a cycle is given by

$$\overline{T}_c = W/(1 - S) \qquad (2)$$

where $S = n\,\overline{m}\,\lambda$, $\overline{m}$ is the average duration of a message and λ is the average arrival rate at each terminal. Equation (2) has a characteristic queueing theory form. The numerator represents overhead, the amount of time during a cycle for which a message is not being transmitted. All of the traffic dependency is contained in the quantity S in the denominator. This load S is the average work presented to the system normalized to the capacity of the channel. In voice networks, a similar quantity has been given the unit of Erlangs. There is a point of instability when $S = 1$ since the average amount of work that is arriving is just equal to the capacity of the system and there is no allowance for overhead. We note that when $W = 0$, the average cycle time is zero. This is consistent if we consider that an infinite number of cycles occur in zero time when the terminals have no messages to transmit. Equation (2) indicates the effect of overhead on performance. Suppose, for example, that the total traffic load into the system is kept constant (i.e., S constant) while the number of terminals is doubled. If overhead is incurred on a per terminal basis, the cycle time is doubled with no increase in traffic.

A more tangible measure of performance for the user is message delay which we define to be the time elapsing between the generation of a message and its transmission over the common line. This delay consists of several components. A message generated at a terminal must wait until it is the terminal's turn to be polled. If the terminal can store more than one message at a time, a queue is formed at each terminal which implies further delay. Finally, a certain amount of time is required simply to transmit the message. There have been a number of analyses of the performance of polling systems.[5] In connection with cycle time, we considered the case of infinite buffers and Poisson message generation. Results on the average delay for this case with constant length messages are shown on Fig. 3. The average delay normalized to the time required to transmit a message is shown as a function of the total load into the system, $S = \overline{m}\lambda n$. The parameters are n, the total number of terminals, and $W/n\overline{m}$ is the overhead per terminal normalized to the message transmission time. The curves show the characteristic rapid increase in delay as the load approaches one. A strong dependence on overhead is also evident. For loadings less than 0.5, which is the region where the system will be operated, overhead dominates. These points can be illustrated by an example. Suppose that ten terminals share a common 2400 bit/s line. Each terminal generates messages at an average rate of 28.8 messages per busy hour (0.008 messages/s). The messages are each 1200 bits long. Finally, assume that in order to poll each terminal and listen for a response 50 ms are required. The load in the system is $S = 0.04$. From Fig. 3, the average delay is approximately 0.25 seconds. Now suppose that as a convenience to users the number of terminals sharing the line is doubled without increasing the load. We see from Fig. 3 that the average message delay doubles. Of the two polling techniques hub polling tends to have lower overhead than roll-call polling. We may view hub polling as a more distributed form of control of access to the system.

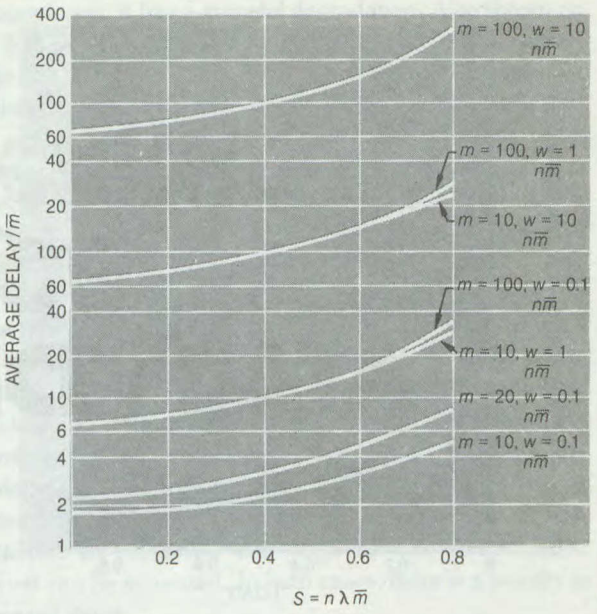


Fig. 3. Polling delay versus load.

LOOP NETWORKS

The ring or loop topology shown in Fig. 2 finds extensive application in distributed processing where computers and

[4]For the Poisson distribution the probability of k message arrivals in T seconds is $P_k = (\lambda T)^k \exp(-\lambda T)/k!$ $k = 0,1,2\cdots$ where λ is the average arrival rate. The interarrival time is exponentially distributed, i.e., P_r [interarrival time $\leq \tau$] $= 1 - \exp(-\lambda\tau)$; $\tau \geq 0$.

[5]See the review of the literature at the end of the paper.

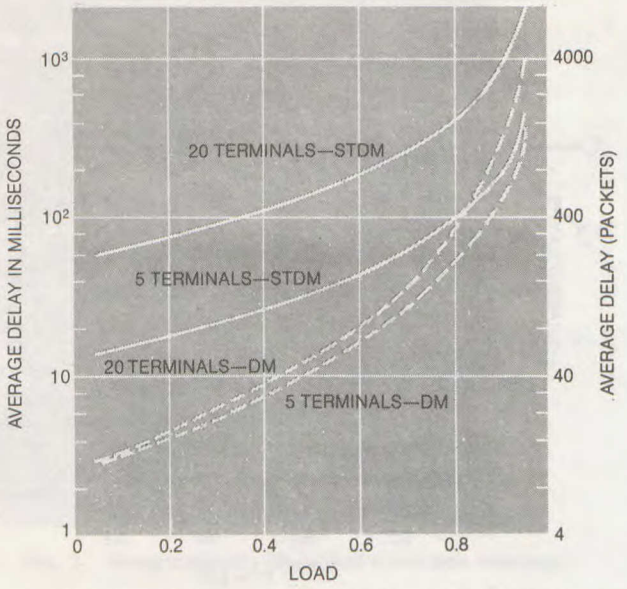


Fig. 4. Average delay versus load in STDM and DM [34].

peripherals in close proximity (within a kilometer) are tied together. In this application, it is necessarily true that the traffic is bursty. However, the loop structure lends itself to interesting multiplexing techniques which may be appropriate to bursty sources. The most obvious technique is a form of time division multiplexing which in this context is commonly called Time Division Multiple Access (TDMA). Assuming synchronous transmission, the flow on the line is partitioned into segments each of which is dedicated to a particular terminal. A terminal simply inserts messages into segments assigned to it. The shortcoming of this system in the case of many lightly loaded terminals is that very often terminals have nothing to send and segments are wasted. At the same time, empty segments may be passing by terminals which do have messages. The same drawback applies to Frequency Division Multiple Access (FDMA) in which each terminal is allocated a fixed bandwidth. A recent study has shown that FDMA is inferior to TDMA from the point-of-view of performance [11].

An alternate technique to TDMA in a loop context is Demand Assignment (DA). The flow is the same as in TDMA except that the blocks are not assigned to any terminal. When an empty block passes by a terminal which has a message to transmit, the block is seized by the terminal and the message along with addressing information is inserted. There is an increase in the utilization of the line over TDMA at the cost of an increase in the complexity of the terminals. On Fig. 4, the average delay is shown as a function of the load with the number of terminals in the system as a parameter. The results illustrate the inefficiency of TDMA for lightly loaded systems where the dominant factor is the time required to transmit a single message. At light loading, demand multiplexing is superior to TDMA by a factor equal to the number of terminals sharing the line. As the load increases, the difference between the two systems decreases since in demand multiplexing different terminals will tend to have messages at the same time. Once again, the lesson that we carry away from this study of loop systems is that in lightly loaded systems, a distributed control of access to the channel is more sufficient.

The TDMA technique is also appropriate to the tree topology of Fig. 1. It is necessary to establish synchronization among the terminals. Each terminal is assigned a periodically recurring time slot. However, in the tree topology as well as in the loop topology the TDMA technique is not efficient for bursty sources.

RANDOM ACCESS (ALOHA)

Random access techniques hitherto associated with radio and satellite systems have recently been applied to local area networks [12]. The origin of these methods is the ALOHA protocol which is the ultimate in distributed control. Again, we assume that n terminals are sharing the same channel as depicted in Fig. 1. As soon as a terminal generates a new message it is transmitted on the common line. Along with the message, the terminal transmits address bits and parity check bits. If a message is correctly received by the central controller a positive acknowledgment is returned to the terminal on the return channel. Since there is no coordination among the terminals it may happen that messages from different terminals interfere with one another. If two or more messages collide, the resulting errors will be detected by the controller which returns a negative acknowledgment or no acknowledgment. An alternative implementation is to have the terminal itself detect collisions simply by listening to the channel. After a suitable timeout interval a terminal involved in a collision retransmits the message. In order to avoid repeated collisions the retransmission intervals are chosen randomly. The key element of the ALOHA protocol and its descendants is the retransmission traffic on the common line. If the rate of newly generated traffic is increased, the rate of conflicts among terminals increases to the point where retransmitted messages dominate and there is saturation. This effect is expressed succinctly in the formula

$$S = G \exp(-G) \qquad (3)$$

where S is the normalized load into the system generated at all terminals and G is the total traffic on the line including all retransmissions. In the derivation of (3), it is assumed that all messages are the same length and that they are generated at a Poisson rate. The plot of (3) on Fig. 5 shows that the channel saturates at 18 percent of its capacity inasmuch as the input cannot be increased beyond this point. Thus, it appears that simplicity of control is achieved at the expense of channel capacity.

The basic ALOHA technique can be improved by rudimentary coordination among the terminals. Suppose that a sequence of synchronization pulses is broadcast to all terminals. Again, let us assume constant length messages or packets. A so-called slot or space between synch pulses is equal to the time required to transmit a message. Messages, either newly generated or retransmitted, can only be transmitted at a pulse time. This simple device reduces the rate of collisions by half since only messages generated in the

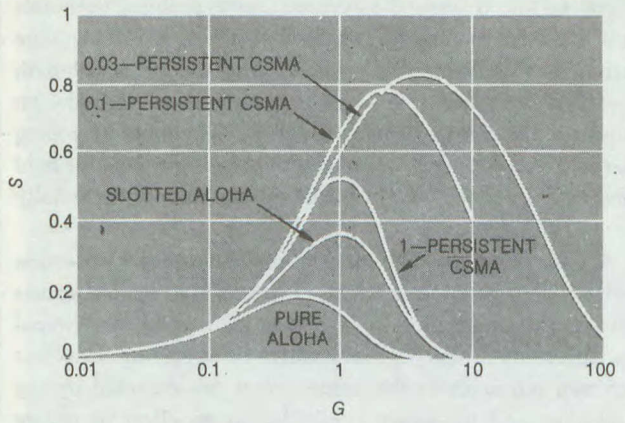


Fig. 5. Input load as a function of channel traffic for several random access techniques [47].

same interval interfere with one another. In pure ALOHA, the "collision window" is two message intervals. The equation governing the behavior of slotted ALOHA is

$$S = G \exp(-2G). \quad (4)$$

We see from the plot of (4) on Fig. 5 that the channel saturates at approximately 36 percent of capacity.

An extension of the ALOHA technique that is particularly appropriate for local distribution is Carrier Sense Multiple Access (CSMA). Before transmitting a message a terminal listens on the common channel for the carrier of another terminal which is in the process of transmitting. If the channel is free the terminal transmits; if not, transmission is deferred. Variations on the basic technique involve the retransmission strategy. We illustrate retransmission strategies by means of the *P*-persistent CSMA strategy. If the channel is busy then the terminal transmits at the end of the current transmission with probability *P*. With probability 1-*P*, transmission is delayed by τ seconds which is the maximum propagation time between any pair of terminals. Due to propagation delay there may be more than one terminal transmitting at the same time in which case messages are retransmitted after random timeout intervals. The value of *P* is chosen so as to balance the probability of retransmission with channel utilization. The characteristic equations for CSMA are plotted on Fig. 5. The form is similar to pure and slotted ALOHA. The ability to sense carrier from other terminals leads to considerable improvement in throughput. As indicated, decreasing *P* leads to improved throughput which is obtained at the expense of increased delay. The curves shown in Fig. 5 are for a propagation delay 0.01 normalized to message transmission time. As this normalized delay is increased the performance of CSMA degrades.

There have been a number of analyses of random access protocols focusing on message delay as a function of throughput. On Fig. 6, we summarize the results of this work in the form of normalized message delay as a function of load. For lightly loaded systems, pure and slotted ALOHA perform well. However, as the load increases the increasing rate of retransmission rapidly degrades performance. Since the carrier sense protocol keeps the channel clear by avoiding retransmission, it has a graceful degradation. Also shown on Fig. 6 is delay for roll-call polling. As we have seen earlier, there is a severe penalty for overhead required when the number of terminals is increased. The curves also show that the performance of the polling protocol degrades more gracefully than that of the random access protocols. This is where the beneficial effect of the controller is seen. By scheduling transmission, the avalanche effect of retransmissions in the random access protocols is prevented.

The curves for the random access techniques in Fig. 5 show the same basic form in which a level of input traffic *S* can lead to two possible levels of line traffic *G*. It can be shown that this characteristic may lead to an unstable state resulting in saturation of the channel and a drop in throughput. By choosing system retransmission parameters properly, unstable states can be prevented. An example of this is decreasing the parameter *P* in the *P*-persistent CSMA protocols. In ALOHA, the range of the retransmission interval can be increased. In both cases, there is a penalty in increased delay.

ADAPTIVE TECHNIQUES

The deleterious effect of overhead on the performance of polling systems is abundantly clear from the foregoing results.

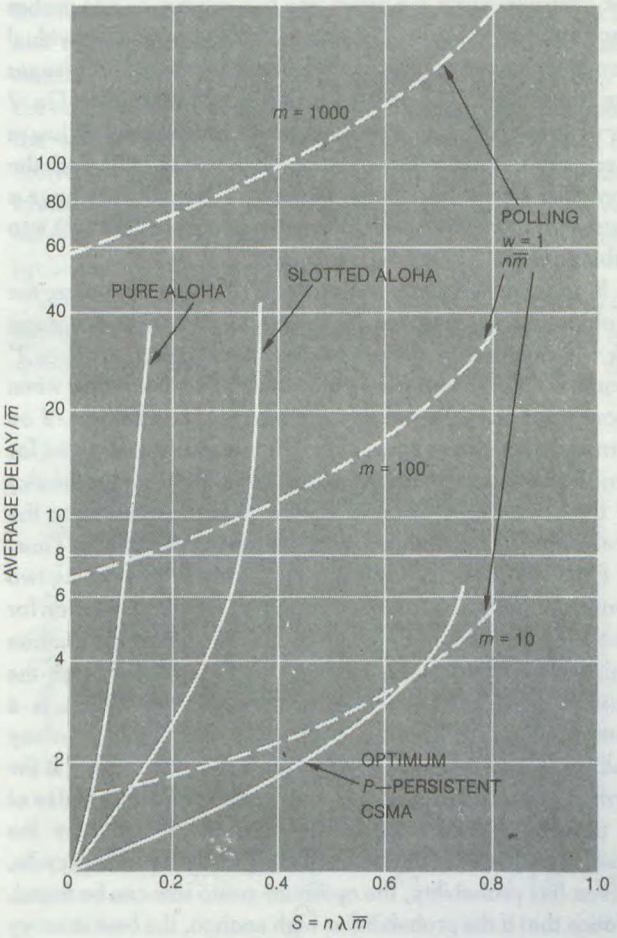


Fig. 6. Delay in random access systems [47].

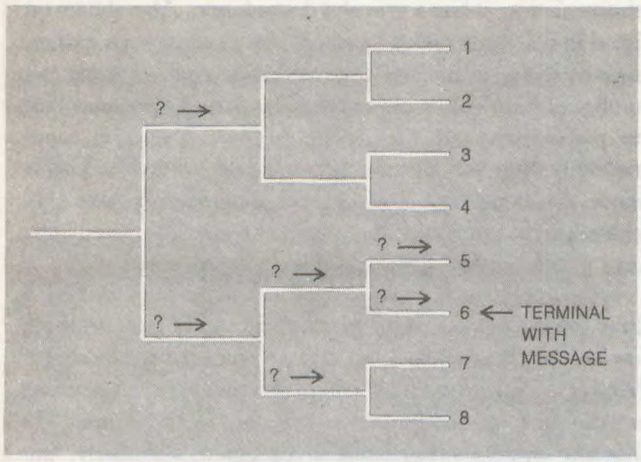


Fig. 7. Probing illustration.

In order to ameliorate this effect an adaptive technique has been devised recently. The essence of the technique, which has been designated probing, is to poll terminals in groups rather than one at a time. In order to implement the technique, it is assumed that the central controller can broadcast to all terminals in a group simultaneously. If a member of a group of terminals being probed has a message to transmit, it responds in the affirmative by putting a noise signal on the common line. Upon receiving a positive response to a probe, the controller splits the group into two subgroups and probes each subgroup in turn. The process continues until individual terminals having messages are isolated whereupon messages are transmitted. The probing protocol is illustrated on Fig. 7 for a group of eight terminals of which terminal 6 has a message. The algorithm is essentially a tree search which the controller begins by asking, in effect, "Does anyone have a message?" Branches with affirmative responses are split into subbranches.

If only one terminal in a group of 2^k has a message, the probing process requires the controller to transmit at most $2k + 1$ inquiries. In contrast, conventional polling requires 2^k inquiries. The comparison may not be so favorable when more than one terminal has a message. For example, if all terminals have messages, $2^{k+1} - 1$ inquiries are required for probing. This consideration leads to adaptivity where the size of the initial group to be probed is chosen according to the probability of an individual terminal having a message. Thus, in Fig. 7, for example, one may begin a cycle by probing two groups of four rather than one group of eight. The criterion for choosing the sizes of the groups is the amount of information gained from an inquiry. If the initial group is too large the answer to an inquiry is almost certainly, "Yes there is a message." However, if the group is split into too many subgroups, the answer to an inquiry is too often, "No." If the arrival of messages to terminals is Poisson, the probability of a terminal having a message can be calculated by the controller given the duration of the previous probing cycle. Given this probability, the optimum group size can be found. Notice that if the probability is high enough, the best strategy may be to poll every terminal.

The results of simulation for the adaptive technique are shown on Fig. 8 where the average time to probe all terminals in a 32 terminal network is shown as a function of message arrival rate. In Fig. 8, the cycle time and the message length are normalized to the amount of time required to make an inquiry. The comparison made with conventional polling shows a considerable improvement in performance for light loading. Moreover, due to the adaptivity there is no penalty for heavy loading.

Although the probing concept was devised in connection with polling systems it is also appropriate in a random access context. Suppose that in response to a probe a terminal transmits any messages that it might be harboring. Conflicts between terminals in the same group are detected by the controller and the group is divided in an effort to isolate individual terminals. Each subgroup is given access to the line in turn. Optimal initial group sizes can be chosen by means of very much the same criterion as in polling systems. Probing too large a group results in almost certain conflict. The opposite extreme gives too many probes of empty groups of terminals. Again, the optimum group size can be chosen adaptively as the process unfolds. The probability of a terminal having a message is a function of the previous cycle and the average message generation rate at a terminal. This probability determines optimum initial group size.

Control of the adaptive process need not be as centralized as in the foregoing. Suppose that as in slotted ALOHA synchronizing pulses are broadcast to all terminals. Suppose further that the slots between synch pulses are subdivided into two equal subslots. In the tree search protocol, the first subslot is devoted to an upper branch and the second to a lower. Consider the example in Fig. 9(a) and (b) depicting an eight terminal system of which 5, 7, and 8 have messages. The first subslot is empty since it is dedicated to terminals 1-4. In the second subslot, terminals 5, 7, and

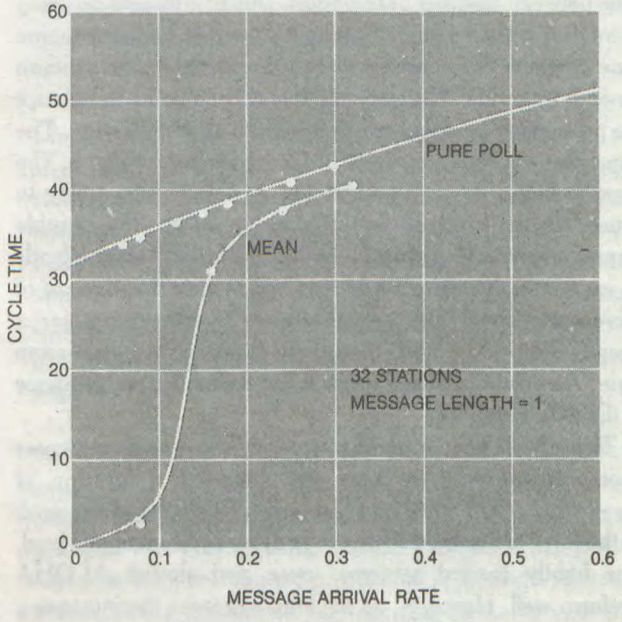
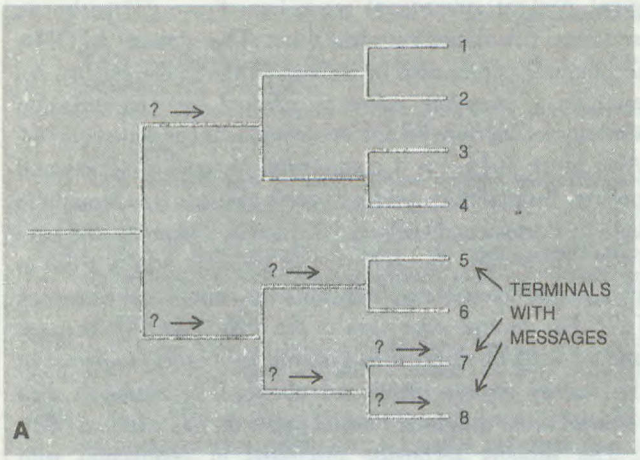


Fig. 8. Average cycle time versus message arrival probing technique [49].



B

NO MESSAGE TERMINALS 1-4 EMPTY	TERMINALS 5, 7, AND 8 CONFLICT	TERMINAL 5 TRANSMITS	TERMINAL 7 AND 8 CONFLICT	TERMINAL 7 TRANSMITS	TERMINAL 8 TRANSMITS
	SLOT 1		SLOT 2		SLOT 3

Fig. 9. (a) Tree search illustration. (b) Tree search illustration.

8 conflict. The conflict is resolved in subsequent slots. After this conflict resolution process has begun, any newly arrived messages are held over until the next cycle. Again, the algorithm can be made adaptive by adjusting the size of the initial groups to be given access to the channel according to the probability of a terminal having a message.

Upper and lower bounds on average delay as a function of the input load are shown in Fig. 10. Notice that the system saturates when the load is 43 percent of capacity. This contrasts with the case of slotted ALOHA where this maximum is 36 percent of capacity. Recent improvements of the technique have pushed this figure to over 50 percent. There are no unstable states where the system is saturated by retransmissions and conflicts. If conflicts persist each terminal is assigned an individual slot and the system reverts to TDMA.

The so-called "random urn" is another technique in which the size of groups granted access is chosen adaptively. The assumption underlying this protocol is that at the beginning of a cycle the total number of terminals having messages is known to all terminals. Access is granted to groups of size k where k is chosen so as to maximize the probability that only one terminal has a message. If, as in heavily loaded systems, all terminals have messages, then the optimum group size is one and the system is simple TDMA. Under light loading, the random urn scheme behaves as ALOHA. The key issue in implementing this scheme is determining the number of terminals having messages. One possibility is a reservation interval at the beginning of a cycle. In this interval, terminals having messages indicate as such. From this, terminals can estimate the number of other terminals having messages. Simulation studies indicate that performance is insensitive to small errors in this estimate.

Related to random access multiplexing are a large number of reservation techniques in which sources, upon becoming active, reserve part of the channel. The reservation techniques are appropriate to sources which are active infrequently but transmit a steady stream while active. The traffic from such sources is not bursty. However, the request methods are and consequently may be treated by the techniques discussed in the foregoing. For example, reservations could be made using the ALOHA technique over a separate channel.

REVIEW OF LITERATURE

There is an analogy between polling systems and machine patrolling in which a repairman examines n machines in a fixed sequence. If a machine is broken he pauses to make repairs. This is the analog in polling systems to message transmission. The overhead that is incurred is the time required to walk between machines. This walktime corresponds to the time required to poll a terminal. The earliest work on this problem was done for the British cotton industry by Mack *et al.* [13]. Based on this work, Kaye [14] derived the probability distribution of message delay for the case where terminals store a single fixed length message. This result is the one shining example of a simple expression for probability distributions in polling models. Some idea of the delicacy of the model may be gained from Mack's analysis of very much the same situation but with a variable repair time [15] (corresponding to variable length messages in polling systems). In order to find a solution it is necessary to solve a set of 2^{n-1} linear equations. For a treatment of work on related problems, see Cox and Smith [16].

A great deal of work has been devoted to the case of the infinite buffer. The earliest work in this area involved just two queues with zero overhead [17],[18]. Later, this was

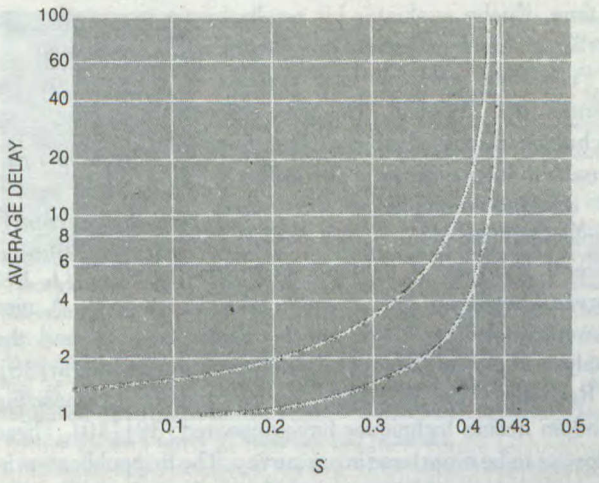


Fig. 10. Average delay versus load tree search [50].

generalized to two queues with nonzero overhead [19],[20]. In terms of the models that we are concerned with, the first papers of interest are those of Cooper and Murray [21] and Cooper [22]. The number of buffers is arbitrary and both the gated and exhaustive services models are considered. The drawback is that the analysis assumes zero overhead. The characteristic functions of the waiting times are found. Also found is a set of n $(n + 1)$ linear equations whose solution yields the mean waiting time at each buffer when the message arrival time is different for each. The assumption of zero overhead here may yield useful lower bounds on performance.

For a long time, the only work on an arbitrary number of queues with nonzero overhead was by Liebowitz [23] who suggested the independence assumption. In 1972, both Hashida [24] and Eisenberg [25] separately published results on multiple queues with nonzero overhead. Both used imbedded Markov chain approaches. (Some of Hashida's results are plotted on Fig. 3.) Computer communications stimulated the next significant step in polling models. Konheim and Meister [26] studied a discrete time version of model. Transmission time over the channel is divided into fixed size discrete units called slots. Messages are described in terms of data units which fit into these slots. (An 8-bit byte is a good example of a data unit.) The analysis is carried out by imbedding a Markov chain at points separated by slots. In most of this work, the emphasis was upon symmetric traffic. Recently, Konheim and Meister's work was extended to the case of asymmetric traffic [27]. Interestingly, it was found that in the case of asymmetric traffic, the order in which terminals are polled affects performance.

A significant remaining problem involves nonexhaustive service where, at most, a fixed number of messages are transmitted from a particular buffer. If there are more than the fixed number of messages at the buffer they are held over until the next cycle. If there are less than this fixed number the next terminal is polled immediately after the buffer is emptied. At the present writing no exact analysis is available. There have been several analyses of systems of this kind based upon approximations [28]-[30]. The latest of these is by Kuehn who obtains results when at most one message is removed at a time. Kuehn evaluates his results by comparing them to earlier results by Hashida and Ohara and to simulation.

Pioneering work on loop systems was carried out by Farmer and Newhall [10] who proposed the hub-polling technique discussed above. The demand multiplexing approach in loop systems is due to Pierce [31],[32]. A version of demand multiplexing was used by Fraser in the implementation of the Spider network [33]. There have been several analyses of demand multiplexing [34]-[37]. The curves shown on Fig. 4 were taken from [34]. A nice summary of later work on the implementation and the analysis of performance of loop networks is contained in [38].

Recently, two thorough survey papers emphasizing random access techniques have appeared [39],[40]. These allow us to be more terse in our survey. The first publication in the area is due to Abramson [41],[42] who derived (3) under the simplifying assumption of Poisson retransmitted traffic. A great deal of subsequent work has shown (3) to be an accurate description of ALOHA. The slotted ALOHA technique was proposed by Roberts [43] who derived (4). An analysis of message delay as effected by retransmission strategy for the pure ALOHA technique is contained in [44]. Also given in [44] is a comparison of random access and polling. Instability in random access systems was brought to light by Carleial and Hellman [45] and by Kleinrock and Lam [46]. The results on carrier sense multiple access given in Figs. 5 and 6 are drawn from work by Tobagi and Kleinrock [47]. For several extensions of the basic ALOHA concept and for work on reservation systems, the reader is referred to the survey papers mentioned above. The reader is also referred to an insightful tutorial paper on this material [48].

The probing technique discussed in connection with adaptive systems is due to Hayes [49]. The distributed adaptive protocol described above was devised by Capetanakis [50] who also found the increase in the capacity given by adaptive techniques. More recent work in this area is contained in [51]-[55]. Kleinrock and Yemini devised the random urn scheme [56].

CONCLUSION

We have reviewed the basic techniques of implementing local distribution for bursty data sources. A couple of generalizations emerge from this study. It seems that under conditions of light loading distributed control is best. However, as the loading increases distributed control leads to difficulties and centralized control gives the better performance. This is entirely in conformity with everyday experience with automobile traffic. At 4 A.M., stop signs minimize delay. However, along heavily traveled routes at rush hour, stop signs would cause collisions (in the usual sense of the word) and the centralized control of traffic lights is required.

ACKNOWLEDGMENT

The author expresses his thanks to Pauline Fox for her efforts in preparing the manuscript.

REFERENCES

[1] P. Jackson and C. Stubbs, "A study of multiaccess computer communications," in *AFIPS Conf. Proc.*, vol. 34, p. 491.

[2] E. Fuchs and P. E. Jackson, "Estimates of distributions of random variables for certain computer communications traffic models," *CACM*, vol. 13, no. 12, pp. 752-757, 1970.

[3] E. Brockmeyer, H. L. Halstrøm, and A. Jensen, "The life and works of A. K. Erlang," *Trans. Danish Academy Tech. Sci.*, ATS no. 2, 1948.

[4] A. K. Erlang, "The theory of probabilities and telephone conversations," *Nyt Tidsskrift Matematik*, B.V20, pp. 33-39, 1909.

[5] ——, "Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges," *Electroteknikeren*, vol. 13, pp. 5-13, 1917; in English: *PO Elect. Eng. J.*, vol. 10, pp. 189-197, 1917-1918.

[6] L. Kleinrock, *Queueing Systems, Vol. 1: Theory* and *Vol. 2: Computer Applications*. New York: Wiley, 1975.

[7] H. Kobayashi and A. G. Konheim, "Queueing models for computer communications system analysis," *IEEE Trans. Commun.*, vol. COM-25, pp. 2-29, Jan. 1977.

[8] M. Schwartz, *Computer Communication Network Design and Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1977.

[9] J. Martin, *Teleproccessing Network Organization*. Englewood Cliffs, NJ: Prentice Hall, 1970.

[10] W. D. Farmer and E. E. Newhall, "An experimental distributed switching system to handle bursty computer traffic," in *Proc. ACM Symp. Problems Optimization Data Commun. Syst.*, pp. 1-34, Pine Mountain, GA, Oct. 1969.
[11] I. Rubin, "Message delays in FDMA and TDMA communications channels," *IEEE Trans. Commun.*, vol. COM-27, pp. 769-777, May 1979.
[12] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, pp. 395-404, July 1976.
[13] C. Mack, T. Murphy, and N. L. Webb, "The efficiency of *N* Machines unidirectionally patrolled by one operative when walking and repair times are constant," *J. Royal Stat. Soc. Ser. B.*, vol. 19, pp. 166-172, 1957.
[14] A. R. Kaye, "Analysis of a distributed control loop for data transmission," in *Proc. Symp. Comput. Commun. Network Teletraffic*, Polytech. Inst. Brooklyn, Brooklyn, NY, Apr. 4-6, 1972.
[15] C. Mack, "The efficiency of *N* machines unidirectionally patrolled by one operative when walking time is constant and repair times are variable," *J. Royal Stat. Soc. Ser. B.*, vol. 19, pp. 173-178, 1957.
[16] D. R. Cox and W. L. Smith, *Queues*. London: Methuen, 1958.
[17] B. Avi-Itzhak, W. L. Maxwell, and L. W. Miller, "Queues with alternating priorities," *J. Oper. Res. Soc. America*, vol. 13, no. 2, pp. 306-318, 1965.
[18] L. Takacs, "Two queues attended by a single server," *Opns. Res.*, vol. 16, pp. 639-650, 1968.
[19] J. S. Sykes, "Simplified analysis of an alternating priority queueing model with setup time," *Opns. Res.*, vol. 18, pp. 399-413, 1970.
[20] M. Eisenberg, "Two queues with changeover times," *Opns. Res.*, vol. 19, pp. 386-401, 1971.
[21] R. B. Cooper and G. Murray, "Queueing served in cyclic order," *Bell Syst. Tech. J.*, vol. 48, pp. 675-689, Mar. 1969.
[22] ——, "Queues served in cyclic order: Waiting times," *Bell Syst. Tech. J.*, vol. 49, no. 3, pp. 399-413, Mar. 1970.
[23] M. A. Liebowitz, "An approximate method for treating a class of multiqueue problems," *IBM J.*, vol. 5, pp. 204-209, July 1961.
[24] O. Hashida, "Analysis of multiqueue," *Rev. Elect. Commun. Lab.*, NTT vol. 20, Nos. 3 and 4, pp. 189-199, Mar. and Apr. 1972.
[25] M. Eisenberg, "Queues with periodic service and changeover times," *Opns. Res.*, vol. 20, pp. 440-451, 1972.
[26] A. G. Konheim and B. Meister, "Waiting lines and times in a system with polling," *J. ACM*, vol. 21, pp. 470-490, July 1974.
[27] G. B. Swartz, "Polling in a loop system," *J. ACM*, vol. 27, pp. 42-59, Jan. 1980.
[28] O. Hashida and K. Ohara, "Line accommodation capacity of a communication control unit," *Rev. Elect. Commun. Lab.*, NTT vol. 20, pp. 231-239, 1972.
[29] S. Halfin, "An approximate method for calculating delays for a family of cyclic type queues," *Bell Syst. Tech. J.*, vol. 54, pp. 1733-1754, Dec. 1975.
[30] P. J. Kuehn, "Multiqueue systems with nonexhaustive cyclic service," *Bell Syst. Tech. J.*, vol. 58, pp. 671-699, Mar. 1979.
[31] J. R. Pierce, "How far can data loops go?," *IEEE Trans. Commun.*, vol. COM-20, pp. 527-530, June 1972.
[32] ——, "A network for the block switching of data," *Bell Syst. Tech. J.*, vol. 51, pp. 1133-1145, July/Aug. 1972.
[33] A. G. Fraser, "Spider—A data communications experiment," Computing Sci. Tech. Rep. 23, Bell Laboratories, Murray Hill, NJ, 1974.
[34] J. F. Hayes, "Performance models of an experimental computer communications network," *Bell Syst. Tech. J.*, vol. 53, pp. 225-259, Feb. 1974.
[35] J. F. Hayes and D. N. Sherman, "Traffic analysis of a ring switched data transmission system," *Bell Syst. Tech. J.*, vol. 50, pp. 2947-2978, Nov. 1971.
[36] A. G. Konheim and B. Meister, "Service in a loop system," *J. ACM*, vol. 19, pp. 92-108, Jan. 1972.
[37] J. D. Spragins, "Loop transmission systems—Mean value analysis," *IEEE Trans. Commun.*, vol. COM-20, Part II, pp. 592-602, June 1972.
[38] B. K. Penney and A. A. Baghdadi, "Survey of computer communications loop networks," *Comput. Commun.*: Part 1, vol. 2, no. 4, pp. 165-180; Part 2, vol. 2, no. 5, pp. 224-241.
[39] F. A. Tobagi, "Multiaccess protocols in packet communications systems," *IEEE Trans. Commun.*, vol. COM-28, pp. 468-489, Apr. 1980.
[40] S. S. Lam, "Multiple access protocols," TR-88, Dep. of Comput. Sci., Univ. of Texas at Austin, to appear in *Computer Communication: Start of the Art and Direction for the Future*, W. Chou, Ed. Englewood Cliffs, NJ: Prentice-Hall.
[41] N. Abramson, "The ALOHA system—Another alternative for computer communications," in *1970 Fall Joint Comput. Conf. AFIPS Conf. Proc.*, vol. 37, pp. 281-285.
[42] ——, "The ALOHA system," *Comput. Commun. Networks*, N. Abramson and F. Kuo, Eds. Englewood Cliffs, NJ: Prentice-Hall.
[43] L. G. Roberts, "ALOHA packet system with and without slots and capture," *Computer Commun. Rev.*, vol. 5, pp. 28-42, Apr. 1975.
[44] J. F. Hayes and D. N. Sherman, "A study of data multiplexing techniques and delay performance," *Bell Syst. Tech. J.*, vol. 51 pp. 1985-2011, Nov. 1972.
[45] A. B. Carleial and M. E. Hellman, "Bistable behaviour of ALOHA-type systems," *IEEE Trans. Commun.*, vol. COM-23, pp. 401-410, Apr. 1975.
[46] S. S. Lam and L. Kleinrock, "Packet switching in a multiaccess broadcast channel: Performance evaluation," vol. COM-23, pp. 410-423, Apr. 1975.
[47] F. A. Tobagi and K. Kleinrock, "Packet switching in radio channels," *IEEE Trans. Commun.*: Part I: Carrier Sense Multiple Access Modes and Their Throughput Delay Characteristics, vol. COM-23, pp. 1400-1416, Dec. 1975; Part III: Polling and (Dynamic) Split Channel Reservation Multiple Access, vol. 2, COM-24, pp. 832-845, Aug. 1976.
[48] L. Kleinrock, "On resource sharing in a distributed communications environment," *IEEE Commun. Mag.*, vol. 17, pp. 27-34, Jan. 1979.
[49] J. F. Hayes, "An adaptive technique for local distribution," *IEEE Trans. Commun.*, vol. COM-26, pp. 1178-1186, Aug. 1978.
[50] J. Capetanakis, "Tree algorithms for packet broadcast channels," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 505-515, Sept. 1979.
[51] A. Grami and J. F. Hayes, "Delay performance of adaptive local distribution," in *Proc. ICC '80*, Seattle, WA, pp. 39.4.1-39.4.5, June 1980.
[52] N. Pippenger, "Bounds on the performance of protocols for a multiple access broadcast channel," Report RC 7742, Math Science Dep., IBM Thomas J. Watson Research Center, Yorktown Heights, NY, June 1979.
[53] P. A. Humblet and J. Mosely, "Efficient accessing of a multiaccess channel," presented at the IEEE Conf. Decision Contr., Albuquerque, NM, Dec. 1980.
[54] C. Meubus and M. Kaplan, "Protocols for multiaccess packet satellite communication," in *Proc. NTC '79*, Washington, DC, Dec. 1979, pp. 11.4.1-11.4.5.
[55] E. P. Gundjohnsen *et al.*, "On adaptive polling technique for computer communication networks," in *Proc. ICC '80*, Seattle, WA, June 1980, pp. 13.3.1-13.3.5.
[56] L. Kleinrock and Y. Yemini, "An optimal adaptive scheme for multiple access broadcast communication," presented at the ICC '78, Toronto, Ont., Canada, June 1978.

Jeremiah F. Hayes received the B.E.E. degree from Manhattan College, New York, NY, in 1956. He received the M.S. degree in mathematics from New York University, New York, NY, in 1961, and the Ph.D. degree in electrical engineering from the University of California, Berkeley, CA, in 1966.

From 1956 to 1960 he was a Member of the Technical Staff at Bell Laboratories, Murray Hill, NJ. He worked at the Columbia University Electronics Research Laboratories, New York, NY, from 1960 to 1962. In the interval 1966 to 1969 he was a member of the faculty at Purdue University, Lafayette, IN. During the summer of 1967 he was employed by the Jet Propulsion Laboratory, Pasadena, CA. From 1969 to 1978 he was a member of the Technical Staff at Bell Laboratories, Holmdel, NJ. Since September 1978 he has been a member of the Electrical Engineering Department at McGill University, Montreal, P.Q., Canada.

Professor Hayes is a Senior Member of the IEEE. He is currently the Editor for Computer Communication of the IEEE TRANSACTIONS ON COMMUNICATIONS. His research interest is primarily in the area of computer communications.