DEPARTMENT OF COMMUNICATIONS

CANADA

MINISTÈRE DES COMMUNICATIONS

Volume 3

Time Shared Systems

- Communications Processor Theory

- Communications Processor Hardware

by

John deMercado

Terrestrial Planning Branch
June 1972

DEPARTMENT OF COMMUNICATIONS

CANADA

MINISTÈRE DES COMMUNICATIONS

Volume 3

Time Shared Systems

- Communications Processor Theory
- Communications Processor Hardware

by

John deMercado

Terrestrial Planning Branch
June 1972

Acknowledgements

The purpose of these notes is to promote dialogue within the Terrestrial Planning Branch and serve as a basis for our computer-communication systems implementation program.

The notes are only in first draft form and borrow heavily from the references. They should be read in conjunction with the attached reference papers.

As a revised version is planned the author would appreciate any corrections or omissions in the text that were brought to his attention. He also wishes to thank Messrs. John Harris, S. Mahmoud and Kalman Toth for their valuable contributions.

Miss Gail Widdicombe and Miss Yollande Chartrand typed them in record time from an almost unreadable handwritten manuscript.

Contents

Communications Processor Theory

REFERENCES:

1. W.W. Chu, "A study of asynchronous time division multiplexing for time sharing computer systems", FJCC 1969.

2. W.W. Chu, "Demultiplexing considerations for statistical multiplexers", IEEE Transactions in Communication techology June 1972.

3. W.W. Chu, "Design Considerations of Statistical Multiplexers" ACM Symposium on Problems in the optimization of data communication systems", October 1969.

4. Jack S. Sykes, "Analytical Model of Half-Duplex Interconnections of Computers", IEEE Transactions on Communications Technology Vol. Com.17, No. 2, April 1969.

Contents (cont'd)

Communications Processor Hardware

1. Michael Townsend, Communication Control by Computer - an Introduction. Telecommunications May 1972.

2. D. Doll, Planning Effective Data Communications Systems - Data Processing Magazine, November 1970.

3. H. Becker, Communications Processing for Large Data Networks - Data Processing Magazine, November 1970.

Communications Processor Theory

Communications Processor Theory

Introduction

In these notes, a brief outline of the ways in which auxiliary storage and I/O devices gain access to the resources of the main memory is presented. The study of these ways is called Communications Processor Theory. It is conceptually useful to think in terms of conventional communications theory as these theories stand almost in a 1:1 correspondence.

METHODS OF CONNECTING DEVICES TO MEMORY MODULES

Direct Connection

A basic problem with auxiliary storage and I/O devices is gaining access to a direct-transfer path to main memory. Further, logic is required to determine where in memory to transfer to or from and to determine how much information to transfer. Theoretically, these problems can be solved by providing a direct-transfer path, addressing, and count logic for each device to each memory module. Such a scheme is shown in Figure 1. This approach is usually unsatisfactory because of its cost in terms of line-selection logic and line-driving circuits. This cost can be reduced by the use of information available about device and memory information-transfer rates. From Figure 1, it is apparent that a memory module can transfer information to only one device at a time and devices can transfer to only one memory module at a time. Transfer can take place over a maximum of n or m lines, depending on which is larger, but there are n x m lines. Therefore, at a given instant, most of the access lines will be idle, and their associated logic and driver circuitry not used.

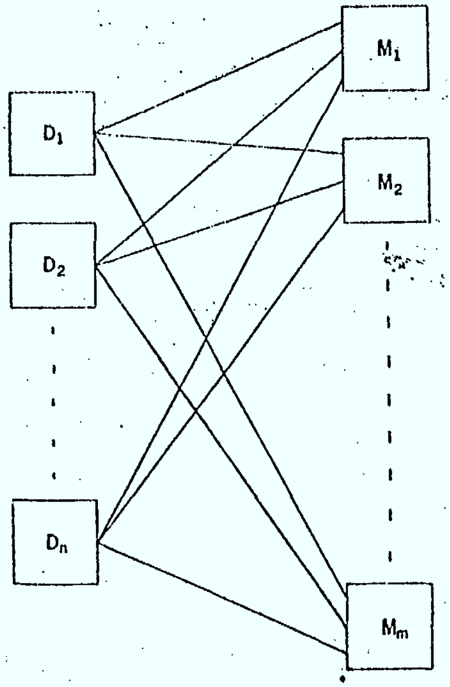


Figure 1: Direct communication of n devices with m memory modules (where n > m, usually).

Cross-Bar Connection

The above fact, suggests that the logic associated with each device should be used to determine which access line to use and the logic associated with each memory module could be used to arbitrate between competing requests for access and centralize it in the interconnection scheme as shown in Figure 2. The logic required, in this scheme, for determining the path between devices and modules is called a crossbar switch. The circled points called switchpoints, determine the connection path from devices to modules. Conflicts for access to a given module line would be resolved by a priority scheme and analyzed via queuing theory. This use of a crossbar switch reduces the number of line drivers required, but the switching logic required to set up a transfer path is probably as great as in Figure 1. The crossbar switch however has the disadvantage over the direct-connection scheme in that localizing the switching logic in one central unit makes the system highly dependent on the reliability of the crossbar switch.

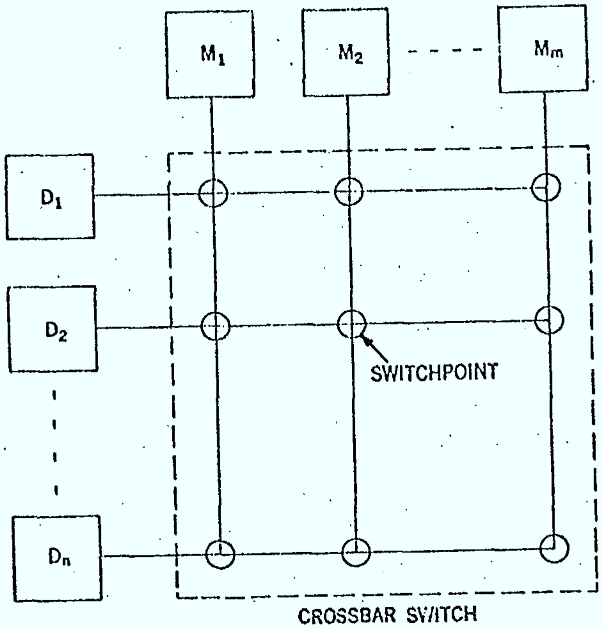


Figure 2: A crossbar switch for n devices to communicate with m memory modules.

Distributed Connection

The logic of the cross bar switch can be distributed into the memory modules to increase reliability. The resulting organization is shown in Figure 3. In a practical system n is usually larger than m. Thus, not all devices can have access to memory simultaneously, however since devices transfer at varying rates, access paths between them can be shared.

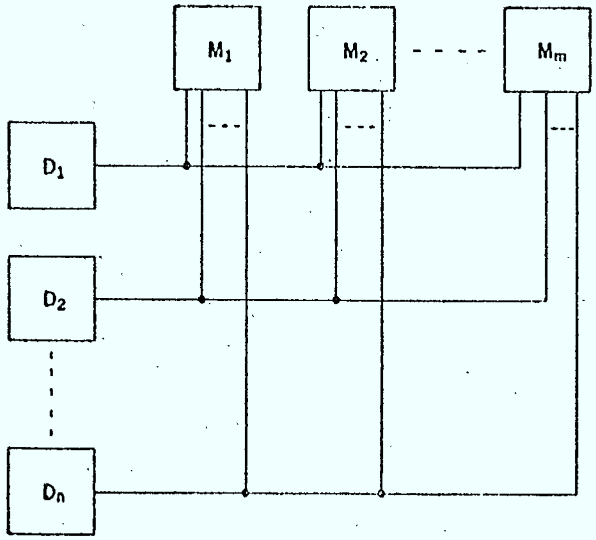


Figure 3: Communication switching distributed into the memory modules for connecting n devices with memory modules.

Shared Connection

Some devices transfer at very low rates of a few bits per second, and others transfer at very high rates of hundreds of thousands to millions of millions of bits per second. Therefore, slower devices can share access paths. Figure 4 shows a method of sharing access paths using timeshared (multiplexed) busses.

The limitation of the number of devices which can multiplex a transfer path is determined by the bandwidth of the path and the transfer rates of the devices. The maximum transfer rate of a device must be less than the bandwidth of a path. The maximum transfer rate of all devices on the system which can transfer concurrently is a function of the memory-bus and memory-box bandwidths. This function is a statistical function based on the traffic flow. If at peak transfer points the transfer rate exceeds that of a bus or memory box, saturation is said to occur. For example, all memory busses may be operating below saturation, but the system can be saturated because all busses are trying to access the same memory module. The design goal is to have the system operate near saturation at all times. Without buffering (temporary storage) of information, a saturation condition may cause information to be lost. The amount of buffering required for a device depends on the statistical properties of the system (the distributions of transfer-rate requirements to the memory modules and busses), the transfer rate of the device, and the cost or possibility of losing information from the device.

There are many tradeoffs between numbers of access paths, bandwidth of access paths, buffering, and the costs of each of these, in order to meet a given information-transfer rate. The concepts discussed above apply to memory-bus structures and to other I/O busses connected to I/O processors, as discussed below.

MULTIPLEXING AN ACCESS PATH

Access to a shared transfer path is commonly governed by a control line which runs serially through all devices on the path (Figure 4). The simplest scheme is to assign priority by position in the control ring. When a transfer is completed, control goes to the device requesting access in the highest priority position. With this approach, devices are assigned positions based on worst-case transfer conditions. However, using such a scheme may prevent the path from utilizing its maximum bandwidth. This rigidity is unnecessary if devices can request access to the path with variable priority.

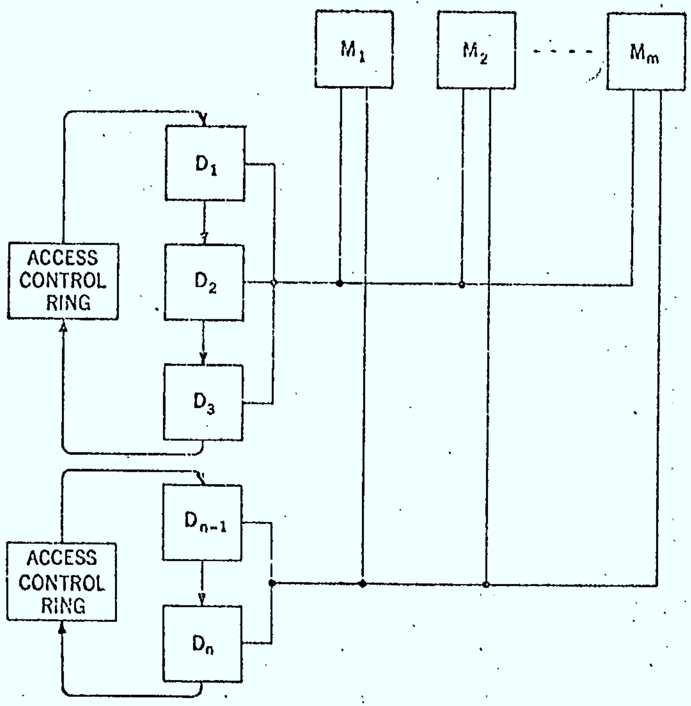


Figure 4: Communication of n devices with m memory modules sharing access paths.

CODES

As has been pointed out in the notes in Volume 1 on "Communications Considerations", character information is coded into a string of bits, both for handling by the computer and for transmission. Many codes are in use, but one code format, the ASCII (American Standard Code for Information Interchange) is rapidly gaining widespread use in North America. It encodes 128 characters (see the Table of the code on page 12 of Volume 4) into 7 binary information units (bits) and then some equipment uses 1 additional bit to give an error-detection ability, thereby encoding characters in 7 or 8 bits.

SYNCHRONOUS AND ASYNCHRONOUS TRANSMISSION

Multiplexing techniques were introduced to increase channel utilization and thus reduce the communication costs in time-sharing systems. Two techniques are commonly used for this purpose. These techniques are also discussed in the volume 1 on Communications Aspects.

1. Synchronous Time Division Multiplexing (STDM)

Example: consider the transmission of messages from terminals to computer, each terminal is assigned a fixed time duration. After one user's time duration has elapsed, the channel is switched to another user. The main advantage of STDM is that buffering is limited to one character per user line and thus addressing is usually not required.

STDM has certain disadvantages which can be realized immediately from the fact that any terminal is assigned a fixed time duration, whether it is idle during this time or not makes no difference to the multiplexer. Thus the channel remains idle during certain time durations (assigned to idle terminals). This is an inefficient utilization of the communication channels. Data collected from several representative operating time sharing systems revealed that during an average call, 95% of the user-to-computer channel and 65% of computer-to-user channel are idle.

2. Asynchronous Time Division Multiplexing (ATDM) (see Fig. 5) (Statistical Multiplexers)

The multiplexer here switches from one user to another user whenever the one user is idle, and asynchronously time multiplexes the data. With such an arrangement, each user would be granted access to the channel only when he has a message to transmit. The crucial attributes of such a multiplexing technique are:

a) an address is required for each transmitted message;

b) buffering is required to handle the statistical peaks in random message arrivals (see figure 6).

An operating example of an ATDM system for analog speech is the "Time assignment speech Interpolation" (TASI) system used by the Bell system on the Atlantic Ocean Cable.

The ATDM systems operate efficiently under the following conditions:

1) Low overflow probability (same or lower order of magnitude as the line error rate).

2) An acceptable expected message queuing delay due to buffering.

To study the overflow probability and the expected message queuing delay, analyses of the statistical behaviour of the buffer are needed (see Chu (1) for detailed discussion of this topic).

We shall discuss now the design aspects of these statistical multiplexers and try to identify the trade-offs among the different design considerations.

An optimal multiplexer is the one that yields minimum operating costs and at the same time satisfies the required performance (delays and overflow probabilities). The operating costs of the multiplexers can be divided into transmission costs and storage costs. The transmission costs are dependent on the transmission rates of the lines and the number of lines used. The storage costs are dependent on the buffer length required, the cost of overhead in buffer management, and the cost of waste spaces of fixed-size messages. The queuing delays imposed by multiplexers are dependent, for example, on delays due to buffering, and computer scheduling algorithms.

Figure 5 shows the components of a statistical multiplexer. The asynchronous multiplexer's buffer behaviour can be analyzed by queuing models with finite waiting lines.

In describing the buffer behaviour, we are interested in the following parameters:

P_{of} $\equiv$ buffer overflow probability (the average fraction of the total number of characters that overflow from the buffer).

P $\equiv$ traffic intensity (a measure of the degree of congestion of the multiplexed channel)

D $\equiv$ expected queuing delay

N $\equiv$ buffer size

$\bar{\ell}$ $\equiv$ the average length of a burst (string of characters).

Studies of several existing time sharing systems revealed some results that relate the previously mentioned parameters (figures 8, 9,10,11,12 and13, Chu (3)).

Three types of messages are commonly considered in the design of the statistical multiplexers (figure 7).

The constant length message model corresponds to the user-to-computer traffic where users type characters one at a time at the terminal.

The random length messages correspond to the computer-to-user traffic. The central processor of a time-sharing computer sequentially performs fractions of each user's job and the output traffic to the users is strings of characters which are called bursts. The random nature of the message length greatly complicates the problem of storage allocation. A way to simplify this problem is to segment messages into fixed-sized blocks. An address as well as linking information to subsequent blocks is assigned to each block. Since each block has an address and is uniform in size, a block can be stored in any vacant position in the buffer. Thus the storage allocation problem is greatly simplified. The wasted storage space created by segmentation is:

1) some number of address characters to identify each block and to link the subsequent blocks.
2) the unfilled space of the last block.

The trade-off between the extra storage cost and the saving in buffer management is the main consideration to decide whether the random length messages should be segmented into fixed-sized blocks.

A design example:

Consider the design of a time-sharing system that consists of many remote terminals and that employs the ATDM technique with full duplex operation between the terminals and the central processor.

The typical value of the character interarrival time per user line can be approximated as exponentially distributed with mean 0.5 seconds (from the study of several operating systems). Thus the character arrivals can be treated as Poisson arrivals with a rate = 2 char/sec. Voice grade private lines can easily transmit 240 char/sec. from users. Suppose this operating system consists of m = 48 terminals, and assume all the terminals to be statistically independent and have the same average traffic characteristics.

The buffer should be designed such that the overflow probability is less than 10^{-6}.

Requirements: determine the buffer size and the queuing delay incurred by each character.

$$P = \text{traffic intensity} = 1.5 \times 48 \times 2/240 = 0.6$$

where total no. of characters/sec = no. of terminals x characters/sec/terminal
= 48 x 2.

and the factor (1.5) is introduced to account for the characters necessary for addressing and framing (assuming 50% of the source information is necessary for that purpose.) From figure 8 the buffer size that corresponds to $P_{of} = 10^{-6}$ and $P = 0.6$ is 14 character length. From figure 9, the normalized queuing delay due to buffering is equal to 1.25 character holding times. Since each character holding time is equal to $\frac{1}{u} = \frac{1}{240}$ m.sec.

= 4.16 millisecond, the waiting time of each character is 5.06 milliseconds.

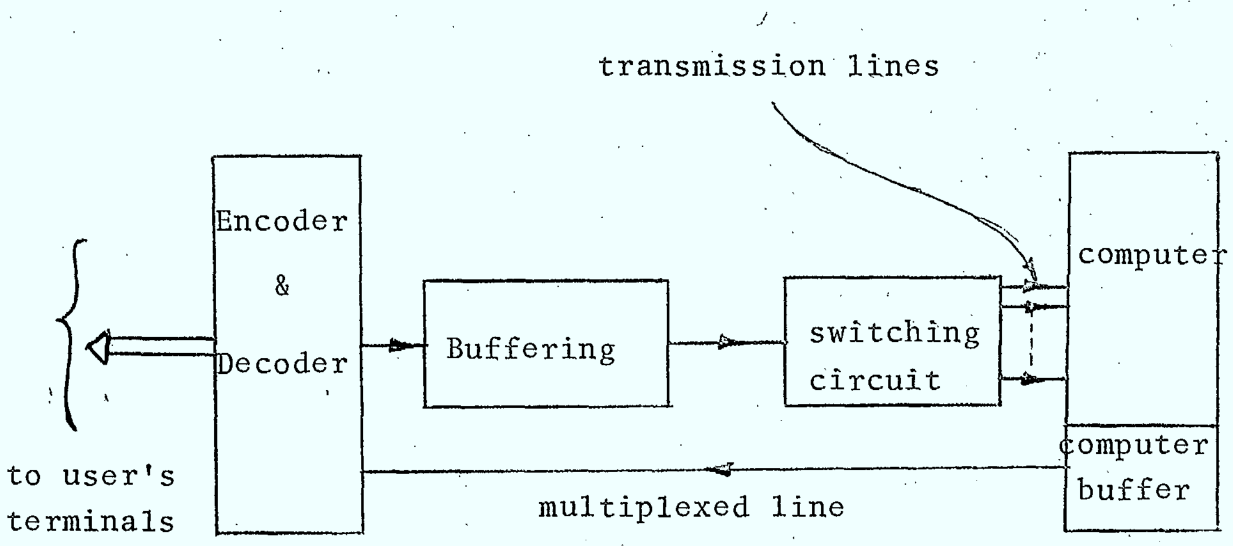


FIGURE 5

Asynchronous time division multiplexing system for time sharing computer communications.

Note: the switch circuit is used when we have more than one transmission line.

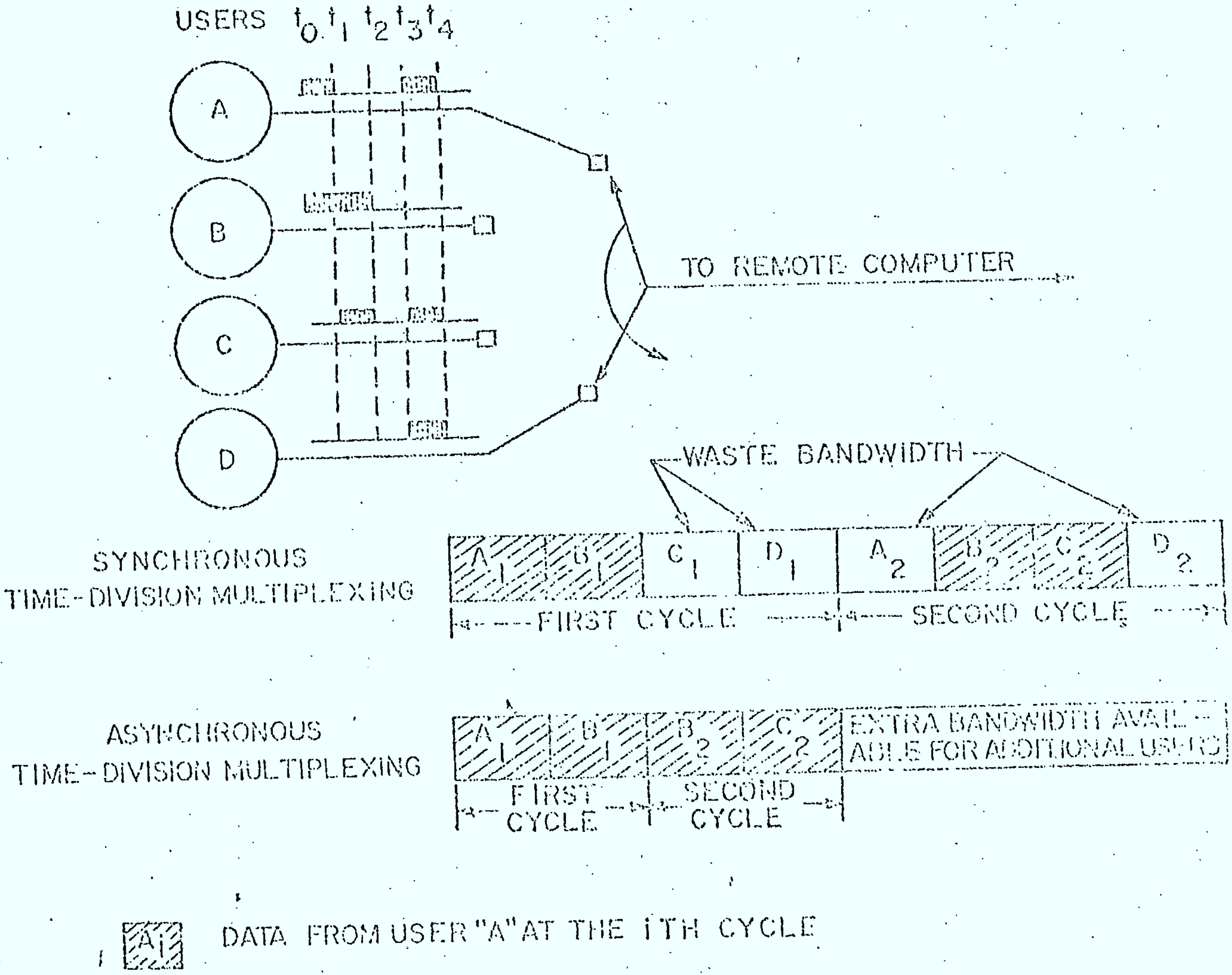
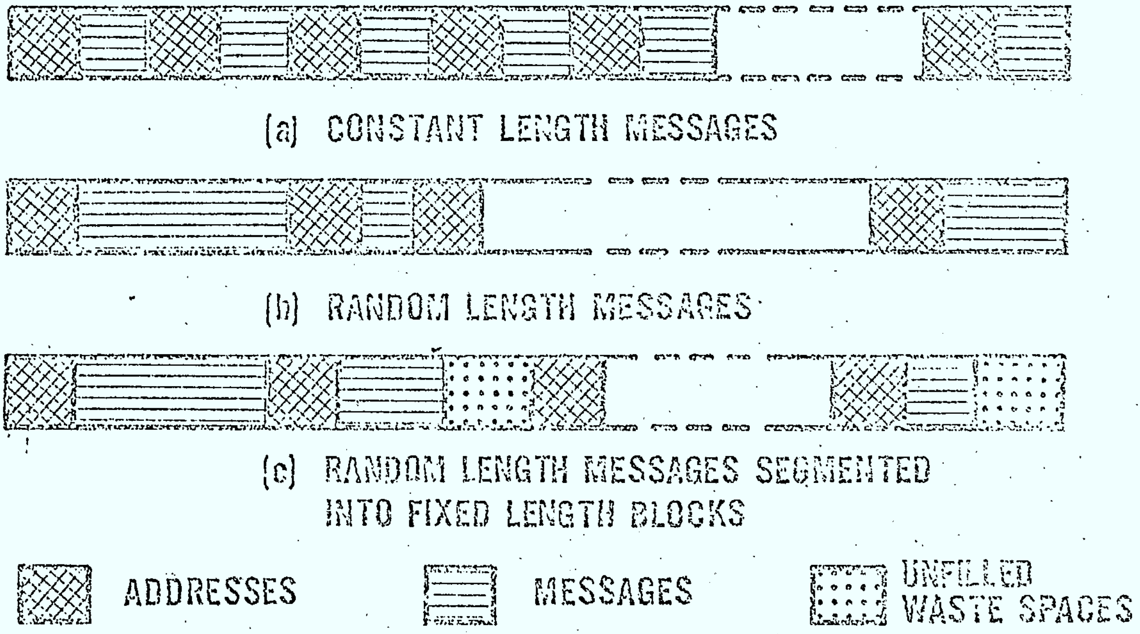

USERS
t_0 t_1 t_2 t_3 t_4
A
B
C
D
TO REMOTE COMPUTER
WASTE BANDWIDTH
SYNCHRONOUS
TIME-DIVISION MULTIPLEXING
A_1 B_1 C_1 D_1 A_2 B_2 C_2 D_2
FIRST CYCLE
SECOND CYCLE
ASYNCHRONOUS
TIME-DIVISION MULTIPLEXING
A_1 B_1 B_2 C_2
EXTRA BANDWIDTH AVAILABLE FOR ADDITIONAL USERS
FIRST CYCLE
SECOND CYCLE
A_i DATA FROM USER "A" AT THE ITH CYCLE

FIG. 6 TIME-DIVISION MULTIPLEXING



(a) CONSTANT LENGTH MESSAGES

(b) RANDOM LENGTH MESSAGES

(c) RANDOM LENGTH MESSAGES SEGMENTED INTO FIXED LENGTH BLOCKS

FIG. 7 THREE TYPES OF DATA STRUCTURES

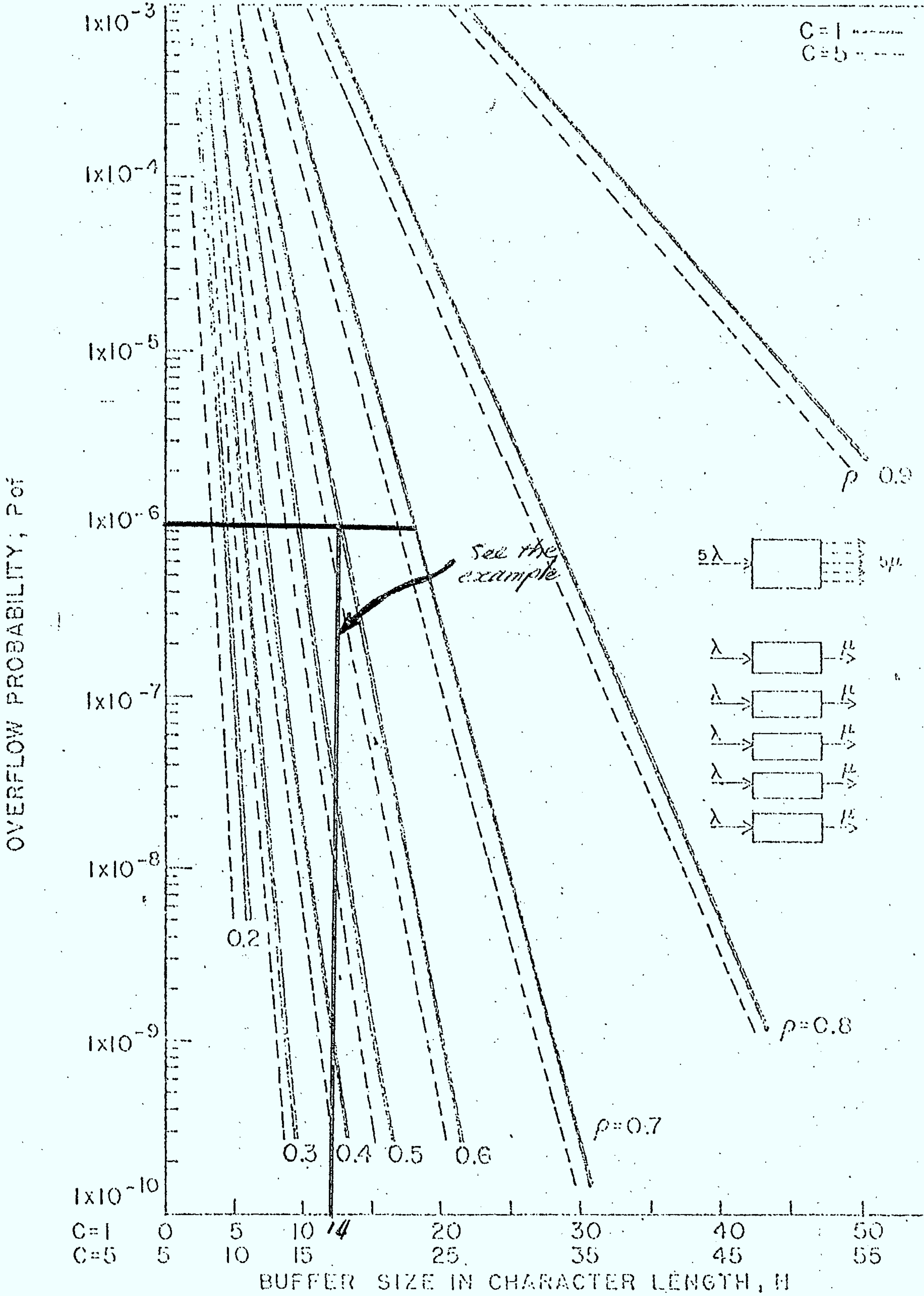
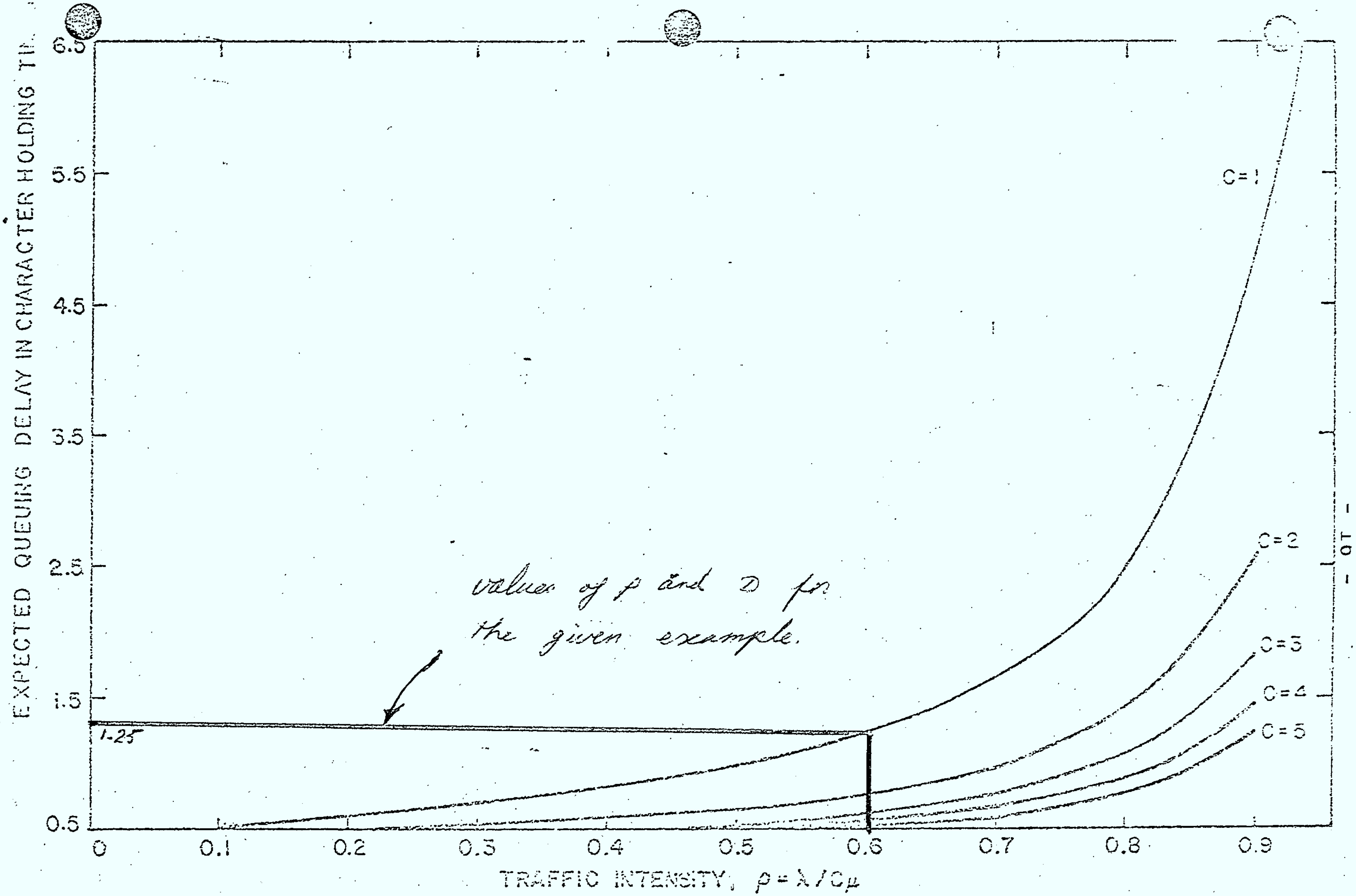


FIG. 8 OVERFLOW PROBABILITY VS BUFFER SIZE



values of ρ and D for the given example.

1.25

FIG. 9 EXPECTED QUEUING DELAY VS TRAFFIC INTENSITY
(AT OVERFLOW PROBABILITY = 10^{-6})

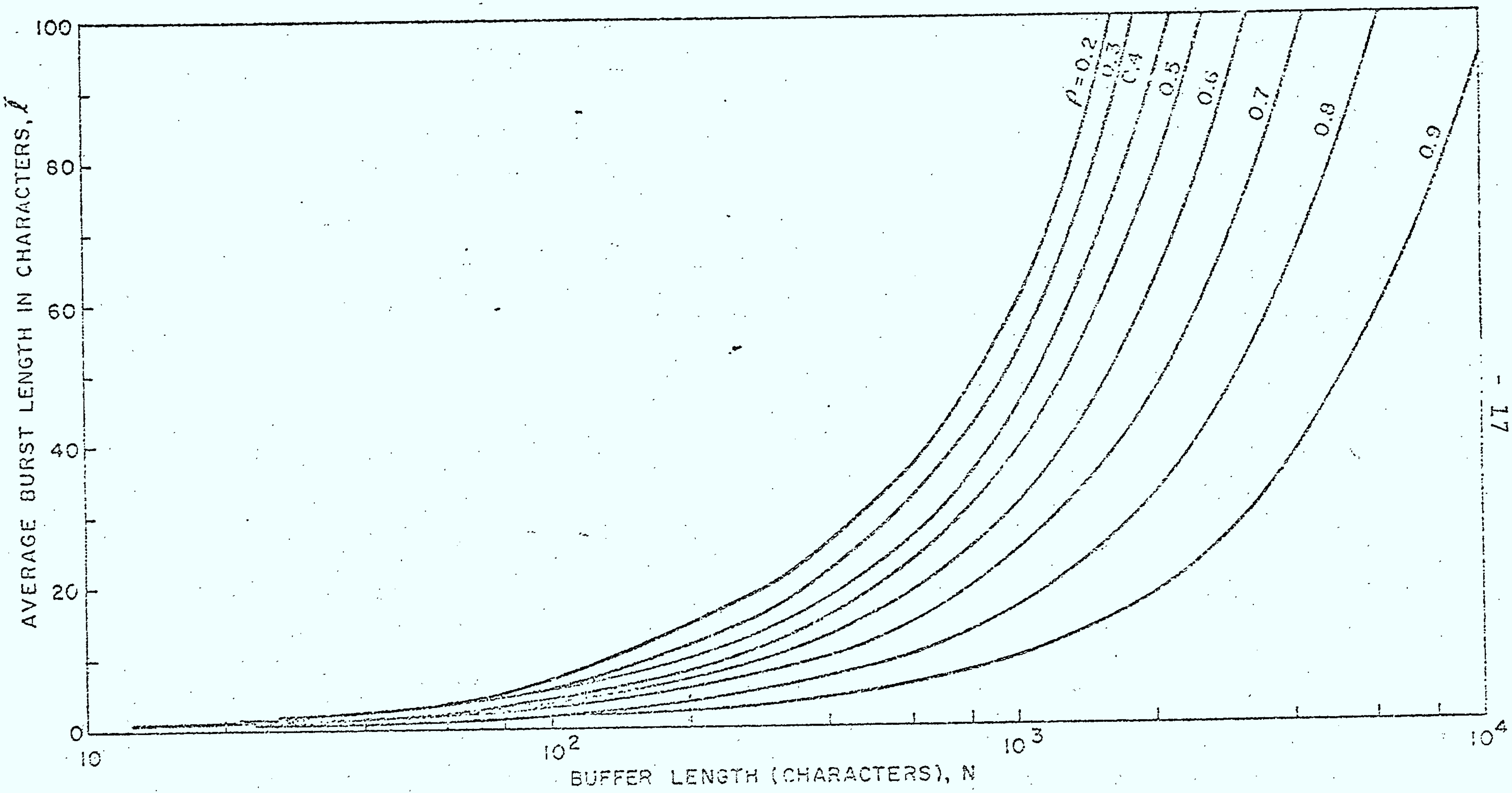


FIG. 10 BUFFER LENGTH VS AVERAGE BURST LENGTH, $P_{OF} = 10^{-6}$

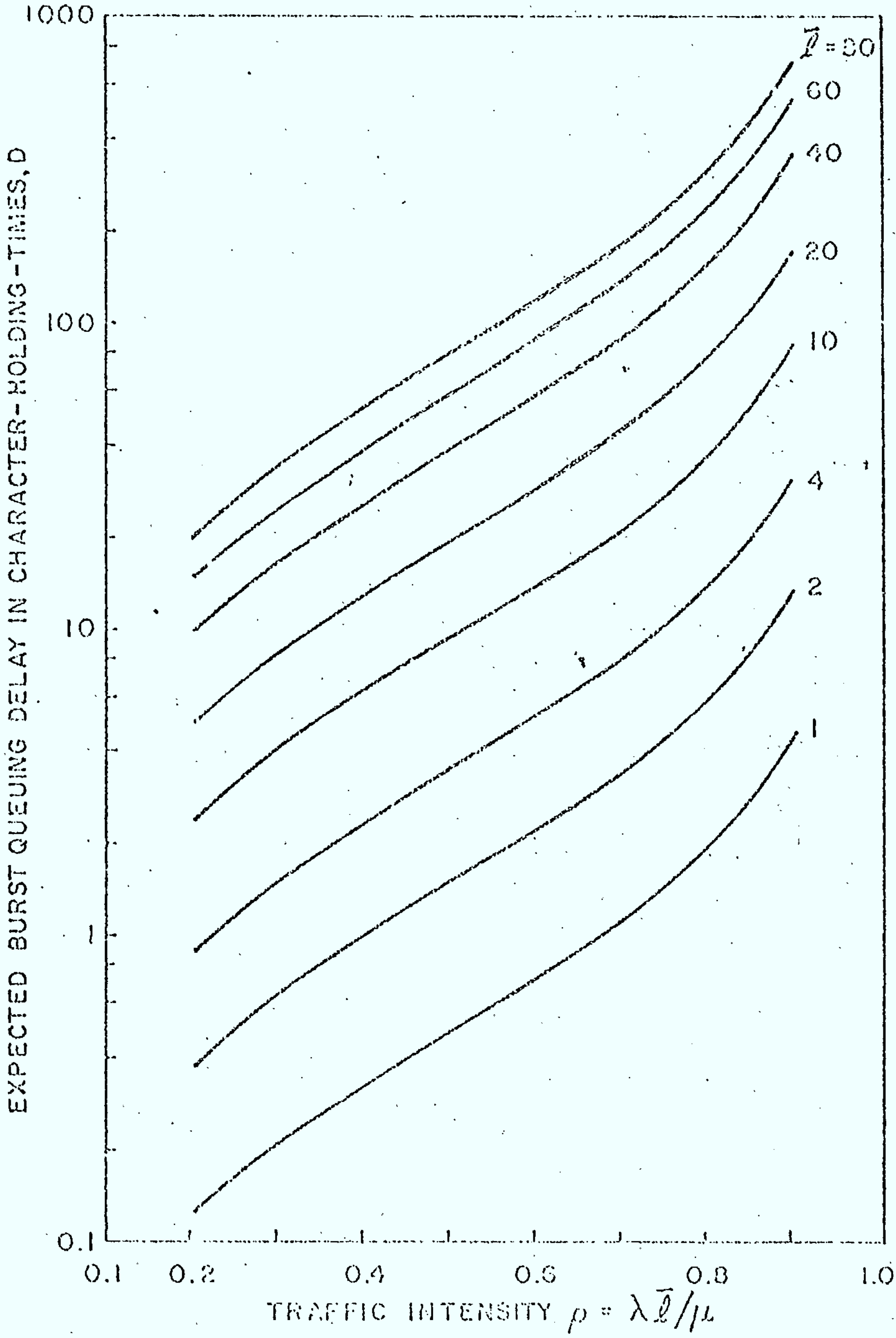


FIG. 11 TRAFFIC INTENSITY VS EXPECTED BURST QUEUING DELAY

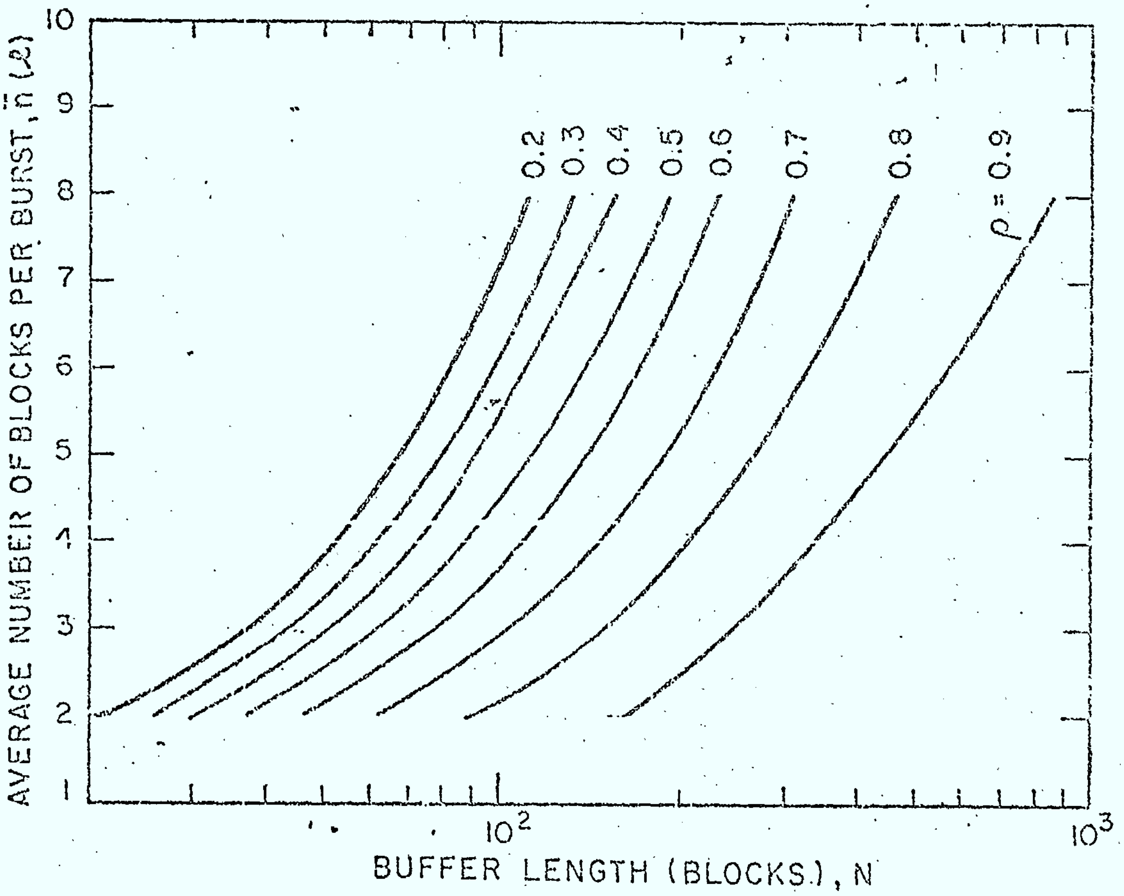


FIG. 12 BUFFER LENGTH VS AVERAGE NUMBER OF BLOCKS PER BURST
(AT OVERFLOW PROBABILITIES = 10^{-6})

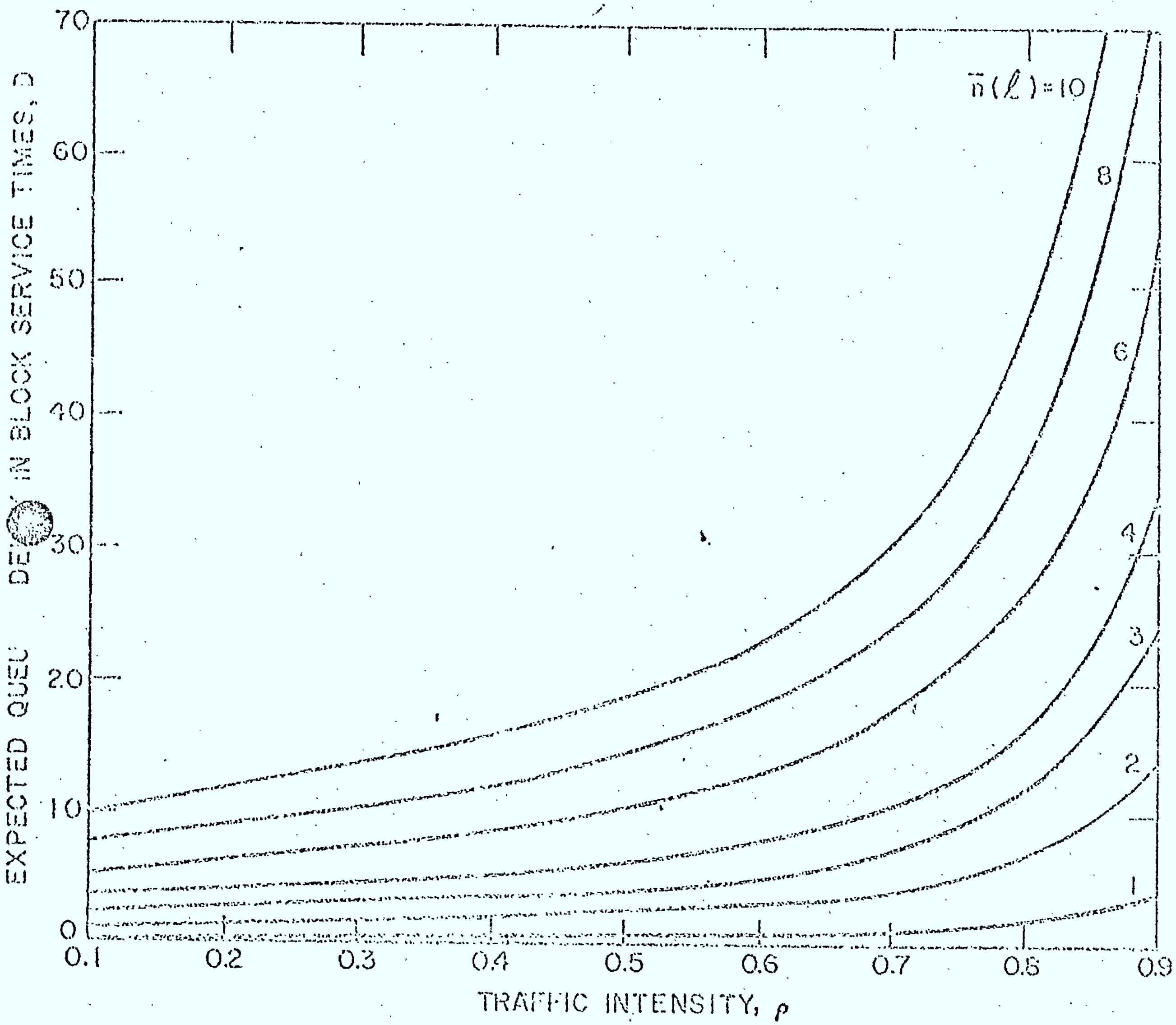


FIG. 13 EXPECTED QUEUING DELAY VS TRAFFIC INTENSITY
(AT OVERFLOW PROBABILITIES = 10^{-6})

Concentrators

Rental of communication lines, particularly high-speed lines over long distance, is expensive and therefore a number of techniques have been developed to share this resource among several terminals. For example, a commercial timesharing service might have a machine in Tel-a-viv and customers in Jerusalem, Haifa and Gaza. The average customer wont want to pay long-distance rates and the timesharing firm could lease one telephone line between each of these places and its central computer and place a concentrator in Haifa, Gaza and Jerusalem. The customer then dials the computer with a local number and is connected with the concentrator. As far as the customer is concerned, he has made direct contact with the computer. The concept is illustrated in Figure 14.

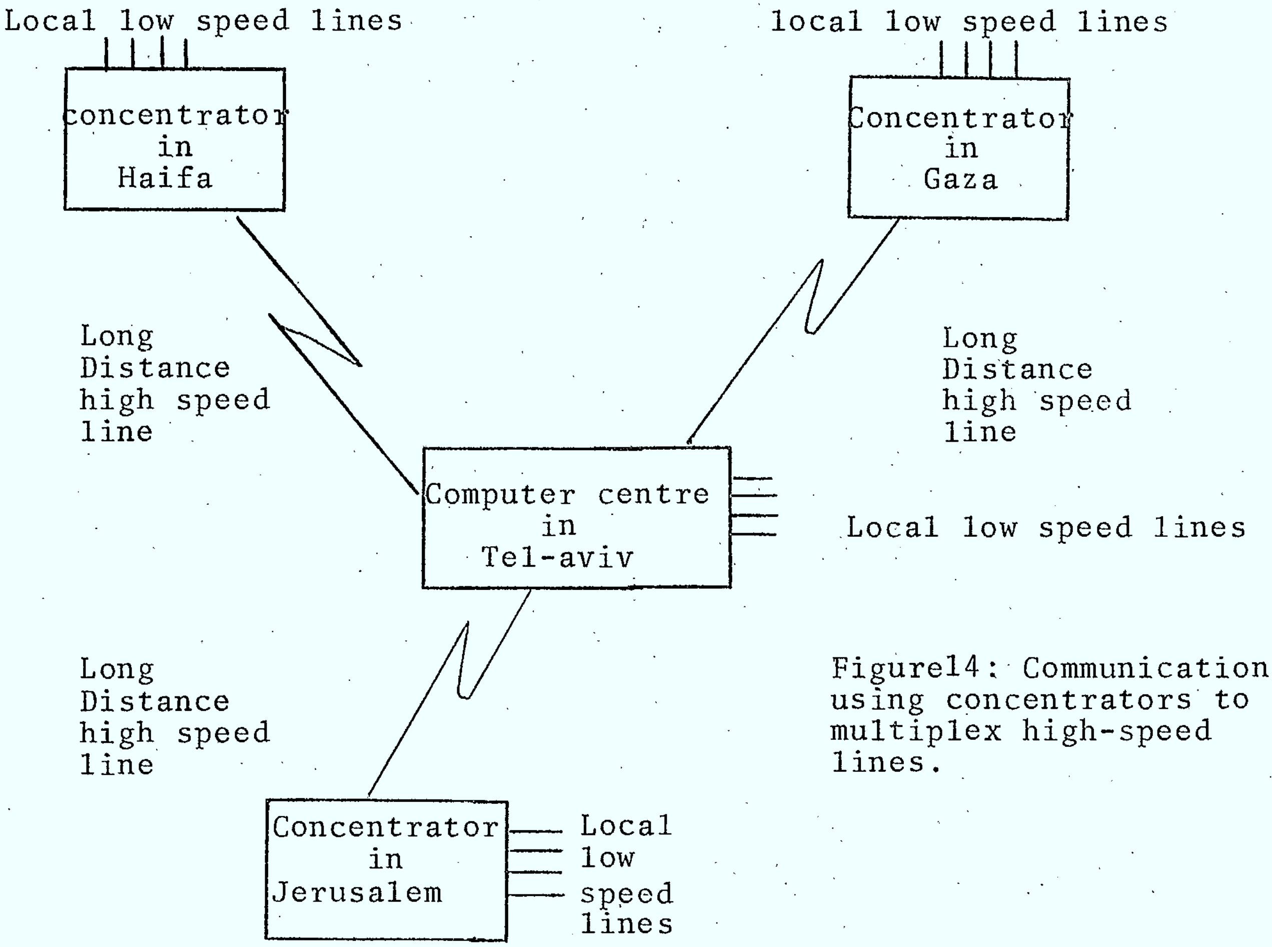

Local low speed lines
concentrator in Haifa
local low speed lines
Concentrator in Gaza
Long Distance high speed line
Long Distance high speed line
Computer centre in Tel-aviv
Local low speed lines
Long Distance high speed line
Concentrator in Jerusalem
Local low speed lines

Figure14: Communications using concentrators to multiplex high-speed lines.

Concentrators are usually small computers that collect messages from the users in a given area, via low-speed asynchronous telephone lines. The concentrator then transmits the messages over the high-speed leased line to the computer, either asynchronously or synchronously. A message contains a special character or characters, such as a carriage return, recognized by the concentrator. When the concentrator recognizes and end-of-message character from a user, it enters the user's message, preceded by information to identify the user, in a buffer. The output of the buffer then feeds a high speed telephone line.

Messages from the central computer are sent to the concentrator preceded by appropriate identification. The concentrator then sorts the messages into buffers for each user and transmits them to the user at slow speed. With this technique, an expensive resource, namely the long distance telephone line, is shared by several users.

Multidrop and Polling Techniques

It is possible to have several concentrators on one line or to attach several terminals to one line. This can be done, using a technique called multidrop for sending information to terminals and called polling for receiving information from terminals. With multidrop, the message is preceded by a device address and followed by an end-of-message character or characters. The message is sent down the line, and all devices decode the address; only the device addressed connects itself to the line. When the end-of-message character is received, the device disconnects itself. The network may be able to precede the message with several addresses or have a broadcast code indicating that several or all devices should receive the message. This concept is illustrated in Figure 15.

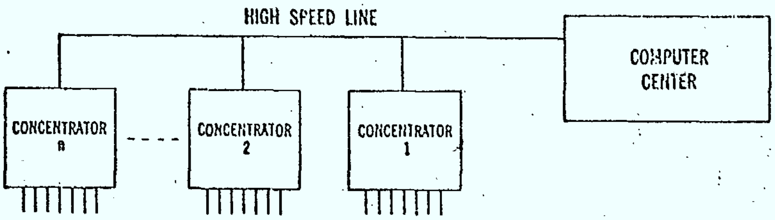


Figure 15: Communications where several concentrators share one line.

Only one device can transmit to the computer at a time, and the line is organized by polling the devices under computer control. A polling message is sent down the line, which asks "terminal X do you have anything to transmit?" Terminal X replies with a code for yes or no. If the answer is "no" the next polling message is sent.

Line Buffers

The incoming serial string of bits at the computer interface must be converted into 8 parallel bits representing a character and stored in a buffer register. To accomplish this function required stripping off the start and stop bits on asynchronous transmission. The error-detection bit may also be utilized to check the transmission at this time. Similarly on output, the parallel bits representing a character must be converted to serial form and have the start/stop bits added. An error-detection bit may also have to be added if the character is represented in the computer without the error-detection bit. These functions are performed by a device commonly called a line buffer. The concept is illustrated in figure 16. The line buffer has additional tasks of sending and receiving control signals to data sets to connect, disconnect and for proper synchronization. The line buffers also receive status information from the modems indicating, for example, whether or not they are still connected to the line. This status information can be tested by the computer, or it can be used to generate a signal if the state of the line changes. These functions are important for automatic answering of the line and to avoid system difficulties on accidental or erroneous disconnect. When such a disconnect occurs, it is important that the user on that line be automatically logged out, in order to free the line and protect the user. There is one line buffer for each line.

Communications Controller

The line buffers must in turn communicate with main memory. This communication is handled by attaching the line buffers to a device commonly called a communications controller.

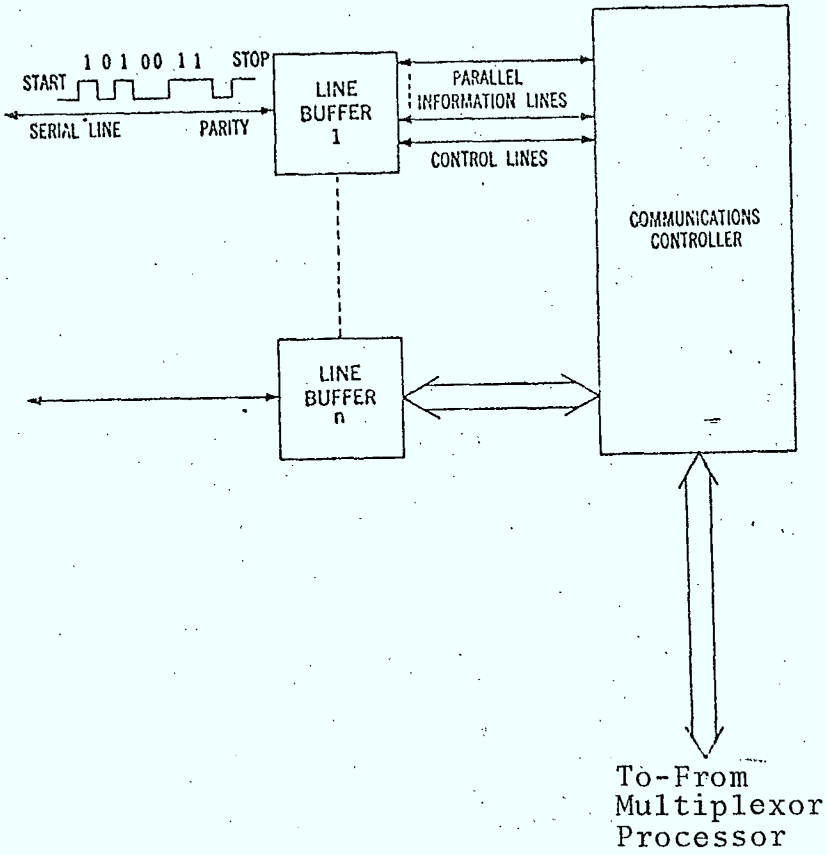

1 0 1 00 1 1
START
STOP
SERIAL LINE
PARITY
LINE BUFFER 1
PARALLEL INFORMATION LINES
CONTROL LINES
COMMUNICATIONS CONTROLLER
LINE BUFFER n
To-From Multiplexor Processor

Figure 16 The communications line interface at the computer.

Timesharing systems are increasingly called upon to handle terminals with a variety of transmission speeds from 110 to several thousand bauds. It is important that new devices be easily interfaced without major hardware or software modifications. These goals cannot be easily achieved with the simple communications controller described above, which scans each line in sequence, and a communications controller with a variable-priority system is usually required. Such a device, after completing a service request, would simultaneously scan all line buffers requesting service and service the one with the highest priority next.

In some systems, the communications controller is connected to a separate small computer which allows a number of processing tasks to be performed, such as editing, message assembly, file creation, and so forth, without interfering with the work of the central machine.

The Terminal Interface

The terminal is the point of interface between the user and the system. It is here that the capabilities of the man and the machine must be matched. There are many terminals available from manufacturers which can usefully be classified into typewriter, cathode-ray tude, and special-purpose terminals. These are discussed in more detail in Volume .

Memory Protection

Memory protection is a problem common to all time-sharing systems. It can be approached in a variety of ways. Protection is needed in general for two reasons, namely intentional and accidental attempts to address the wrong areas of memory. In a system serving a number of users, there is frequently a need to keep information confidential. Human nature being what it is, it is conceivable that one person would want to look in on someone else's information. Mistakes can be made by both humans and equipment. Mistakes can be costly in that they can destroy information, or even disrupt operation.

It is therefore necessary to allow certain areas of memory to be accessed only under tightly-controlled conditions. This protection can be accomplished by software or a combination of software and hardware. Either approach involves a responsibility on the part of the executive program.

The essential technique is to check addresses as they are generated to make certain they are valid. Many time-sharing CPU's include special registers which automatically check addresses. These registers are set by the executive when the program is entered. They define the limits of memory to be accessed by the program. If an address is called for which lies outside the limits, the execution is automatically trapped for checking.

Other memory protection schemes use various types of codes -- or combination locks. A code might be contained within a user number (or determined from a table keyed to user numbers) which would define the areas of memory and storage open to that user. If a user or his program attempted to address any other area, the execution would again be trapped. Codes can be inserted at the beginning of pages to indicate which users are to have access to the information. Such codes could also indicate if the information in the page was to be read only, and not changed, by users. The code could also indicate if the contents were executable by a user but not readable by him. A case in point might be a grading routine in a computer-assisted-instruction situation.

In a random file, it might be necessary to prefix a small code to each item to indicate whether it holds permanent or temporary information. Random files also encounter the possibility of two users requesting the same information -- such as a stock number or airline flight. Such events must be guarded against so that one updating can't occur until the other has been completed. The executive might check a table of information in use and place the second request on a queue until the first transaction has been completed.

Magnetic tape offers a built-in form of protection in that read and write operations can't occur at the same time. Before a tape file is "opened" for either reading or writing, the executive can require a check of the label at the beginning of the tape to ascertain that it is the correct one. When a tape is "closed" a master table can be updated giving the location of data and an indication of its status. Code words could also be used on magnetic tape to prevent the reading of confidential information unless the user or program presented the proper combinations.

Memory protection is usually pronounced in general purpose systems where users have more control over what the computer does. In single application systems, where only precoded programs are used, protection schemes are often less complete. For example, in a savings system, a teller at his terminal could not create instructions to read records from another institution. Thus, protection at the teller level would be necessary. Management terminals might have the capacity to insert special instructions into the system, so memory protection might be necessary at this level.

Control Protection

Certain instructions in the machine are usually not generally available for any process to execute. For example, processes are usually prevented from directly performing I/O instructions because other processes may be using a particular device. All attempts to perform I/O operations generally go through the system so that these requests for service can be scheduled. Further, user processes must be prevented from halting the machine or executing any other instructions which might interfere with the system or other users.

User and System Mode

To handle the above problem, many machines are designed to operate in two modes, usually called system mode and user mode. In system mode, all instructions in the machine are executable. In user mode, certain classes of instructions, called privileged instructions, are prohibited. If an attempt is made to execute a privileged instruction, a fault interrupt is generated which transfer control to the system.

Switching Between User and System Mode

The problem of communications between modes is one requiring careful consideration, because if the design of the mode switching scheme has not been carefully considered, programming can be awkward and time can be wasted in extra checking on the validity of the calls between modes.

A straightforward mode-switching scheme, implemented on the XDS-940 utilizes programmed operations. The programmed-operator concept is in effect a method for making subroutine calls logically appear to the user as machine instructions. When a bit in the instruction word signifying a programmed operator is detected, the bits which are normally interpreted as the operation code are then interpreted as an address to which control is transferred. Two types of programmed operator exist on the XDS 940: 1) a system programmed operator in which the transfer is made to locations in the system code and the mode is switched to system mode (the previous mode is saved) and 2) a user programmed operator in which transfer is made to locations in the user's process. All calls to the system for assistance are thus made with system programmed operators. To return from system mode to user mode, the system executes a jump instruction addressed through the user map rather than the system map. While in system mode, the system can access locations in user processes by use of the user map. Besides giving a simple interface between modes, the programmed operator concept allows the user to think he is programming a machine different and more powerful that that provided by the bare hardware.

Reprinted from –
AFIPS – Conference Proceedings, Volume 35
Copyright © by AFIPS Press
Montvale, New Jersey
07645

A study of asynchronous time division multiplexing for time-sharing computer systems

by W. W. CHU*

Bell Telephone Laboratories, Incorporated
Holmdel, New Jersey

INTRODUCTION

In order to reduce the communications costs in time-sharing systems and multicomputer communication systems, multiplexing techniques have been introduced to increase channel utilization. A commonly used technique is Synchronous Time Division Multiplexing (STDM). In Synchronous Time Division Multiplexing, for example, consider the transmission of messages from terminals to computer, each terminal is assigned a fixed time duration. After one user's time duration has elapsed, the channel is switched to another user. With synchronous operation, buffering is limited to one character per user line, and addressing is usually not required. The STDM technique, however, has certain disadvantages. As shown in Figure 1, it is inefficient in capacity and cost to permanently assign a segment of bandwidth that is utilized only for a portion of the time. A more flexible system that efficiently uses the transmission facility on an "instantaneous time-shared" basis could be used instead. The objective would be to switch from one user to another user whenever the one user is idle, and to asynchronously time multiplex the data. With such an arrangement, each user would be granted access to the channel only when he has a message to transmit. This is known as an Asynchronous Time Division Multiplexing System (ATDM). A segment of a typical ATDM data stream is shown in Figure 2. The crucial attributes of such a multiplexing technique are:

1. An address is required for each transmitted message, and
2. Buffering is required to handle the random message arrivals.**

If the buffer is empty during a transmission interval, the channel will be idle for this interval.

An operating example of an ATDM system for analog speech is the "Time Assignment Speech Interpolation" (TASI) system used by the Bell System on the Atlantic Ocean Cable.[1] Using TASI, the effective transmission capacity has been doubled and the system operates with a negligible (with respect to voice transmission) overflow probability of about 0.5 percent, even without buffering.

The feasibility of the ATDM system depends on: (1) An acceptably low overflow probability—of the same or lower order of magnitude as the line error rate—that can be achieved by a reasonable buffer size, and (2) an acceptable expected message queuing delay due to buffering. To estimate these parameters, analyses of the statistical behavior of the buffer are presented below. The user-to-computer traffic is in

*Present address: Computer Science Dept., UCLA, Los Angeles California, 90024.

** There may be other reasons for providing buffering such as: tolerating momentary loss of signals (e.g., fading), momentary interruptions of data flow, permitting error control on the line, etc. Under these conditions, the buffer should be designed to satisfy also the above specific requirements.

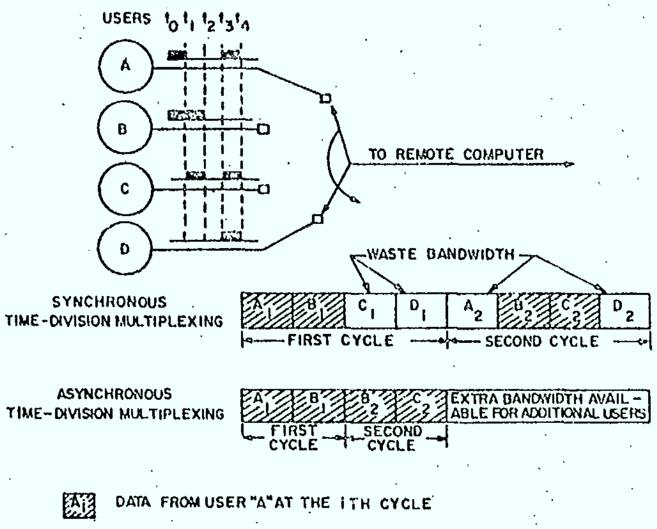
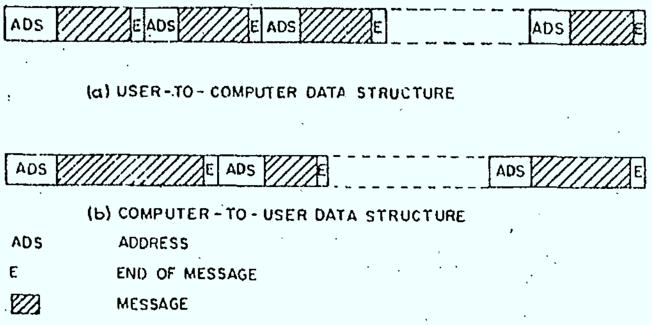

USERS t_0 t_1 t_2 t_3 t_4
A
B
C
D
TO REMOTE COMPUTER
WASTE BANDWIDTH
SYNCHRONOUS TIME-DIVISION MULTIPLEXING
A_1 B_1 C_1 D_1 A_2 B_2 C_2 D_2
FIRST CYCLE
SECOND CYCLE
ASYNCHRONOUS TIME-DIVISION MULTIPLEXING
A_1 B_1 B_2 C_2
EXTRA BANDWIDTH AVAILABLE FOR ADDITIONAL USERS
FIRST CYCLE
SECOND CYCLE
A_i DATA FROM USER "A" AT THE ith CYCLE

Figure 1—Time-division multiplexing



(a) USER-TO-COMPUTER DATA STRUCTURE

(b) COMPUTER-TO-USER DATA STRUCTURE

Figure 2—Asynchronous time division multiplexing data stream

units characters, while the computer-to-user traffic is in units strings of characters which we shall call bursts. The length of the bursts are different from one to another and are treated as random variables. Because of the asymmetrical nature of the traffic characteristics, the statistical behavior of the buffer in the user-to-computer multiplexer and the computer-to-user multiplexer are quite different and, therefore, are treated separately. An example is given to illustrate the multiplexer design in a time-shared computer-communications system that employs ATDM technique.

Analysis of buffer behavior

User-to-computer buffer

An ATDM system consists of a buffer, encoding/decoding circuit, and a switching circuit (in the case of multiple multiplexed lines) as shown in Figure 3. For the analysis of the statistical behavior of user-to-computer buffer, the character (fixed length) arrivals from the sources to the buffer are assumed to be generated from a renewal counting process; that is, the character interarrival times are independent and identically distributed. Since the line transmits with constant speed, the time it takes to transmit each fixed length character (service time), $1/\mu$, is assumed to be constant. For reliability and simplicity in data transmission, synchronous transmission is assumed. The data are taken out synchronously from the buffer for transmission at each discrete clock time. The data arriving at the buffer during the periods between clock times have to wait to begin transmission at the beginning of the next clock time, even if the transmission facility is idle at the time of arrival. In queuing theory terminology, the above system implies there is a gate between the server and waiting room which is opened at fixed intervals. Thus we shall analyze the queuing model† with finite buffer size (waiting line) and synchronous multiple transmission channels (servers). Powell and Avi-Itzhak[2] analyzed a similar queuing model with an unlimited waiting line. Birdsall,[3] and later Dor[4] analyzed a queuing model with limited waiting room but with a single server. In here, the model is generalized to accommodate multiple servers with limited waiting room.

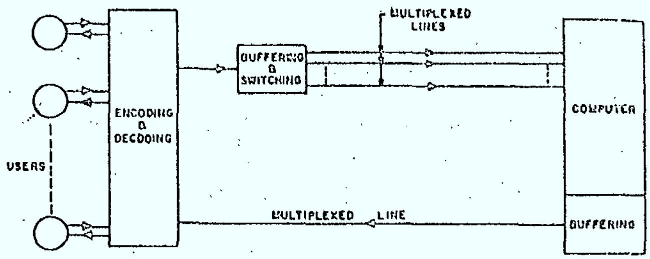


Figure 3—Asynchronous time division multiplexing system for time-sharing computer communications

To establish the set of state equations for analysis of a buffer with a size of N characters and c servers, we assume that the system has reached its equilibrium. Let p_n be the probability that there are exactly n characters in the system (in the buffer and in service) at the end of a service time, and a_c be the probability

† The results derived from this study can also be used as a conservative estimate (upper bound) for the case in which the lines are permitted to transmit the characters arrived during the service interval. The estimate yields better approximation for the heavy than light traffic intensity case. Because under heavy traffic case, the lines are usually all busy and the characters that arrive during the service interval have to wait and cannot be serviced during the service interval. The maximum over design in a buffer system with c transmission lines that permits to transmit the characters arrived during service interval is c characters.

there are no more than c characters in the system at that time, i.e.,

$$a_c = \sum_{i=0}^{c} p_i \qquad (1)$$

Without loss of generality, we can let the service interval equal to unity. We shall express the probability of number of characters present in the buffer at the end of the unit service time interval (left side of equation (2)) in terms of the probability of the number present in the system at the beginning of the interval (right side of equation (2)), multiplied by the probability of a given number of characters arriving during the service interval. As this can occur in different combinations, we add the probabilities. With synchronous transmission, all characters in service would finish their service and leave this system at the end of a service interval.

Thus in a unit service interval of time, we have

$$\left.\begin{aligned}
p_0 &= a_c\pi_0 \\
p_1 &= a_c\pi_1 + p_{c+1}\pi_0 \\
p_2 &= a_c\pi_2 + p_{c+1}\pi_1 + p_{c+2}\pi_0 \\
&\;\;\vdots \\
p_n &= a_c\pi_n + p_{c+1}\pi_{n-1} + \cdots + p_{c+n-1}\pi_1 \\
&\qquad + p_{c+n}\pi_0, \text{ for } n \le N - c \\
&\;\;\vdots \\
p_n &= a_c\pi_n + p_{c+1}\pi_{n-1} + \cdots \\
&\qquad + p_{N-1}\pi_{n+1-(N-c)} + p_N\pi_{n-(N-c)} \\
&\qquad \text{for } N - 1 \ge n > N - c \\
\sum_{i=0}^{N} p_i &= 1
\end{aligned}\right\} (2)$$

Due to limited buffer size,

$$p_{i>N} = 0 \qquad (3)$$

Where

π_n = probability of n characters originating from a renewal counting process during a service interval

N = buffer length in characters

c = number of transmission lines

The first equation describes the case in which the buffer is vacant, if no more than c characters are in transmission at the beginning of the interval, and no arrivals occur during the interval. The second equation describes the case in which one character is in the buffer. if no more than c characters are in transmission at the beginning and one arrives during the service time interval; or there are c + 1 in the buffer at the beginning and no character arrives during the service interval, etc. In the numerical computation carried out in this paper, we assume the character arrivals are generated from a Poisson process; that is, $\pi_n = \exp(-\lambda_u)\lambda_u^n/n!$, where λ_u is the average character arrival rate to the user-to-computer buffer (offered load) from the m independent users. Since the buffer has a finite size of N, $p_{i>N} = 0$. Thus, when a character arrives and finds the buffer is full, an overflow will result. Therefore, the average character departure rate from the user-to-computer buffer (carried load), α_u is less than the offered load from the users λ_u. The carried load can be computed from the buffer busy period

$$\alpha_u = \sum_{i=0}^{c-1} i \cdot p_i + c \sum_{i=c}^{N} p_i \qquad (4)$$

The overflow probability of the user-to-computer buffer, the expected fraction of total number of characters rejected by the buffer, is then equal to

$$P_{of} = \frac{\text{offered load—carried load}}{\text{offered load}} = 1 - \alpha_u/\lambda_u \quad (5)$$

The traffic intensity from user-to-computer, ρ_u, measures the degree of congestion and indicates the impact of a traffic stream upon the service streams. It is defined as

$$\rho_u = \lambda_u/c\mu \qquad (6)$$

Channel (server) utilization, η, measures the fraction of time that the lines are busy. It can be expressed as

$$\eta = (1 - P_{of})\lambda_u/c\mu = \alpha_u/c\mu \le \rho_u \qquad (7)$$

Since physically it is impossible for the transmission lines to be more than 100 percent busy, the utilization is limited to a numerical value less than unity. In the no-loss case (unlimited buffer size), $P_{of} = 0$, then $\eta \equiv \rho$.

The time average queuing length in the user-to-computer buffer, L_u, is equal to

$$L_u = \sum_{i=c}^{N} (i - c)p_i + \lambda_u/2 \text{ characters} \quad \text{for } N > c. \quad (8)$$

The first term in Equation (8) is the expected number of characters in the system at the beginning of a service interval. Since the characters could not leave the system during the service interval, we add the time average number of character arrival (for Poisson arrivals) during the service interval which is $\lambda_u/2$. The expected (time average) queuing delay of each character at the user-to-computer buffer due to buffering, D_u, can be evaluated by using Little's[5] result. We have

$$D_u = L_u/(\lambda_u(1 - P_{of})) \text{ service times} \quad (9)$$

For the single server case, that is, $c = 1$, the set of state equations (2) becomes an imbedded Markov Chain, and can be solved iteratively to obtain the state probabilities as shown in References 3 and 4. For the multiple server case, however, the multiple dependence on the various states prevents us from using the iterative techniques for solution. Thus, the set of state probabilities, p_i's, must be solved from the set of linear matrix equations (2). The overflow probability, queuing delay, and queue length are then computed from the p_i's via Equations 4, 5, 8 and 9.

The size of the matrix (Equation 2) corresponds to the buffer length. The matrix equation was solved by the Gauss elimination method.[6] For purposes of accuracy, double precision was used in all phases of the computation. From the character arrival rate, λ_u, the coefficient values can be computed from (2) and they are stored in the computer program. Due to the limitation of the computer word size, double precision on IBM 360/65 provides 15-digit accuracy. Therefore, when the coefficient value is less than 10^{-15}, it is set equal to zero. The computation time required to solve this type of system equation is largely dependent on its size. For a 10×10 matrix the computation time is about 0.8 seconds, while a 50×50 matrix equation takes about 1.67 minutes.

Numerical results are presented in Figures 4, 5 and 6. These results reveal the relationships among the overflow probabilities, number of transmission lines used, traffic intensities, and buffer sizes.

Computer-to-user buffer

In a previous section, the buffer behavior has been analyzed for a finite queue with multiple server, Poisson arrivals, and constant service time, which corresponds to the users-to-computer traffic. The

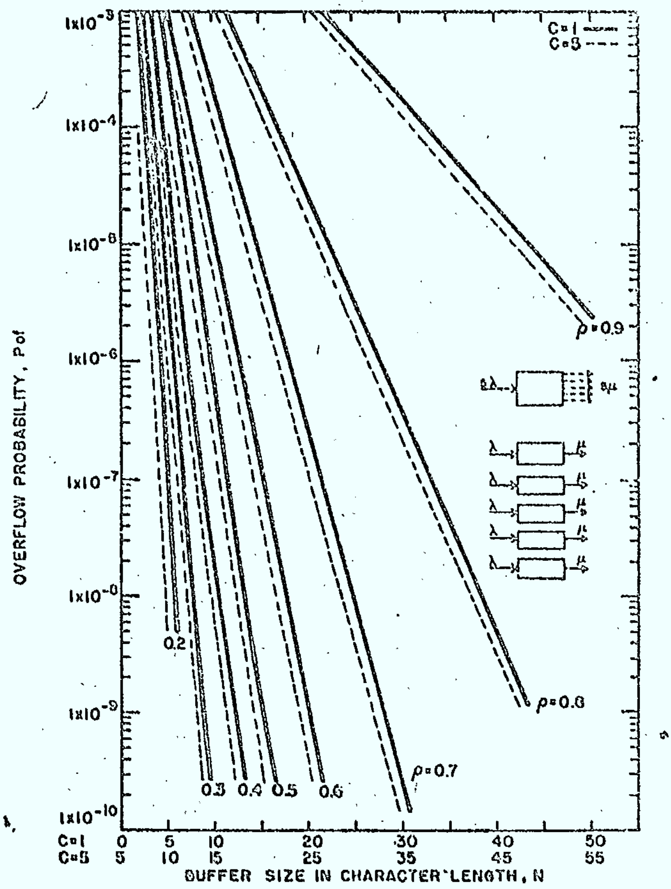


Figure 4—Overflow probability vs buffer size

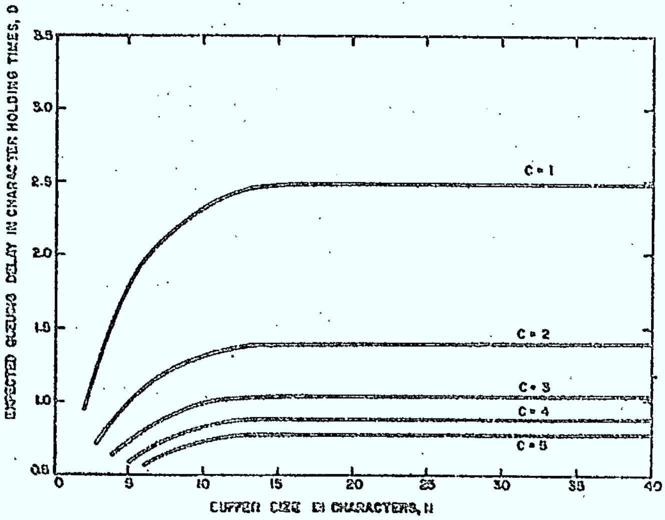


Figure 5—Expected queuing delay vs buffer size

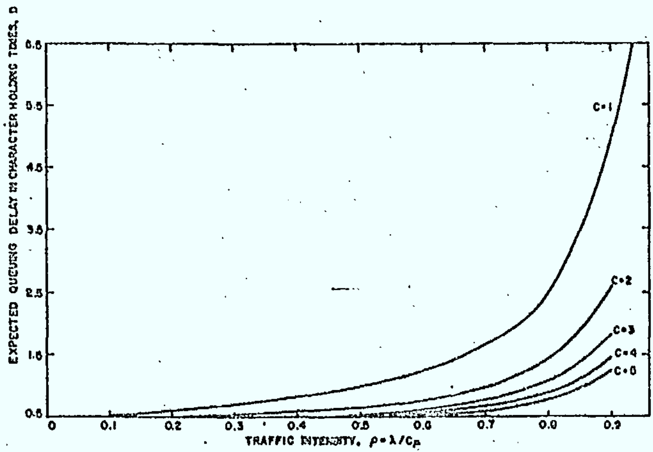


Figure 6—Expected queuing delay vs traffic intensit y

computer-to-user traffic, however, is quite different from the users-to-computer traffic. The central processor of a time-sharing computer sequentially performs fractions of each user's job and the output traffic to the users are strings of characters which we shall call bursts. The length of the bursts are different from one to another and are treated as random variables. It is assumed that the internal processing speed of the computer is very fast as compared to the line transmission speed. Further, it is assumed that the various processing tasks generated by the user-computer interactions are independent from one user to another and have exponential interarrival times for a given user. In ATDM operation with these assumptions, the arrivals of bursts at the common output transmission buffer for the group of users are approximated as random. In this section, we shall analyze this buffer behavior under the assumptions of a finite queue, single server with batch (burst) arrivals, and constant service time.

Using the burst length and traffic intensity as parameters, we would like to find the relationships among the overflow probabilities, expected burst delays due to buffering, and buffer sizes.

Let us consider the case that the burst length, L is geometrically distributed with mean, $\bar{\ell} = 1/\theta$; and the number of bursts arrived during a unit service interval (time to transmit a character from the multiplexed line), N, is Poisson distributed with mean, λ_c bursts/service time. The distributions of L and N are as follows:

$$f_L(\ell) = \theta(1-\theta)^{\ell-1} \qquad \ell = 1, 2, \cdots \qquad (10)$$

$$f_N(n) = \exp(-\lambda_c)\lambda_c^n/n! \qquad n = 0, 1, 2, \cdots \qquad (11)$$

The total number of characters that arrived during the time to transmit a character on the multiplexed line is a random sum, S_N, and is equal to

$$S_N = \sum_{i=0}^{N} L_i \qquad (12)$$

where L_i, a random variable distributed as (10), is the number of characters contained in the ith arriving burst. N, a random variable distributed as (11), is the total number of bursts arriving during the unit service interval. For simplicity in notation, we let $S = S_N$.

The characteristic function of S, $\phi_S(u)$, can be expressed in terms of the characteristic function of L, $\phi_\ell(u)$, and λ_c.

$$\phi_S(u) = \exp[-\lambda_c + \lambda_c\phi_L(u)] \qquad (13)$$

Since the burst lengths are geometrically distributed the characteristic function of L is

$$\phi_L(u) = \theta \cdot \exp(iu) \Big/ \Big(1 - (1-\theta)\exp(iu)\Big) \qquad (14)$$

where $i = \sqrt{-1}$. Substituting (14) into (13), then

$$\phi_S(u) = \exp[-\lambda_c + \lambda_c \cdot \theta \cdot \exp(iu)/ (1-(1-\theta)\exp(iu))] \qquad (15)$$

From (15), it can be shown that the probability density of j characters arriving during a unit service interval, $f(S = j) = f_j$, is a compound Poisson distribution as shown in (16)

$$f_j = f(S = j) = \begin{cases} \displaystyle\sum_{k=1}^{j} \binom{j-1}{k-1} (\lambda_c\theta)^k (1-\theta)^{j-k}\exp(-\lambda_c)/k! & j = 1, 2, \cdots \\ \exp(-\lambda_c) & j = 0 \end{cases} \qquad (16)$$

The expected value of S is given by $E[S] = E[L]E[N] = \lambda/\theta$, and the variance of S is given by

$$\mathrm{Var}[S] = \lambda(2-\theta)/\theta^2 \qquad (17)$$

The time required to compute the probability density function of S, f_j, from (16) is dependent on the size of j. For large j (e.g., $j > 1000$), the computation time could be very large and prohibitive. A convenient and less time consuming way to compute f_j is from $\phi_S(u)$ by using the Fast Fourier Transform[7] inversion method as follows:

$$f_j = \sum_{r=1}^{M} \phi_S(r)\exp[-2\pi irj/M] \qquad j = 0, 1, 2, \cdots, M-1 \qquad (18)$$

where

$r = 2\pi u/M$

M = total number of input points to represent $\phi_S(r)$ = total number of output values of f_j.

In order to accurately determine $\phi_S(r)$, it is computed with double precision on the IBM 360/65. Further, we would like to use as many points as possible to represent $\phi_S(r)$; that is, we would like to make M as large as possible. Because of the word length limitation of the computer, double precision provides 15-digit accuracy. Therefore, when $f_j < 10^{-15}$, it is set equal to zero. M is selected such that $f_{j \geq M} < 10^{-15}$. The M's are different for different values of λ_c and $\bar{\ell}$.

The following is the set of state equations for a buffer size of N characters with batch renewal arrivals, single server, and constant output rate.

$$p_n = \pi_0 p_{n+1} + \sum_{i=1}^{n} \pi_{n-i+1} p_i + \pi_n p_0$$

or

$$p_{n+1} = \frac{1}{\pi_0}\left[p_n - \sum_{i=1}^{n} \pi_{n-i+1} p_i - \pi_n p_0\right] \qquad (19)$$

$$n = 0, 1, 2, \cdots, N-1$$

$$\sum_{i=0}^{N} p_i = 1 \qquad (20)$$

and

$$p_{i>N} = 0 \qquad (21)$$

The above equations are reduced from Equation (2) by letting c=1.

The average character departure rate from the buffer (carried load), α_c, is less than the average character arrival rate to the buffer (offered load), $\beta = \lambda_c/\theta$, from the computer. The carried load can be computed from the probability that the buffer is idle,

$$\alpha_c = 1 - p_0 \qquad (22)$$

The overflow probability of the buffer with burst input, the expected fraction of total number of characters rejected by the buffer, is equal to

$$P'_{of} = \frac{\text{offered load-carried load}}{\text{offered load}} = 1 - \alpha_c/\beta \qquad (23)$$

The traffic intensity from computer-to-user is

$$\rho_c = \beta/\mu = \lambda_c/(\theta\mu) = \lambda_c\bar{\ell}/\mu \qquad (24)$$

The set of state Equations (19) is an imbedded Markov Chain. In the following numerical computations, we shall assume that the character arrivals are generated from a compound Poisson process, i.e., $\pi_i = f_i$. The state probabilities can be solved iteratively and expressed in terms of p_o. From (20), we can find the value of p_o. Thus we find all the state probabilities. The overflow probabilities for various burst lengths can then be computed from (23). These results are presented in Figure 7 which provides the relationships (at $P'_{of} = 10^{-6}$) between burst lengths and buffer sizes for selected traffic intensities.

In the above analysis, we have treated each character as a unit. However, in computing the expected burst delay, D_c, due to buffering, we should treat each burst as a unit. The service time is now the time required to transmit the entire burst. For a line with

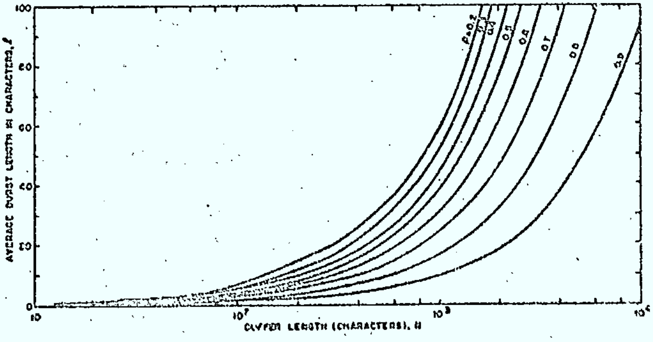


Figure 7—Buffer length vs average burst length, $P'_{of} = 10^{-6}$

constant transmission rate, the service time distribution is the same as the burst length distribution except by a constant transmission rate factor. When overflow probability is very small, for example, $P'_{of} = 10^{-6}$, then D_c can be approximated by the expected burst delay of the infinite waiting room with Poisson Arrivals and single server with geometric service time, M/G/1, model.[8,9] Hence

$$D_c = \frac{\lambda E(L^2)}{2(1-\rho)} = \frac{\lambda_c(2-\theta)}{2(\theta-\lambda_c)} \quad \text{character-holding times} \quad (25)$$

where $E(L^2)$ = second moment of burst length, L. The delays are computed from (25) for selected traffic intensities and burst lengths. Their results are portrayed in Figure 8.

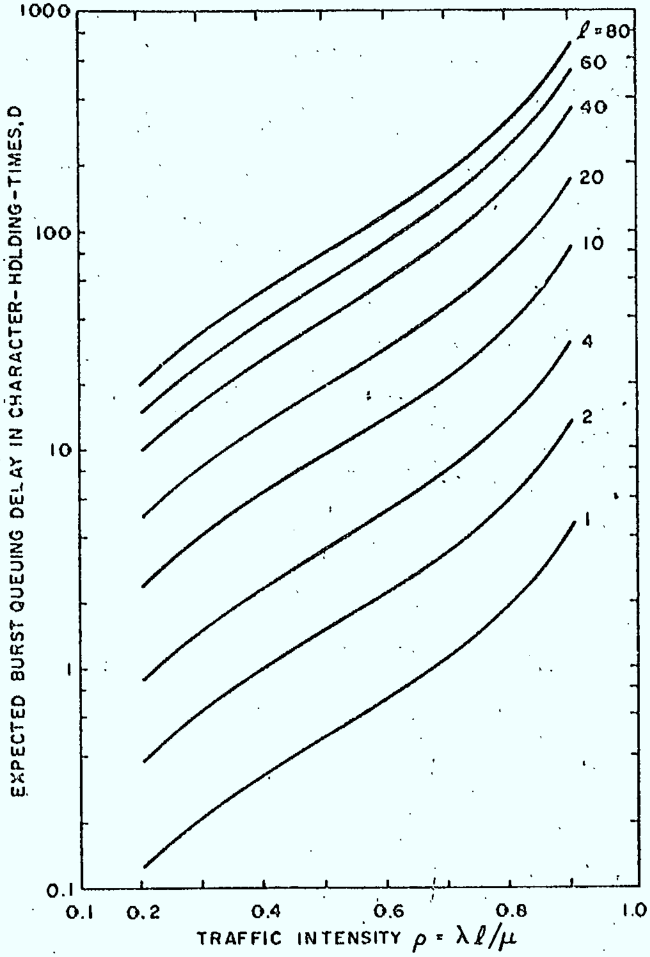


Figure 8—Traffic intensity vs expected burst queuing delay

Discussion of results

We shall first discuss the user-to-computer buffer behavior. Figure 4 portrays the relationships between overflow probabilities and buffer size for selected traffic intensities and selected numbers of servers. The curves for two-, three-, and four-servers lie in the region between the single and the five-server curves. For a given traffic intensity, the overflow probability decreases exponentially with buffer size. For a typical traffic intensity of 0.8 ,a buffer of twenty-eight character length will achieve an overflow probability in the order of 10^{-6}. A larger buffer size is needed for $\rho_u > 0.8$ in order to achieve the same degree of buffer performance. For a given ρ, the queuing delay increases as the overflow probability decreases (or the buffer size increases). When the overflow probability is less than 10^{-4} (for $\rho_u = 0.8$, this overflow probability corresponds to a buffer size of about eighteen characters), the delay increment with buffer length becomes negligible and the delay can be approximated as independent of buffer size as shown in Figure 5.

For the data transmissions in time-sharing systems, the buffer overflow probability should be somewhat less than the line error rate. For currently available lines, the error rate is about 10^{-5}. Therefore from Figure 5, we know that the queuing delay range of interest is almost independent of the buffer length. Figure 6 describes the queuing delays (at overflow probability = 10^{-6}) for various traffic intensities. The queuing delay increases exponentially with ρ. For a given ρ, the queuing delay decreases with the increase of number of servers. Figures 4 and 6 agree with our intuition that whenever multiple servers are needed, it is always advantageous to use a common buffer rather than using several single lines with separate buffers.

Next we shall discuss the computer-to-user buffer behavior. The overflow probability depends upon the buffer size, the traffic intensity, and expected burst length. For a given average buffer length, the overflow probability increases as the traffic intensity increases. For a given traffic intensity, and a desired buffer overflow probability, the required buffer size increases as the average burst length increases. Figure 7 provides the relationships between the average burst length and required buffer size to achieve an overflow probability of 10^{-6} for selected traffic intensities.

When the average burst length equals unity, then the result reduces to the case of Poisson arrivals, single server and constant service time as had been analyzed.[3,4] For a given traffic intensity, required buffer size for average burst lengths $\bar{\ell}(\bar{\ell} > 1)$, $N_{\bar{\ell}}$, to achieve the same degree of overflow probability is much greater than that for unity burst length, N_1. In general, $N_\ell > \bar{\ell} \times N_1$. As $\bar{\ell}$ increases, the difference between N_ℓ and $\bar{\ell} \times N_1$ increases. For example, for $\rho_c = .8$, $\bar{\ell} = 1$, the required buffer size to achieve $P'_{of} = 10^{-6}$ is $N_1 = 28$ characters. When $\bar{\ell} = 4$, then from Figure 7, $N_4 = 212 > 4 \times 28 = 112$ characters. In the same manner, if $\ell = 20$, $N_{20} = 1200 > 20 \times 28 = 580$ characters. This is due to the fact that the variance of S is proportional to $\bar{\ell}$ as shown in (17). Figure 8 portrays the relationship between expected burst queuing delay and traffie intensity for selected expected burst lengths. For a given expected burst length, the expected queuing delay increases as traffic intensity increases; for a given traffic intensity, the expected queuing delay increases with burst length. These are important factors that affect the delay.

Optimal design of multiplexing system

Let us first consider the design of the user-to-computer multiplexer. Based on the user-to-computer traffic characteristics, the number of user terminals, maximum allowable queuing delay, and overflow probability, several different buffer system configurations might satisfy the desired requirements. Hence there are trade-offs among the number of transmission lines we might use, the transmission rates of the lines, and the buffer sizes. We would like to design the multiplexing system whose total cost (transmission cost and buffer storage cost) is minimum. One way to proceed with this is first to select the set of possible multiplexing system configurations based on the queuing delay requirements from Figure 6. Based on the maximum allowable overflow probability, we can obtain the required buffer length for this set of possible multiplexing system configurations. The optimal user-to-computer part of the multiplexing system can then be selected as that which minimizes the cost of the system.

Next, we shall consider the optimizations of the computer-to-user multiplexer. Data collected from several operating time-sharing systems[10] revealed that the average number of characters sent by the computer to the group of users is an order of magnitude greater than the number of characters sent by the group of users to the computer. Thus, using high transmission rate line for computer output data would significantly reduce in buffer size and the queuing delay due to buffering. Further, the change in the computer system such as changes in the scheduling algorithm[11–17] in the central processor can strongly influence the computer output traffic statistics, which will directly affect the buffer performance, and the design of the decoding system.

In practice, we would like to design a system that has minimum total cost yet satisfies all the requirements such as the inquiry-response delay, average holding time of each user, etc. Since the multiplexing system and the central processor intimately interact with each other, the multiplexing system should be treated as a subsystem of the time-shared computer system. The economical and performance optimization should be carried out jointly between the central processor and available communication facilities.

Example

Consider the design of a time-sharing system that consists of many remote terminals and that employs the ATDM technique with full duplex operation between the terminals and the central processor. Measurements of the traffic characteristics from several operating systems have revealed that the character interarrival time per user line can be approximated as exponentially distributed with mean about 0.5 seconds.[10] Thus, the character arrivals can be treated as Poisson arrivals with a rate of 2 char/sec. A reasonable conservative guess is that 50 percent of the transmitted information is sufficient for addressing and framing. Voice-grade private lines can easily transmit 240 char/sec from users. Suppose this operating system consists of $m = 48$ terminals, all the terminals are assumed to be independent and have the same traffic characteristics. The buffer is designed such that the overflow probability is less than about 10^{-6}. We shall use our model to determine the buffer size and the average queuing delay incurred by each character.

The traffic intensity is $\rho_u = 1.5 \times m\lambda_u / c\mu_u = 1.5 \times 48 \times 2/240 = 0.6$. To achieve the desired overflow probability, from Figure 4, the required buffer length is 14 characters. From Figure 6, the normalized queuing delay due to buffering is equal to 1.25 holding times. Since each holding time is equal to $1/\mu_u = 1/240 = 4.16$ millisecond, the waiting time of each character is 5.06 milliseconds. Now suppose the number of terminals is increased from 48 to 96. In order that traffic intensity be less than unity, two transmission lines are required and the traffic intensity is still equal to 0.6. From Figure 5, the buffer length corresponding to the desired overflow probability for two transmission lines is about 14 characters. The waiting time is about 0.3 holding times which is equal to 3.33 milliseconds. Although the difference between 5.06 milliseconds and 3.33 milliseconds may not be detected by a user at a terminal, a common buffer of the same size operating with two output lines can handle twice the number of input lines as with one output line. Thus, the common buffer approach permits handling a wide range of traffic without substantial variation in buffer size.

Next, we shall consider the buffer design problem that employs the ATDM technique to transmit data from central processor to remote terminals. The traffic statistics as well as the message length are different from that of the users. The burst interarrival time[10] can be approximated as exponentially distributed with a mean of 2.84 seconds. Thus, the bursts can be approximated as Poisson arrivals with a rate of $\lambda_c =$ 0.35 bursts/sec. Further, data collected in the same study indicate that the burst length can be approximated as geometrically distributed with a mean of $\bar{l}$ = 20 characters. Suppose we use a wideband transmission line that transmits 480 char/sec to provide communications from the central processor to 48 terminals. Assuming 20 percent of the transmitted information is used for addressing and framing, then the traffic intensity, $\rho_c = 1.2\lambda_c\bar{l},\mu_c \approx 0.84$. To achieve an overflow probability of 10^{-6}, from Figure 7, we find that the required buffer size is 1,400 characters. From Figure 8, the expected queuing delay for each burst is 85 character-holding times, or 85/480 = 0.176 seconds.

Suppose now we changed our transmission rate from 480 to 960 char/sec; then the traffic intensity $\rho_c \approx 0.42$. The corresponding required buffer size in order to achieve an overflow probability of 10^{-6} is 480 characters, and the delay is 15 character-holding times or 16 milliseconds. Thus, these results also provide insight regarding the trade-off between transmission costs and storage costs.

The above example is based on the output traffic characteristics of a specfic computer scheduling algorithm. As the output traffic statistics changes with different scheduling algorithms, the buffer performance in the multiplexing system is affected. To design an optimal system, we should jointly optimize the scheduling algorithm and the multiplexing system such that yield minimum total cost and also meet the required system performance such as maximum allowable inquiry-response delay, desired overflow probability, etc.

CONCLUSIONS

Queuing analyses indicate that for an allowable overflow probability and queuing delay, moderate buffer sizes can be achieved for asynchronous time division multiplexing for time-sharing computer systems. Further, when multiple transmission lines are required, better buffer performance will be achieved by using a common buffer rather than by using separate ones.

Because of the asymmetric nature of the traffic characteristics of user-to-computer transmission versus computer-to-user transmission, a much larger buffer is required for the computer-to-user multiplexer to handle the larger volume of data generated by the central processor.

The multiplexing system and the central processor in a time-shared environment directly interact with each other. To design an optimal operating system, we should jointly optimize the central processor and the multiplexing system (for example, the interaction between scheduling algorithm and buffer performance) to obtain a minimum cost system that meets the system performance requirements. It is apparent that closer coordination between the computer and communication system designs would be fruitful in terms of economics and technological improvements to the overall system design.

ACKNOWLEDGMENTS

The author wishes to thank E. Fuchs and D. Heyman of Bell Telephone Laboratories for their helpful discussions.

REFERENCES

1 K BULLINGTON J M FRASER
Engineering aspects of TASI
B S T J March 1959 353-364

2 B A POWELL B AVI-ITZHAK
Queuing system with enforced idle time
Operations Research Vol 15 No 6 Nov 1967 1145-1156

3 T G BIRDSALL et al
Analysis of asynchronous time multiplexing of speech sources
IRE Trans on Communications Systems Dec 1962 390-397

4 N M DOR
Guide to the length of buffer storage required for random (Poisson) input and constant output rates
IEEE Trans on E C Oct 1967 683-684

5 J D C LITTLE
A proof of the queuing formula $L = \lambda W$
Operations Research Vol 9 1961 383-387

6 R W HAMMING
Numerical methods for scientists and engineers
McGraw-Hill Book Co Inc N Y 1962 363-364

7 W M GENTLEMAN G SANDE
Fast fourier transforms—for fun and profit
Proc FJCC Vol 29 563-578

8 N U PRABHU
Queues and inventories
John Wiley and Sons Inc N Y 1965 42

9 P M MORSE
Queues Inventories and Maintenance

John Wiley and Sons Inc 1958 15-18

10 P E JACKSON C D STUBBS
A study of multiaccess computer communications
Proc SJCC Vol 34 1969 491-504

11 A L SCHERR
An analysis of Time-Shared Computer Systems
MIT Research Monograph No 36 MIT Press Cambridge Mass 1967

12 P E DENNING
Effect of scheduling on file memory operations
Proc SJCC Vol 30 1967 9-21

13 J E SHEMER
Some mathematical considerations of time-sharing scheduling algorithms
J ACM Vol 14 No 2 April 1967 262-272

14 E G COFFMAN JR
Analysis of two time-sharing algorithms designed for limiting swapping
J ACM July 1968

15 E G COFFMAN L KLEINROCK
Feedback queuing models for time-shared system
J ACM Vol 15 No 4 Oct 1968 549-576

16 L KLEINROCK
Certain analytic results for time-shared processors
Proc IFIP Congress 1968 Edinburgh Scotland Aug 5-10 1968 D119-D125

17 W W CHU
Optimal file allocation in a multicomputer information system
Proc IFIP Congress 1968 Edinburgh Scotland Aug 5-10 F80-85

ACM Symposium on Problems in the Optimization of Data Communications Systems

October 13-16, 1969
Pine Mountain, Georgia

p 35-60 Chu, Wesley W.

Design Considerations of Statistical Multiplexors[*]

Wesley W. Chu

Computer Science Department
University of California
Los Angeles, California

ABSTRACT

Design considerations for statistical multiplexors with Poisson message arrivals and the following three types of messages are considered: 1) constant length messages, 2) random length messages, and 3) segmented random length messages with fixed-size blocks. The operating costs of the multiplexors can be divided into transmission costs and storage costs. The transmission costs are dependent on the transmission rates of the lines and the number of lines used. The storage costs are dependent on the buffer length required, the cost of overhead in buffer management, and the cost of waste spaces of fixed-size block messages. The queuing delays imposed by multiplexors are dependent, for example, on delays due to buffering and computer scheduling algorithms. The trade-offs among transmission costs, storage costs, and delays are described. An optimal multiplexor is the one that yields minimum operating costs yet satisfies the required performances (e.g., delays, overflow probabilities). Some design considerations of multiplexing systems are then illustrated via examples.

[*] This work was done while the author was with Bell Telephone Laboratories, Holmdel, New Jersey.

INTRODUCTION

In order to reduce the communciations costs in time-sharing systems and multiple computer communications systems, multiplexing techniques have been introduced to increase channel utilization. A commonly used technique is synchronous time division multiplexing (STDM). In synchronous time division multiplexing, each terminal is assigned a fixed time duration for the transmission of messages from the terminal to the computer. After one user's time duration has elapsed, the channel is switched to another user. With appropriately designed synchronous operation, required buffering is limited to one character for each transmission line, and addressing is usually not required. The STDM technique, however, often has certain disadvantages. As shown in Figure 1, it is inefficient in channel capacity, and so transmission cost, to permanently assign a segment of bandwidth that is utilized only a portion of the time. Data collected from several representative operating time-sharing systems [1] revealed that during an average call, 95 percent of the user-to-computer channel and 65 percent of computer-to-user channel are idle. These idle periods could be made available for additional users. A more flexible system that efficiently uses the transmission facility is to statistically multiplex the data. With such an arrangement, each user will be granted access to the channel only when he has a message to transmit. This is known as asynchronous time division multiplexing system (ATDM) [2]. The crucial attributes of such a statistical multiplexing technique are: 1) an address is required for each transmitted message, and 2) buffering is required to handle statistical peaks in random message arrivals.*

If the buffer is empty during a transmission interval, the channel will be idle for this interval. Queuing analyses indicate that an allowable overflow probability and queuing delay can be achieved by a moderate size buffer [2]. Design trade-offs among such parameters as transmission rates, number of lines used, buffer lengths, queuing delay, and computer scheduling algorithms are discussed in the following sections.

ANALYSIS OF STATISTICAL MULTIPLEXORS

A statistical multiplexor consists of a buffer, coding/decoding circuit, and a switching circuit (in the case of multiple lines). The buffer behavior can be analyzed by queuing models with finite waiting lines. Data measured from several operating time-sharing systems shows that the message arrivals can be approximated as a Poisson process [3]. The three types of messages (Figure 2) considered are: 1) constant length messages, 2) random length messages, and 3) segmented random length messages with fixed-sized blocks.

* There may be other reasons for providing buffering such as: tolerating momentary loss of signals (e.g., fading), momentary interruptions of data flow, permitting error control on the line, etc. Under these conditions, the buffer should be designed to satisfy also the above specific requirements.

The parameters of interest in describing the buffer behavior are: the buffer behavior overflow probabilities, P_{of}, which is the average fraction of the total number of characters that overflow from the buffer; traffic intensity, ρ, which gives a measure of the degree of congestion of the multiplexed channel(s); expected queuing delay, D; buffer size, N; and average burst (a string of characters) length, $\bar{\ell}$.

For reliability and simplicity in data transmission, synchronous transmission is assumed. The data are taken out synchronously from the buffer for transmission at each discrete clock time. When the buffer and server are empty (transmission facility is idle) at the beginning of a clock time, data arriving at the buffer during the period between clock times have to wait to begin transmission at the beginning of the next clock time. In queuing theory terminology, the above system implies there is a gate between the server and waiting room which is open at fixed intervals.

The constant length message model corresponds to the user-to-computer traffic where users type characters one at a time at the terminals. The service time, $1/\mu$, corresponds to the time to transmit a constant length character. The relationships among the overflow probabilities, number of transmission lines used, traffic intensities, and buffer sizes are portrayed in Figures 3 and 4 [2]. We noted that when multiple transmission lines are required, better buffer performance will be achieved by using a common buffer rather than by using separate ones.

The random length messages correspond to the computer-to-user traffic. The central processor of a time-sharing computer sequentially performs fractions of each user's job and the output traffic to the users is strings of characters which we shall call bursts. The lengths of the bursts can be approximated as geometrically distributed [3]. Again, the service time is the time to transmit a constant length character. The number of characters arriving at the buffer during a unit service time is compound Poisson distributed [2]. The buffer behavior can be analyzed by a finite waiting room queuing model with Poisson batch arrivals and constant service time [2]. The relationships, at $P_{of} = 10^{-6}$, between the average burst lengths and buffer sizes for selected traffic intensities are portrayed in Figure 5.

When the average burst length equals unity, then the result reduces to the case of Poisson arrivals, single server and constant service time as previously have been analyzed [4,5]. For a given traffic intensity, the required buffer size, N_{ℓ}, for average burst lengths $\overline{\ell}$ ($\overline{\ell} > 1$), to achieve the same degree of overflow probability is much greater than that of unity burst length, N_1. In general, $N_{\ell} > \overline{\ell} \times N_1$. For example, for

$\rho = 0.8$, $\bar{\ell} = 1$, the required buffer size to achieve a $P_{of} = 10^{-6}$ is $N_1 = 28$ characters. When $\bar{\ell} = 20$ characters, then from Figure 5, $N_{20} = 1400 > 20 \times 28 = 580$ characters. Further, as $\bar{\ell}$ increases, the difference between N_ℓ and $\bar{\ell} \times N_1$ increases. This is due to the fact that the variance of the total number of characters arriving at the buffer during a unit service time is proportional to $\bar{\ell}$. Figure 6 shows the relationships between expected queuing delay for each burst with various traffic intensities.

In the above, we have discussed the buffer behavior for random length message inputs. The random nature of the message length, however, greatly complicates the problem of storage allocation. A way to simplify this problem is to segment messages into fixed-sized blocks [6]. An address as well as linking information to subsequent blocks is assigned to each block as shown in Figure 2. Since each block has an address and is uniform in size, a block can be stored in any vacant position in the buffer. It is not necessary to rearrange the buffer to obtain an adequate space for the entire burst. Thus, the storage allocation problem is greatly simplified. The wasted storage space created by segmentation is: 1) some number of address characters, b, to identify each block and to link the subsequent blocks, and 2) the unfilled space of the last block.

When the message length is geometrically distributed, the optimal segmented block size, B', that minimizes the waste space [6] is

$$B' = \sqrt{2b\overline{\ell}} - b/3 + \frac{31b^{3/2}}{9\sqrt{2\overline{\ell}}} + O(1/\overline{\ell}) \qquad (1)$$

The actual block size B is equal to the sum of the segmented block size and the required address length; that is

$$B = B' + b \qquad (2)$$

We shall now study the buffer behavior for the segmented fixed-size block inputs. It can be shown that when burst lengths are geometrically distributed with parameters p and q = 1 = p, and when the burst is segmented into fixed-sized blocks B', then the number of blocks per burst is also geometrically distributed with parameter $Q = q^{B'}$ and P = 1 - Q. Each block consists of many characters. Although each character is synchronously transmitted, service for the block may be assumed to be asynchronous (i.e., if the channel is idle, the message arriving at the buffer is transmitted immediately) as a good approximation.* For Poisson burst arrivals with geometrically distributed burst lengths, the number of segmented

* When the block is very short then we should assume the block service is synchronous and the buffer behavior is the same as the random message case, except characters now should be viewed as blocks.

blocks arriving during unit service interval (the time required to transmit a fixed-sized block of characters) is compound Poisson distributed. Using the finite waiting room queuing model with compound Poisson arrivals, and asynchronous constant service [7], the relationships among block overflow probabilities (the average fraction of the total number of blocks that overflow from the buffer), average number of blocks per burst, $\overline{n}(\ell)$, and buffer sizes are portrayed in Figure 7. The average delay of each block for selected traffic intensities is shown in Figure 8.

DESIGN TRADE-OFFS

Let us first consider the design of the multiplexor with constant length message inputs. Based on the average arrival rates, the number of user terminals, maximum allowable queuing delay, and overflow probability, several different buffer system configurations might satisfy the desired requirements. Hence there are trade-offs among the number of transmission lines we might use, the transmission rates of the lines, and the buffer sizes. We would like to design the multiplexing system whose total cost (transmission cost and buffer storage cost) is minimum. One way to proceed with this is first to select the set of possible multiplexing system configurations based on the queuing delay requirements from Figure 4. Based on the maximum allowable overflow probability, we can obtain the required buffer length for this set of possible multiplexing system configurations. The optimal multiplexing system can then be selected as that which minimizes the cost of the system.

Next we shall consider the optimization of the multiplexor with random message length inputs. Since buffer size is dependent on traffic intensity, a high transmission rate would decrease traffic intensity and would significantly reduce the buffer size and the queuing delay. Hence there is a trade-off between transmission cost and storage cost. Further, changes in the computer system such as changes in scheduling algorithms [8-13] of the central processor can strongly influence the computer output traffic statistics, which will directly affect the buffer performance, and its output statistics. In turn, the output statistics affect the design of the demultiplexor. For example for a multiplexor with random inputs and constant service time, the instantaneous output rate can be less than the instantaneous input rate, however, the output traffic statistics are now statistically dependent on each other.* Statistically correlated output traffic from a multiplexor could greatly affect the buffer performance of a demultiplexor. For the same average input rate and service time, random message arrivals yield better buffer performance

* For a queuing system with Poisson arrivals, exponential service time, the output process is still a Poisson process with the same rate as the input process [14]. Further, this is also the only type of process has this unique property.

than that of correlated message inputs. This is because correlated messages tend to form clusters. The degree of correlation may be dependent on the traffic intensity of the multiplexor, which is dependent on computer scheduling algorithms.

Finally, we shall consider the buffer behavior of the fixed-sized block inputs. Due to the extra linking information required as well as the wasted space in the last unfilled block, the storage requirements will be larger than that of the random length message inputs. On the other hand, since the messages have addresses and are uniform in size,it is not necessary to rearrange the buffer to obtain an adequate space for the entire burst. Thus, the storage allocation problem is greatly simplified. The trade-off between the extra storage cost and the saving in buffer management is the main consideration to decide whether the random length messages should be segmented into fixed-sized blocks.

There are two ways to segment the random messages into fixed-sized blocks: Instantaneous segmentation and holding segmentation. The former method transmits fixed-sized message blocks immediately after their segmentations, regardless of whether the block is full. The latter method transmits message blocks instantaneously if these blocks are full, but hold for a period of time, τ, before transmission if a block is not full. The holding time not only provides trade-off between performance (delay) and storage costs, but also influences the output traffic statistics, which in turn affect the design of demultiplexor.

In practice, we would like to design a system that has minimum total cost yet satisfied all the requirements such as the inquiry-response delay, average holding time of each user, etc. Since the multiplexing system and the central processor intimately interact with each other, the multiplexing system should be treated as a subsystem of the time-shared computer system. The economical and performance optimization should be carried out jointly between central processor and the available communication facilities.

EXAMPLE

Consider the design of a time-sharing system that consists of many remote terminals and that employs the ATDM technique with full duplex operation between the terminals and the central processor. Measurements of the traffic characteristics from several operating systems have revealed that the character interarrival time per user line can be approximated as exponentially distributed with mean about 0.5 seconds [3]. Thus, the character arrivals can be treated as Poisson arrivals with a rate of 2 char/sec. A reasonable conservative guess is that 50 percent of the source information is sufficient for addressing and framing. Voice-grade private lines can easily transmit 240 char/sec from users. Suppose this operating system consists of m = 48 terminals, all the terminals are assumed to be statistically independent and have the same average traffic characteristics. The buffer is designed such that the overflow probability is less than about 10^{-6}. We shall use our model to determine the buffer size and the queuing delay incurred by each character.

The traffic intensity is $\rho_u = 1.5 \times 48 \times 2/240 = 0.6$. To achieve the desired overflow probability, from Figure 3, the required buffer length is 14 characters. From Figure 4, the normalized queuing delay due to buffering is equal to 1.25 character-holding times. Since each character-holding time is equal to $1/\mu_u = 1/240 = 4.16$ millisecond, the waiting time of each character is 5.06 milliseconds. Now suppose the number of terminals is increased from 48 to 96. In order that traffic intensity be less than unity, two transmission lines of the same capacity are required, and the traffic intensity is still equal to 0.6. From Figure 3, the buffer length corresponding to the desired overflow probability for two transmission lines is about 14 characters. The waiting time is about 0.8 holding times which is equal to 3.33 milliseconds. Although the difference between 5.06 milliseconds and 3.33 milliseconds may not be detected by a user at a terminal, a common buffer of the same size operating with two output lines can handle twice the number of input lines as with one output line. Thus, the common buffer approach permits handling a wide range of traffic without substantial variation in buffer size.

Next, we shall consider the buffer design problem that employs the ATDM technique to transmit data from central processor to remote terminals. The traffic statistics as well as the message length are different from that of the users. The burst interarrival time [3] can be an approximated as exponentially distributed with a mean of 2.84 seconds. Thus, the bursts can be approximated as Poisson arrivals with a rate of $\lambda_c = 0.35$ bursts/sec. Further, the burst length can be approximated as geometrically distributed with a mean of $\bar{\ell} = 20$ characters. Suppose we use a wideband transmission line that transmits 480 char/sec to provide communications from the central processor to 48 terminals. Assuming 20 percent of the transmitted information is used for addressing and framing, then the traffic intensity, $\rho_c = 1.2 \cdot 0.35 \cdot 20/480 \approx 0.84$. To achieve an overflow probability of 10^{-6}, from Figure 5, we find that the required buffer size is 1,400 characters. From Figure 6, the expected queuing delay for each burst is 85 character-holding times, or $85/480 = 0.176$ seconds.

Suppose now we change our transmission rate from 480 to 960 char/sec; then the traffic intensity $\rho_c \approx 0.42$. The corresponding required buffer size in order to achieve an overflow probability of 10^{-6} is 480 characters, and the delay is 15 character-holding times or 16 milliseconds. Thus, these results also provide insight regarding the trade-off between transmission costs and storage costs.

For the convenience of buffer management, we would like to segment the random length messages into fixed-sized blocks. Suppose the required address messages is about 20 percent of the average burst length; that is, $b = 0.2 \times 20 = 4$ characters. From Equation 1, we find that the optimal segmented block size, $B' = 12$ characters. Hence the actual optimal block size $B = 12 + 4 = 16$ characters. The average number of blocks per burst, $\bar{n}(\ell) = \bar{2}.32$ blocks/burst. Suppose a wideband transmission line that transmits 960 characters per second is used to provide communications from the central processor to 48 terminals The traffic intensity is $\rho = 48 \cdot 0.35 \cdot 2.32 \cdot 16/960 = 0.65$. To achieve an overflow probability of 10^{-6}, from Figure 7, the required buffer size is 65 blocks or 1040 characters. From Figure 8, the average delay for each block is 4.5 block-service times or 75 milliseconds. Comparing with the random message inputs, the required buffer size for the fixed-sized blocks is about twice as large as that of the random length message inputs.

The above example is based on the output traffic characteristics for a specific round robin computer scheduling algorithm. As the output traffic statistics changes with different scheduling algorithms, the buffer performance in the multiplexing system is affected. To design an optimal system, we should jointly optimize the scheduling algorithm and the multiplexing system such that yields minimum total cost and also meet the required system performance such as maximum allowable inquiry-response delay, desired overflow probability, etc.

CONCLUSION

Statistical multiplexors and central processors in a time-sharing environment directly interact with each other. To design an optimal operating system, we should jointly optimize the central processor performance and the multiplexing system to obtain a minimum cost system that meets the system performance requirements. For example, the interaction between scheduling algorithms and buffer performance and the trade-off between the extra storage cost of fixed-size blocks with the overhead of buffer management are significant in planning an ATDM time sharing system. Further research along these lines would be desirable. It is apparent that a closer coordination between the computer and communication system designs would be fruitful in terms of economics and technical improvements to the overall system design.

REFERENCES

1. P. E. Jackson and C. D. Stubbs, "A Study of Multiaccess Computer Communications," Proc. AFIPS, 1969 Spring Joint Computer Conference, pp. 491-504.

2. W. W. Chu, "A Study of Asynchronous Time Division Multiplexing for Time-Sharing Computer Systems" Proc. AFIPS, 1969 FJCC, Las Vegas, Nevada.

3. E. Fuchs and P. E. Jackson, "Estimates of Distributions of Random Variables for Certain Computer Communication Traffic Models," to be presented at ACM Symposium on Problems in the Optimization of Data Communications System, October 13-16, 1969; submitted for publication in J. of the Association for Computing Machinery.

4. T. G. Birdsall, et. al., "Analysis of Asynchronous Time Multiplexing of Speech Sources," IRE Trans. on Communications Systems, December 1962, pp. 390-397.

5. N. M. Dor, "Guide to the Length of Buffer Storage Required for Random (Poisson) Input and Constant Output Rates," IEEE Trans. on Electronic Computers, October 1967, pp. 683-684.

6. E. Wolman, "A Fixed Optimum Cell-Size for Records of Various Lengths," J. of the Association for Computing Machinery, Vol. 12, No. 1, January 1965, pp. 53-70.

7. W. W. Chu, "Buffer Behavior of Poisson Batch Arrivals, Single Server, and Asynchronous Constant Service Time," to be published.

8. A. L. Scherr, "An Analysis of Time-Shared Computer Systems" PhD Diss. MIT, Cambridge, Massachusetts, June 1969.

9. J. E. Shemer, "Some Mathematical Considerations of Time-Sharing Scheduling Algorithms" J. of ACM, Vol. 14, No. 2, April 1967, pp. 262-272.

10. E. G. Coffman, Jr., "Analysis of Two Time-Sharing Algorithms Designed for Limiting Swapping," J. of ACM, July 1968.

11. E. G. Coffman and L. Kleinrock, "Feedback Queuing Models for Time-Shared System," J. of ACM, Vol. 15, No. 4, October 1968, pp. 549-576.

12. L. Kleinrock, "Certain Analytic Results for Time-Shared Processors," Proc. of IFIP Congress 1968, Edinburgh, Scotland, August 5-10, 1968, pp. D119-D125.

13. P. E. Denning, "Effect of Scheduling on File Memory Operations," Proc. AFIPS, 1967 SJCC, Vol. 30, pp. 9-21.

14. P. J. Burke, "The Output of a Queuing System," Operations Research, Vol. 4, pp. 699-704, 1956.

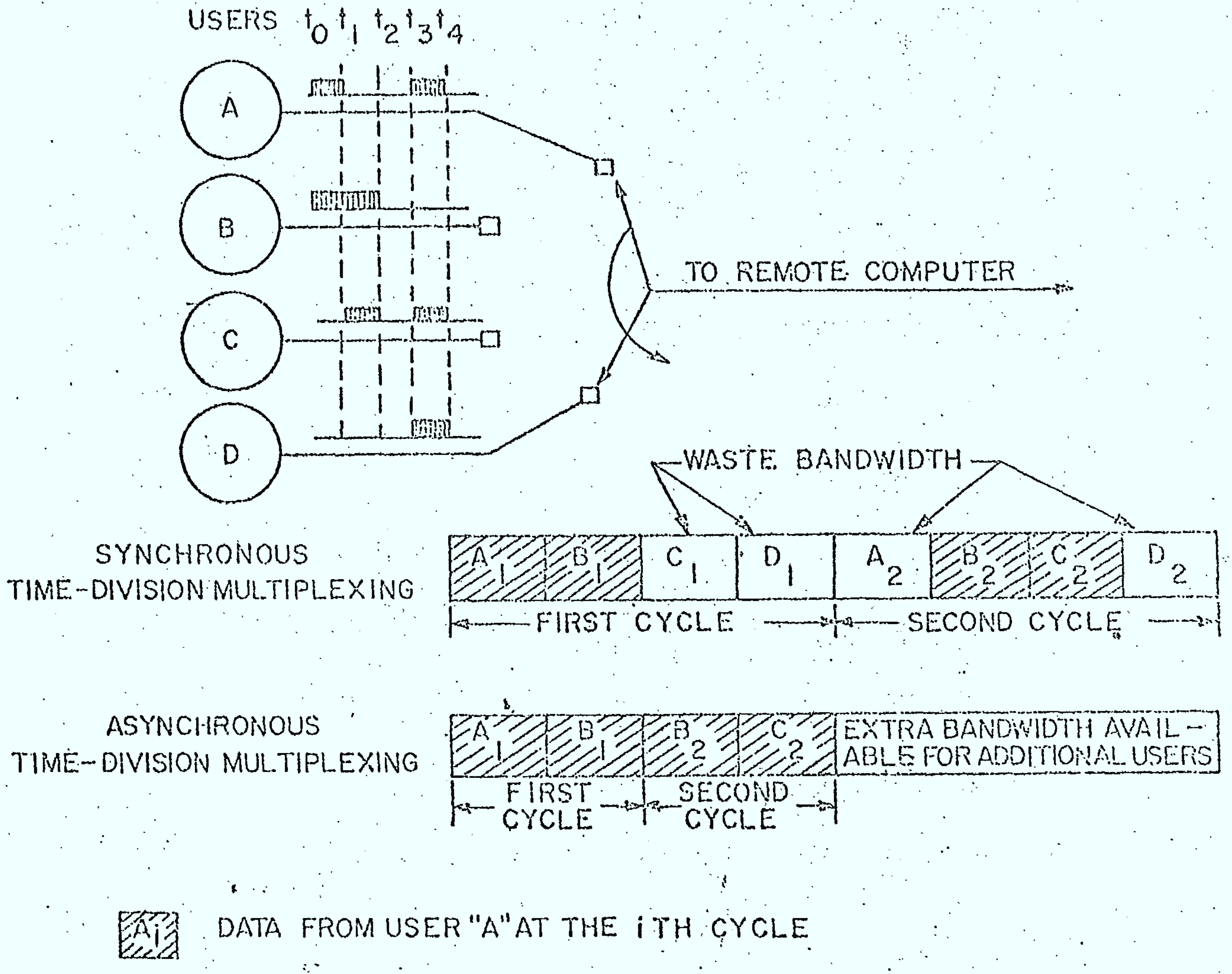
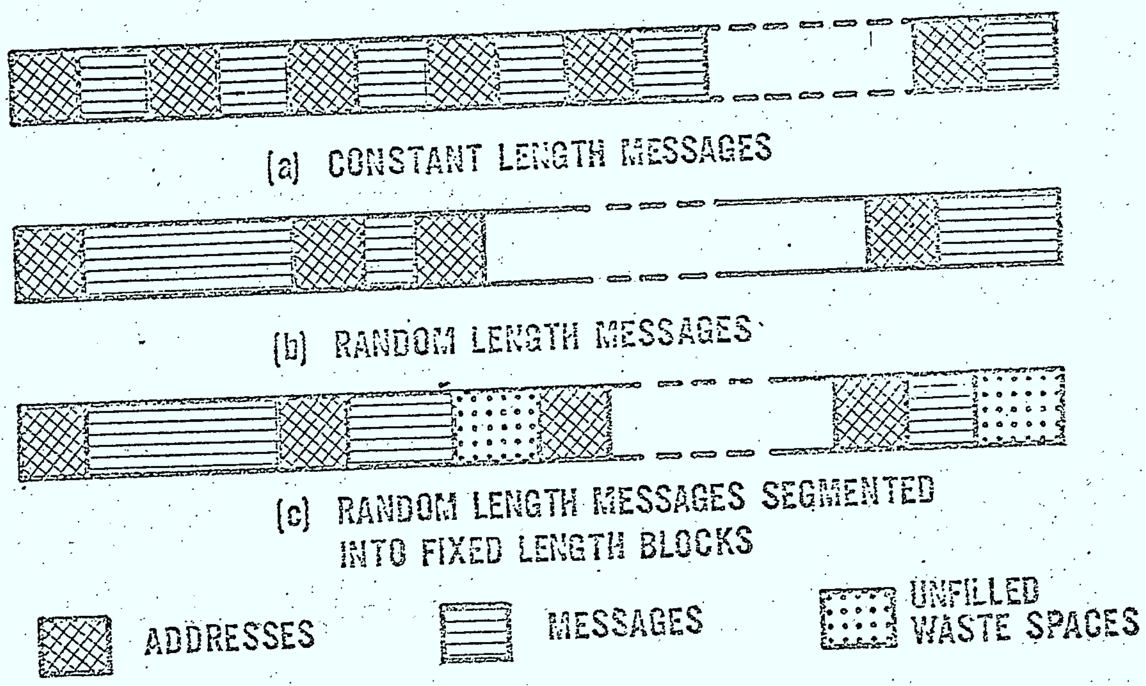

USERS
t_0 t_1 t_2 t_3 t_4
A
B
C
D
TO REMOTE COMPUTER
WASTE BANDWIDTH
SYNCHRONOUS TIME-DIVISION MULTIPLEXING
A_1 B_1 C_1 D_1 A_2 B_2 C_2 D_2
FIRST CYCLE
SECOND CYCLE
ASYNCHRONOUS TIME-DIVISION MULTIPLEXING
A_1 B_1 B_2 C_2
EXTRA BANDWIDTH AVAILABLE FOR ADDITIONAL USERS
FIRST CYCLE
SECOND CYCLE
A_i DATA FROM USER "A" AT THE i TH CYCLE

FIG. I TIME-DIVISION MULTIPLEXING


(a) CONSTANT LENGTH MESSAGES

(b) RANDOM LENGTH MESSAGES

(c) RANDOM LENGTH MESSAGES SEGMENTED INTO FIXED LENGTH BLOCKS

FIG. 2 THREE TYPES OF DATA STRUCTURES

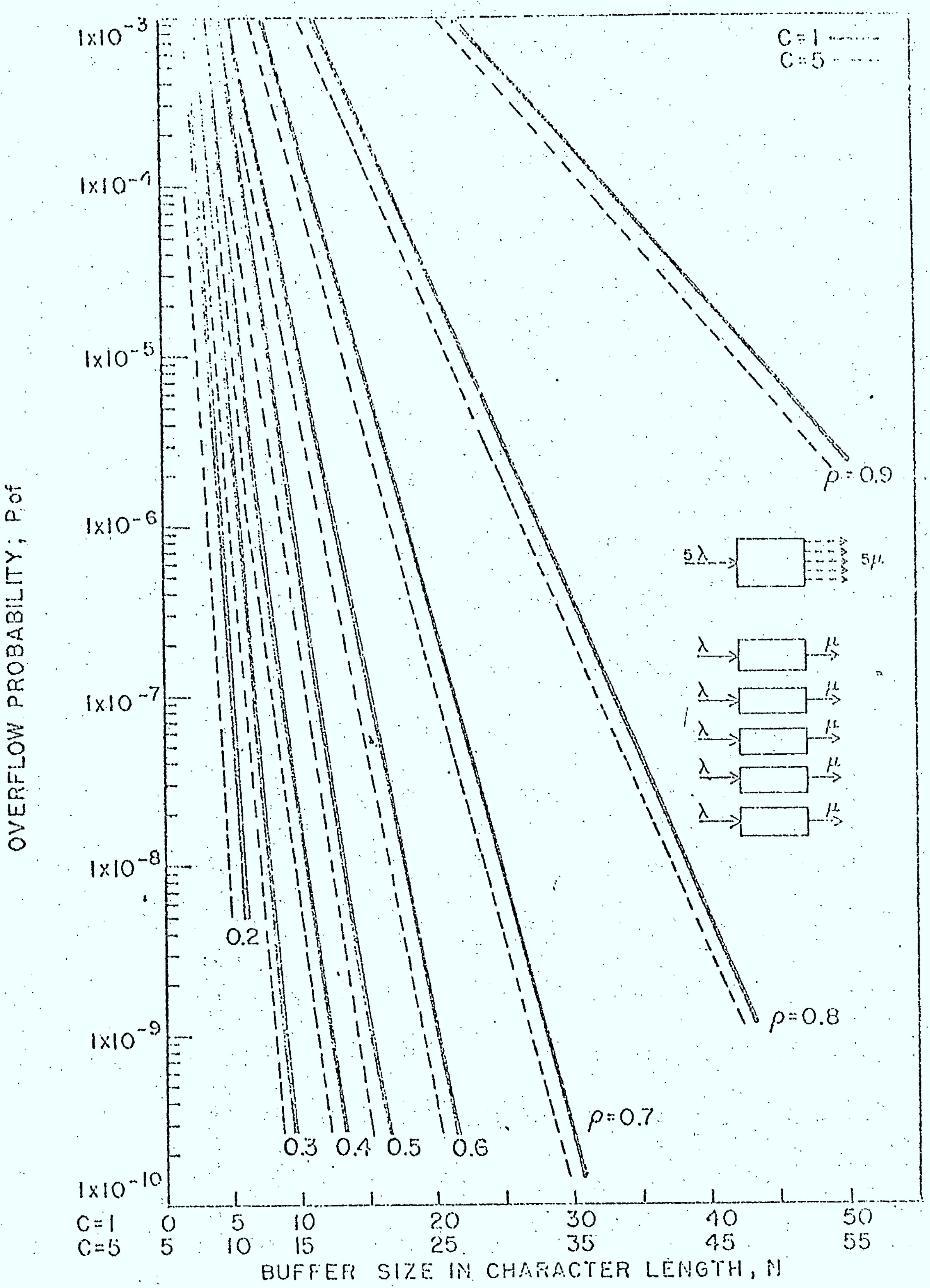


FIG. 3 OVERFLOW PROBABILITY VS BUFFER SIZE

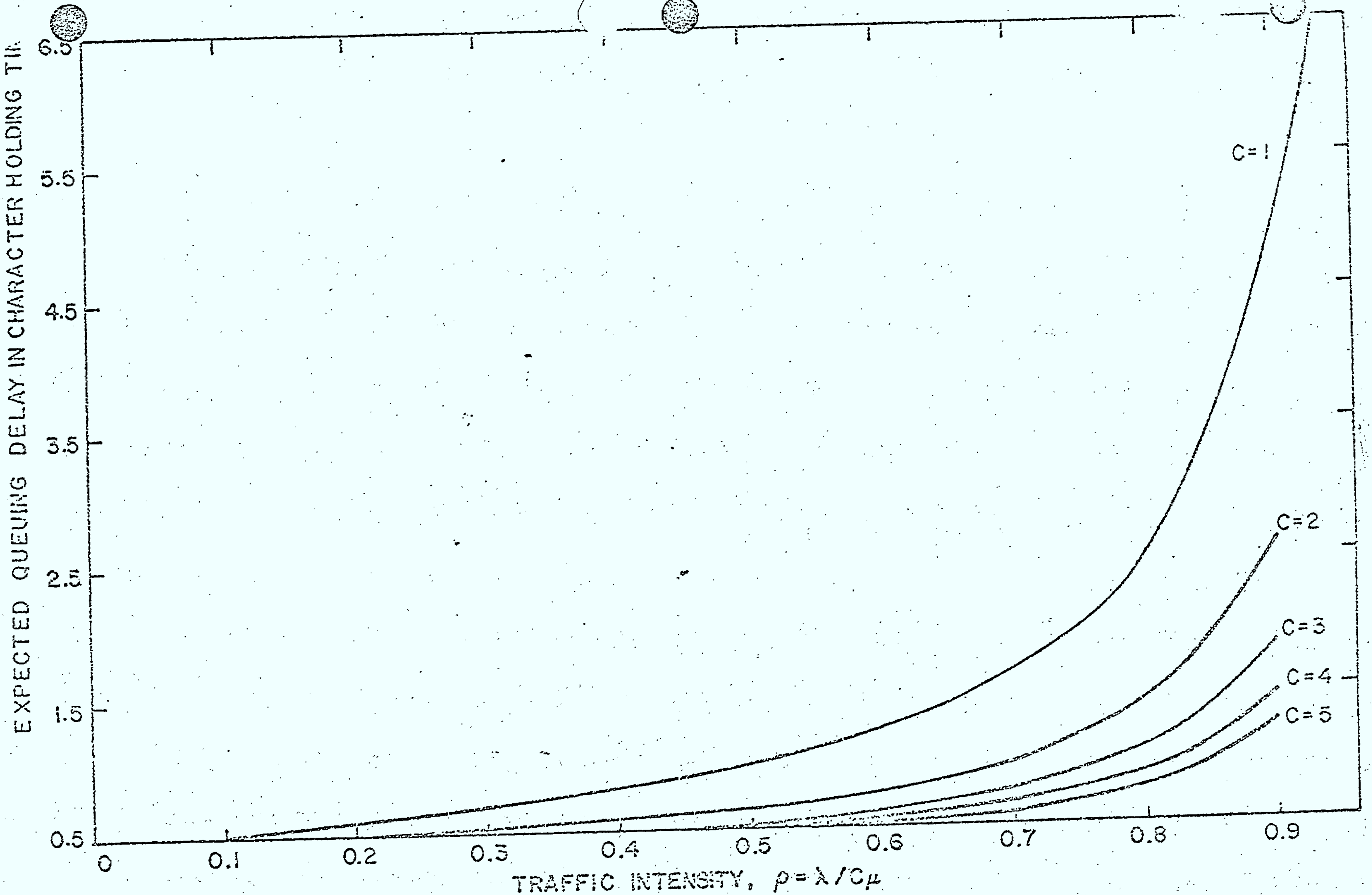


FIG. 4 EXPECTED QUEUING DELAY VS TRAFFIC INTENSITY
(AT OVERFLOW PROBABILITY = 10^{-6})

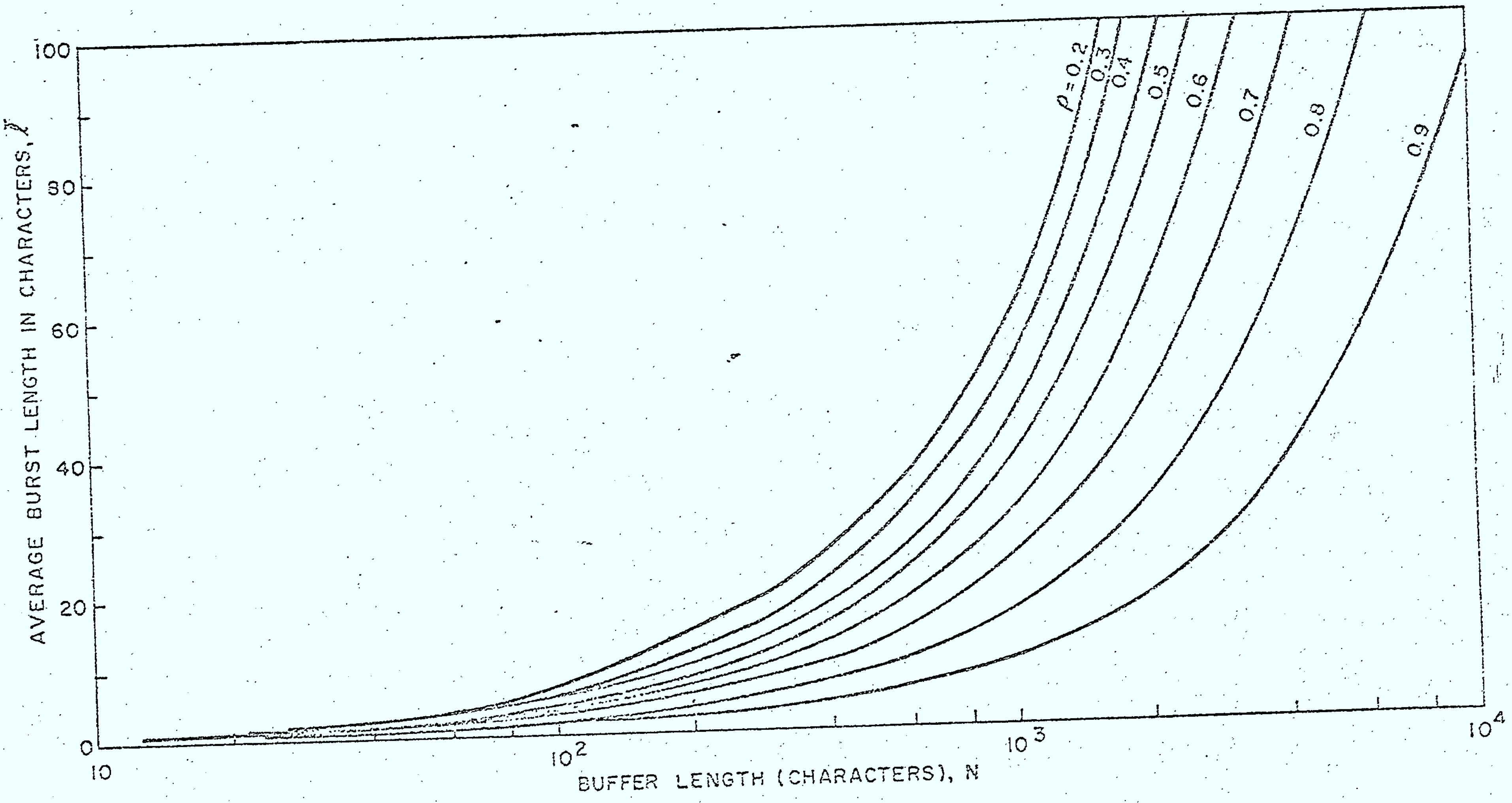


FIG. 5 BUFFER LENGTH VS AVERAGE BURST LENGTH, $P_{OF} = 10^{-6}$

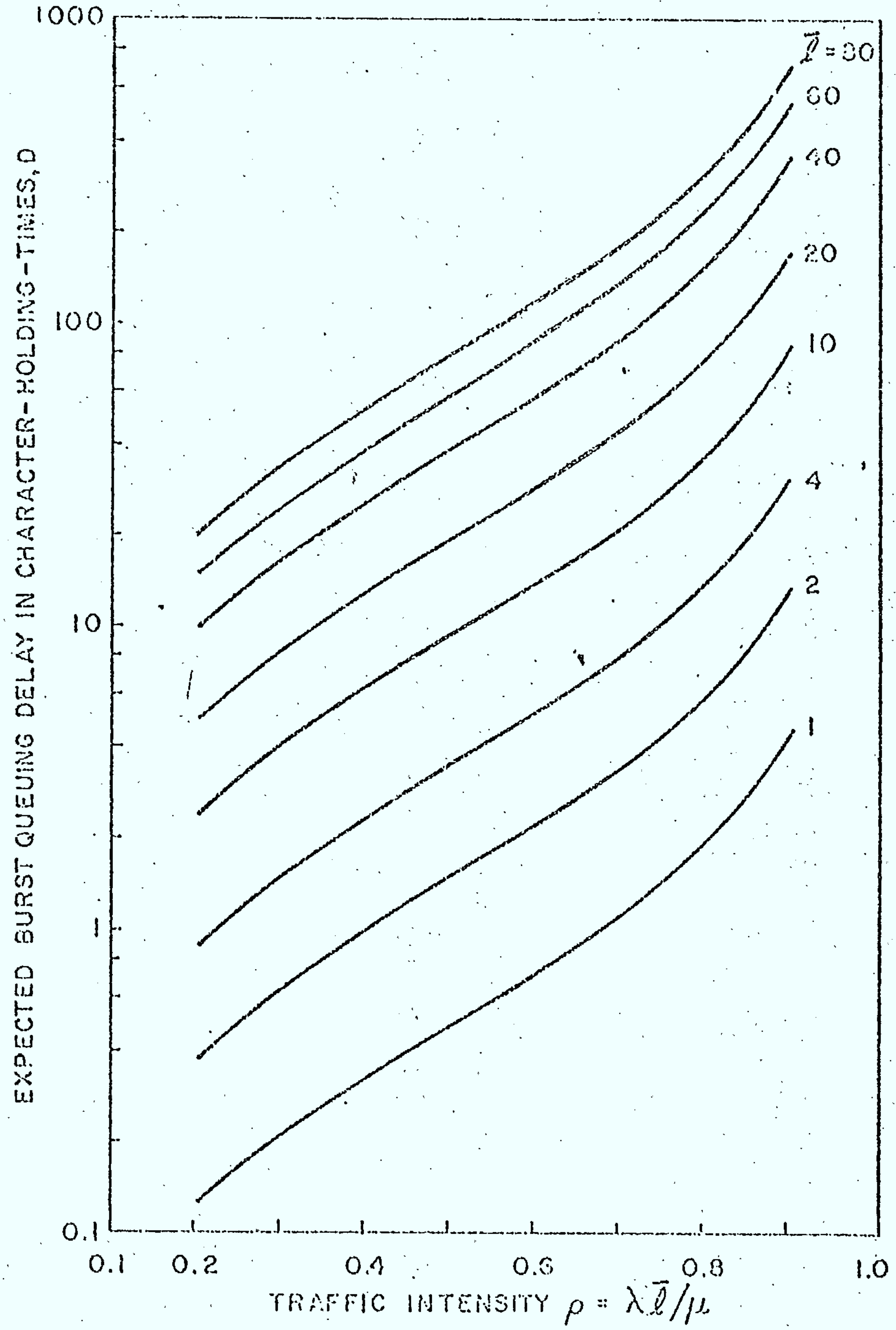


FIG. 6 TRAFFIC INTENSITY VS EXPECTED BURST QUEUING DELAY

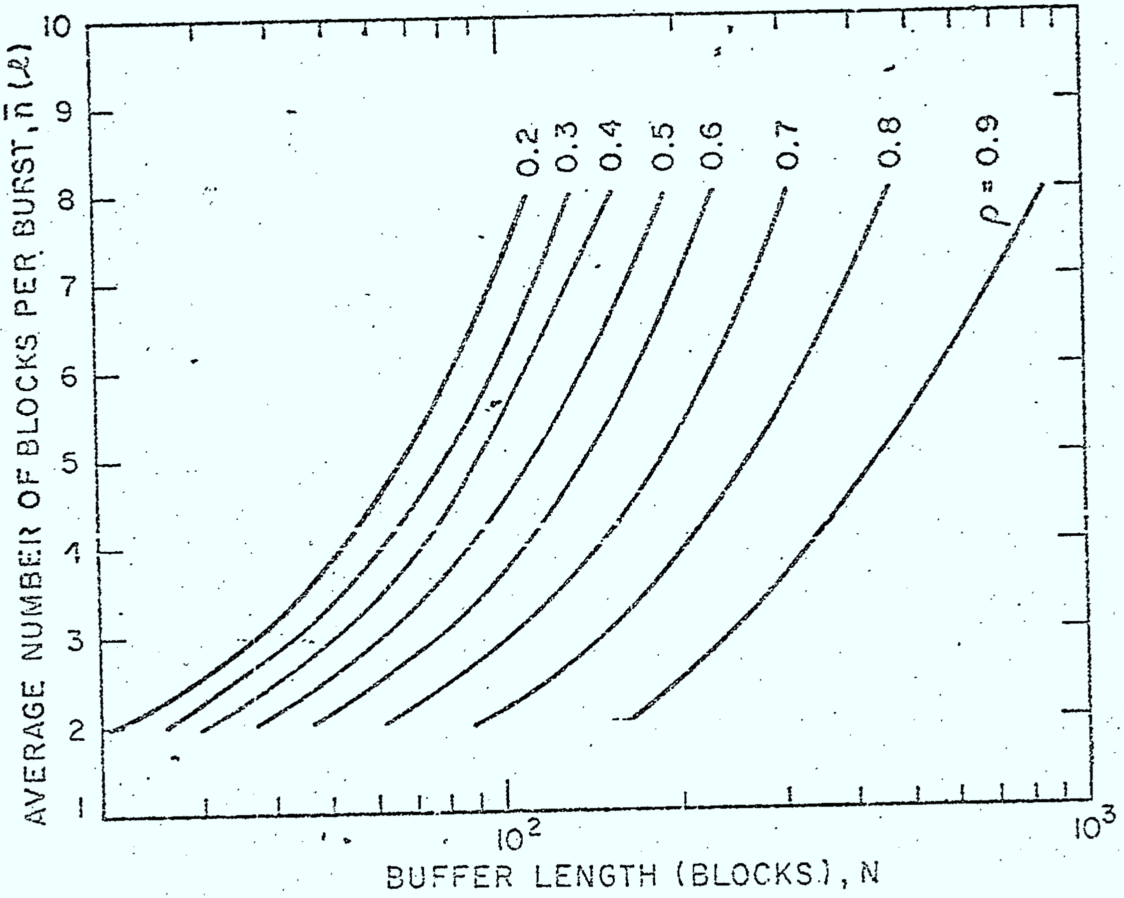


FIG. 7 BUFFER LENGTH VS AVERAGE NUMBER OF BLOCKS PER BURST (AT OVERFLOW PROBABILITIES = 10^{-6})

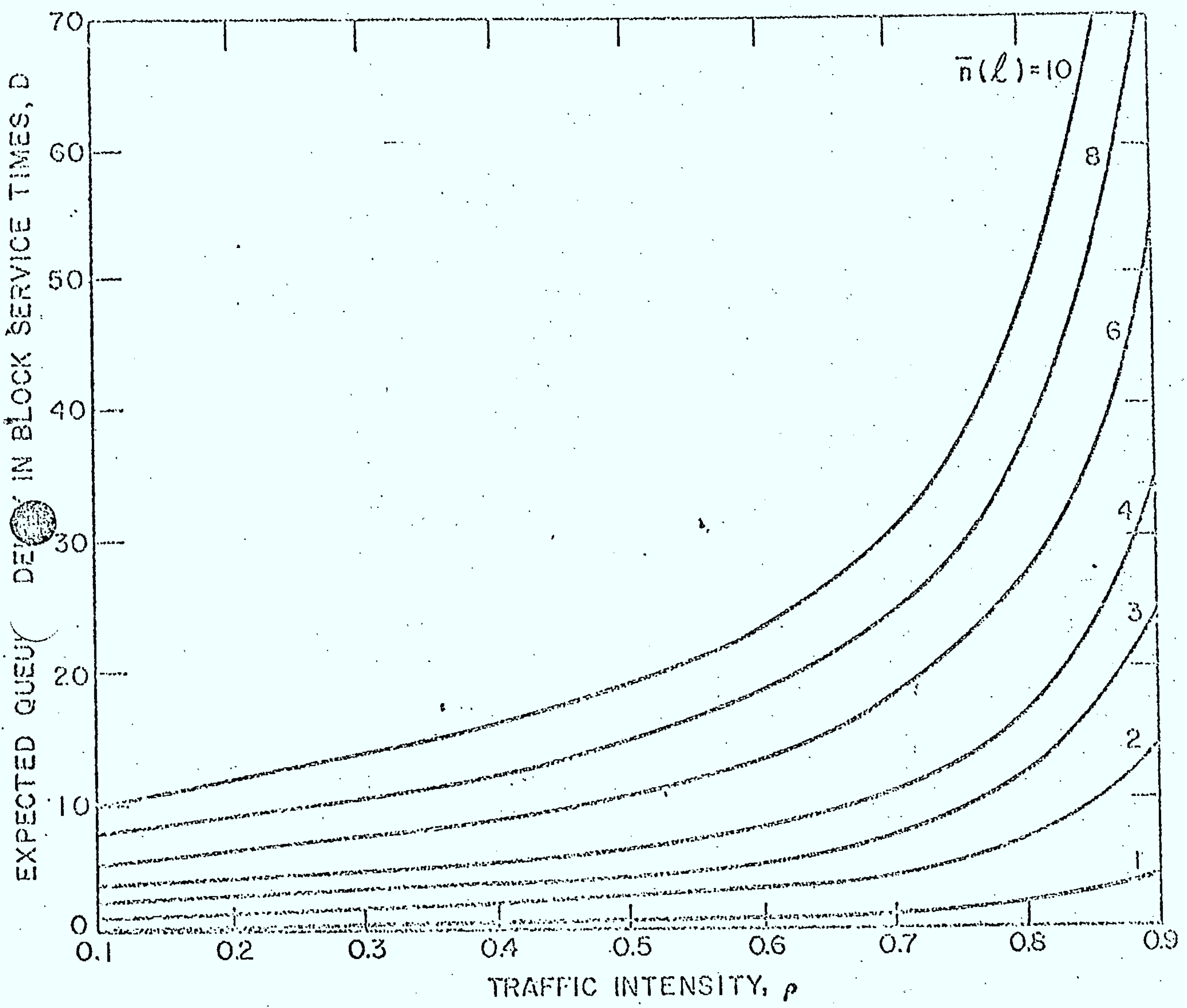


FIG. 8 EXPECTED QUEUING DELAY VS TRAFFIC INTENSITY
(AT OVERFLOW PROBABILITIES = 10^{-6})

To appear in *IEEE Transactions in Communication Technology*, June 1972

DEMULTIPLEXING CONSIDERATIONS FOR STATISTICAL MULTIPLEXORS*

Wesley W. Chu
Computer Science Department
University of California
Los Angeles, California 90024

Abstract

Demultiplexing serves as an important function for statistical multiplexors. Its purpose is to reassemble the received message and distribute it to the appropriate destination. An important cost consideration for this function is the size of the buffer necessary to meet a specified overflow service requirement. The demultiplexing buffer can be modeled as a finite waiting room queuing model with batch Poisson arrivals and multiple distinct constant servers. Stimulation is used to study buffer behavior for traffic arriving at the buffer according to the uniform, linear, step, and geometric destination functions. The relationships among buffer overflow probability, buffer size, traffic intensity, average message length, and message destination are presented in graphs to provide a guide in the design of demultiplexing buffers. Simulation results reveal that buffer input messages which have short average message lengths and uniform traffic destinations yield the best buffer behavior. Thus, in planning the CPU scheduling algorithm and in selecting the demultiplexing output rates, the designing of a computer communications system that uses the statistical multiplexing technique should also consider the output statistics needed to achieve optimal demultiplexing performance.

*This research was supported by the U.S. Office of Naval Research, Mathematical and Information Sciences Division, Contract No. N00014-69-A-0200-4027, NR 048-129 and the Advanced Research Projects Agency of the Department of Defense Contract No. DAHC 15-60-C-0285.

I. Introduction

To increase information processing capability and to share computer resources, remotely located computers and/or terminals are connected together by communication links. Many such computer communication systems as time sharing systems and distributed computer systems are already in existence and in operation. In many cases, the communication cost of such systems is a significant portion of the total operating cost. To increase channel utilization and reduce communication cost, asynchronous time division multiplexing has been proposed for data communications.[1] The design considerations of such a multiplexing (statistical multiplexing) system have been reported in recent literature.[2-6] Here, we study design considerations of an asynchronous time division demultiplexing system, which is an integral part of the design of a statistical multiplexor. In a multiplexing system, outputs from the system go to a single destination for example to a computer while outputs from a demultiplexing system go to many destinations, such as to different.users and/or computers as shown in Figure 1.

The demultiplexor may consist of a buffer, a buffer control unit and a switching circuit. Input messages to the buffer consists of strings of characters (bursts). The switching circuit distributes the output from the buffer to the appropriate destinations according to the designation in the message address. Thus demultiplexor performance is strongly influenced by buffer behavior which depends on buffer input traffic and its distination characteristics. A queuing model with a finite waiting room, batch Poisson arrivals, and multiple distinct constant outputs (Figure 2) is used to study buffer behavior. The complexity of the demultiplexing buffer makes exact mathematical analysis of such a model very difficult. Therefore, computer simulation has been used to study the relationships among message destination function, average traffic level, message burst length, overflow probability (the fraction of the total number of messages rejected by the buffer), and buffer size. These relationships, in graphs, are important in designing demultiplexing systems and store-and-forward computer networks.

II. Analysis of Demultiplexing Buffer

Input messages to the buffer can be represented by three parameters: message arrivals, message lengths, and message destinations. These parameters are intimately related to buffer behavior. In order to describe the input messages arriving at the buffer, three random number generators are used: α which corresponds to message inter-arrival time, β which corresponds to the number of characters in the message, and γ which corresponds to the destination of the message. In the simulation model, the message inter-arrival times α are assumed to be exponentially distributed,* and the message lengths β are assumed to be geometrically distributed. The destination distribution is transformed from the destination function as discussed in the following:

*In the distributed computer network shown in Figure 1b, the traffic input to the demultiplexor is the output from the multiplexor. In some cases the traffic output from the multiplexor can be approximated as Poisson distributed. Should the multiplexor output become very different from Poisson,[7] then the actual message inter-arrival-time statistics should be used in the simulation.

The destination function describes traffic intensities for the set of m destinations; that is,

$$f_d(i) = \rho_i \qquad i = 1,2,\ldots,m \tag{1}$$

where ρ_i is the traffic intensity for the i^{th} destination.

The message destination function for a given set of users depends on their applications and should be derived from measured statistics.

In order to perform random sampling on traffic destination for simulation, we need to transform $f_d(i)$ into a message destination distribution, $f_\gamma(i)$. This transformation can be carried out by normalizing Equation (1) to a probability function; that is,

$$f_\gamma(i) = f_d(i) \Big/ \sum_{j=1}^{m} \rho_j \qquad i = 1,2,\ldots,m \tag{2}$$

Thus,

$$\sum_{i=1}^{m} f_\gamma(i) = 1$$

A set of random numbers, $\{\xi_\alpha, \xi_\beta, \xi_\gamma\}$, corresponding to random variables α, β, and γ, are generated to represent a message arriving at the demultiplexing system. When a message arrives at the buffer, two operations take place: First, the status of the designated facility is interrogated. If the facility is busy and the buffer is not full, the burst enters the buffer and is concatenated with the queue of characters. If the facility is idle, and if the buffer is not full, the first character of the burst is sent to the facility while the remaining characters enter into the buffer and output at each subsequent clock time. Second, the contents of the register which keeps tract of the total length of the buffer is updated. Because of distinct destinations, the total volume of output from the buffer varies with α, β, and γ. The output from the demultiplexing buffer depends on both the number of characters in the buffer and their destinations. The simulation program keeps a record of the number of characters in the buffer at the beginning of each message service interval. When the length of an arriving message exceeds the unoccupied storage space of the buffer, a buffer overflow event has occurred. The frequency of occurrence of such an overflow event gives the estimate of buffer overflow probability, P_{of}. Using the above finite buffer size model for estimating buffer overflow probability requires simulating buffer behavior at various buffer sizes. As a result, the computation time required for such a model, especially for estimating small overflow probabilities, could be prohibitive. Therefore, we introduce an infinite buffer size model, as shown in the following flow chart (Figure 3), to estimate buffer overflow probabilities. In this case, we say that an overflow event has occurred if the buffer queue length from simulation exceeds the fictitious buffer size. Thus, this provides us with a way to estimate buffer overflow events for various buffer sizes via buffer queue length statistics which can be obtained from a single simulation pass. This represents a significant reduction in computation time. In the case of small overflow probabilities (e.g. $P_{of} < 10^{-4}$), this model provides a good approximation of the finite buffer size model.

Traffic intensity is one of the most important parameters that describes the traffic congestion of the demultiplexing system. Since the output messages from the buffer are sent to various destinations, the traffic intensity of each destination would be different and would equal

$$\rho_i = \frac{\lambda_i \bar{\ell}_i}{\mu_i} \qquad (3)$$

where

λ_i = message arrival rate for the i^{th} destinations

$\bar{\ell}_i$ = average message length for the i^{th} destination

μ_i = transmission rate for the i^{th} destination

The average traffic level, $\bar{\rho}$, for the demultiplexing system is the average of the traffic intensities of the m destinations; that is,

$$\bar{\rho} = \frac{1}{m} \sum_{i=1}^{m} \rho_i \qquad (4)$$

To study the effect of destination distributions on buffer behavior, five types of destination functions are used in the simulation model as shown in Figure 4. Further, to isolate the effect of message length of various destinations on buffer behavior, we assumed the average message length for various destinations are equal; that is, $\bar{\ell} = \bar{\ell}_i$. The relationships among P_{of}, buffer sizes, and destination distributions for selected $\bar{\ell}$'s and $\bar{\rho}$'s are obtained from simulation and are protrayed in Figures 5 and 6.

The expected queuing delay due to buffering during demultiplexing is another important parameter in considering the design of statistical multiplexors. Since most systems allow a very low overflow probability, the expected waiting time due to buffering can be approximated by that of an infinite waiting room queuing model with Poisson arrivals and geometric service time. The expected waiting time W_i for sending messages to the i^{th} destination is

$$W_i = \frac{\lambda_i E(X_i^{\,2})}{2(1-\rho_i)} = \frac{\rho_i\ (2\bar{\ell}_i - 1)}{2(1-\rho_i)} \quad \text{character-service-times} \tag{5}$$

where $E(X_i^{\,2})$ is the second moment of the message length X_i for the i^{th} destination.

Since the traffic intensity and the average message length for each destination are different, the expected queuing delay due to demultiplexing for each destination is also different and should be computed from its associated traffic intensity and average message length.

III. Discussion of Results

The relationships between buffer size and buffer overflow probability for selected average traffic levels, average burst lengths, and traffic destination functions are shown in Figures 5 and 6. For a given average message length and desired level of overflow probability, the required buffer size increases as the average traffic level increases. Further, the required buffer size varies drastically with message destination functions. Comparing the five types of traffic destination functions used in our simulation, for a given average traffic level the uniform destination function required the smallest buffer size, and the step 2 destination function required the largest buffer size. This is because both buffer size and queuing delay increase exponentially with ρ_i. Thus for a given average traffic level, different types of destination functions yield different buffer behavior. For example, if one of the destinations is heavily loaded, the average traffic level of the system could be very low, but an extremely large buffer is needed. Further, the expected queuing delay for sending messages to that heavily loaded destination would be very long. We notice that the effect of destination function on buffer behavior increases as average traffic level increases.

Results of buffer behavior simulation shed light on the relationships between demultiplexor traffic inputs and demultiplexing system performance. The results show that to minimize the size of the demultiplexing buffer and the queuing delay due to buffering, we should schedule the inputs to the demultiplexor buffer approximately equally. This can be achieved by controlling the volume of input traffic to the buffer for various destinations and selecting the output transmission rate of the buffer, or both. In a time-sharing system, input traffic to the demultiplexor buffer is governed by the computer scheduling algorithm, such as the process scheduling and the size of CPU quantum time. If we know the relationship* between the CPU quantum time and the volume of the CPU outputs, then a variable quantum time scheduling algorithm would be effective to schedule an equal amount of output traffic to various destinations. The process scheduling as well as the buffer output rate can also be changed to adjust the traffic intensity. To achieve a uniform traffic destination distribution, we should assign high transmission rate lines to those destinations that have high volumes of traffic.

In our simulation model, ten buffer output destinations are used. Since the input traffic to the buffer is random, the ten output destinations can also be viewed as ten groups of outputs provided the outputs in each group have approximately the same value of traffic intensity.** When a demultiplexing buffer has more than ten outputs, we could group them into ten groups according to their traffic intensities, and then select the appropriate graphs to estimate its buffer behavior.

* This relationship depends on the computer system and program behavior in question, and could be obtained via measurement or simulation.

** Simulation results verified this conjecture for two or three outputs in a group.

IV. Example

Consider the design of a demultiplexing system for a time-sharing system that employs the statistical multiplexing technique shown in Figure 1a. Input traffic to the demultiplexor is generated from the computer and is approximated as Poisson arrivals. Message length is approximated as geometrically distributed with a mean of 20 characters. Further, message destinations are partitioned into ten groups, and the destination function is a step-like function as shown in Figure 4c. The average traffic level is about 0.8. From Figure 6b, to achieve an overflow probability of 10^{-5}, the required buffer size is 4,850 characters. From Equation (5), the expected queuing delay due to buffering for the higher traffic destination group is 271 character-service-times and the lower traffic destination group is 35 character-service-times. Therefore, the average-service-time is (271+35)/2 = 153 characters-service-times. Now suppose we change the CPU scheduling to a variable quantum time scheduling algorithm. By assigning larger quantum times to those lower traffic intensity destinations, the destination function changes from a step-like function to an approximately uniform function. From Figure 6b, the required buffer size to achieve $P_{of} = 10^{-5}$ is 2,200 characters, and from Equation (5), the queuing delay is 78 character-service-times. We note that both the required buffer size and the queuing delay due to buffering have been improved after changing the CPU scheduling algorithm. In a practical system design we should further consider the inquiry-response time constraints, the overhead cost of various scheduling algorithms, and the CPU throughput.

The simulation was performed by using the GPSS program on the IBM 360/91 at UCLA. The computing time required for the simulation depends largely on the sample size and the number of destinations used. For a given α,β, and a destination distribution, the computing time required for an experiment (which represents a curve on the graph) of 10^5 samples and ten destinations is about five minutes. As is common in many stochastic experiments, the results are sensitive to sample size. A larger sample size yields a more accurate estimation. For a compromise between accuracy and the computing time required, a sample size of 10^5 was selected for each experiment.

V. Conclusions

The statistical demultiplexor consists of a buffer, a buffer control unit and a switching circuit. Due to the complex output structure of demultiplexing buffer, its behavior is studied via computer simulation. Simulation results reveal that the traffic destination function has a drastic effect on the behavior of the demultiplexing buffer. Optimal buffer behavior is achieved by the uniform destination function; that is, when equal amounts of traffic are sent to the various destinations. Since the CPU scheduling algorithm in a time-sharing system and the message routing algorithm in a distributed computer system strongly affect the traffic destination function, computer operating systems and/or message routing algorithms greatly influence demultiplexor performance. For example, a variable CPU quantum time scheduling algorithm may be effective in producing a uniform destination function and may yield better demultiplexing performance. Thus the simulation model and the results in this paper should serve as a guide in designing a demultiplexing system and in planning an optimal computer communications system.

Acknowledgment

The author wishes to thank Leo Liang of UCLA for his assistance in the GPSS simulation and E. Fuchs and R.J. Roddy of Bell Telephone Labs, Holmdel, N.J., for their critical comments on this paper.

References

1. W.W. Chu, "A Study of Asynchronous Time Division Multiplexing for Time-Sharing Computer System," *AFIPS Conference Proceedings*, Vol. 35, pp. 669-678, 1969.

2. W.W. Chu, "Design Considerations of Statistical Multiplexors," *Proceedings of ACM Symposium on Problems in the Optimization of Data Communication Systems*, Pine Mountain, Georgia, pp. 36-60, 1969.

3. W.W. Chu, "Selection of Optimal Transmission Rate for Statistical Multiplexors," *Proceedings of 1970 International Conference on Communications*, San Francisco, California, pp. 28-22 to pp. 28-25, 1970.

4. T.G. Gordon, et al., "Design of Performance of a Statistical Multiplexor," *Proceedings of 1970 International Conference on Communications*, San Francisco, pp. 28-7 to 28-21, 1970.

5. H. Rudin, Jr., "Performance of a Simple Multiplexor-Concentrator for Data Communication," Vol. Com-19, No. 2, April 1971, pp. 178-187.

6. J.H. Chang, "Comparison of Synchronous and Asynchronous Time Division Multiplexing Techniques," *Proceedings of 1970 International Conference on Communications*, San Francisco, California, pp. 16-10 to 16-17, 1970.

7. C.D. Pack, "The Effect of Multiplexing on a Computer Communication System," Submitted to the *Communication of ACM* for publication.

FIGURE CAPTIONS

1. Statistical Multiplexing Systems

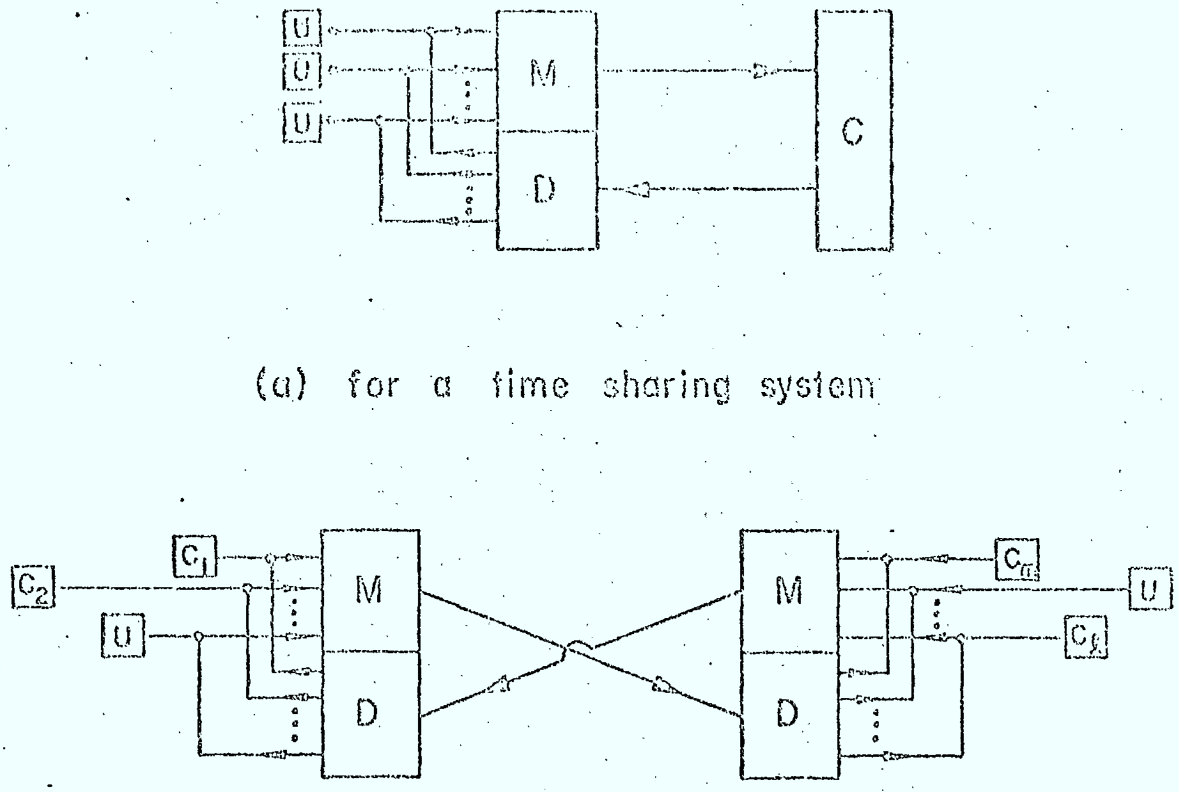
2. A Model for Demultiplexing Buffer

3. A Logic Flow Chart for the Buffer Simulation Model

4. Various Types of Destination Functions for $\ddot{\rho} = 0.8$

5. Buffer Overflow Probabilities vs Buffer Size for Average Traffic Level $\ddot{\rho} = 0.6$, 5a) $\ddot{\ell} = 10$ characters, 5b) $\ddot{\ell} = 20$ characters, 5c) $\ddot{\ell} = 40$ characters

6. Buffer Overflow Probabilities vs Buffer Size for Average Traffic Level $\ddot{\rho} = 0.8$, 6a) $\ddot{\ell} = 10$ characters, 6b) $\ddot{\ell} = 20$ characters, 6c) $\ddot{\ell} = 40$ characters



(a) for a time sharing system

(b) for a distributed computer system

U : user terminal

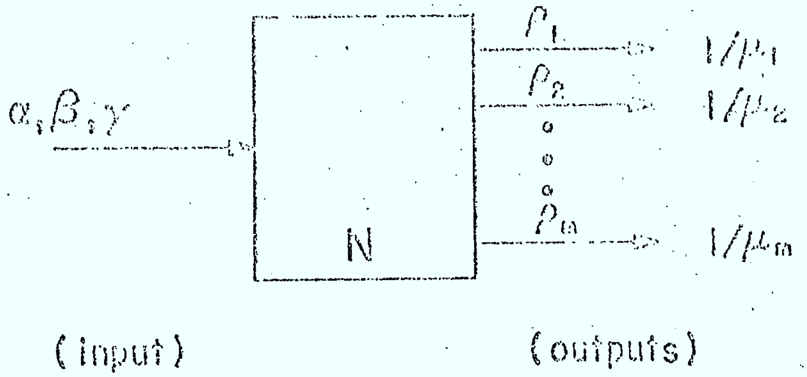
D : statistical demultiplexor

M : statistical multiplexor

C : computer

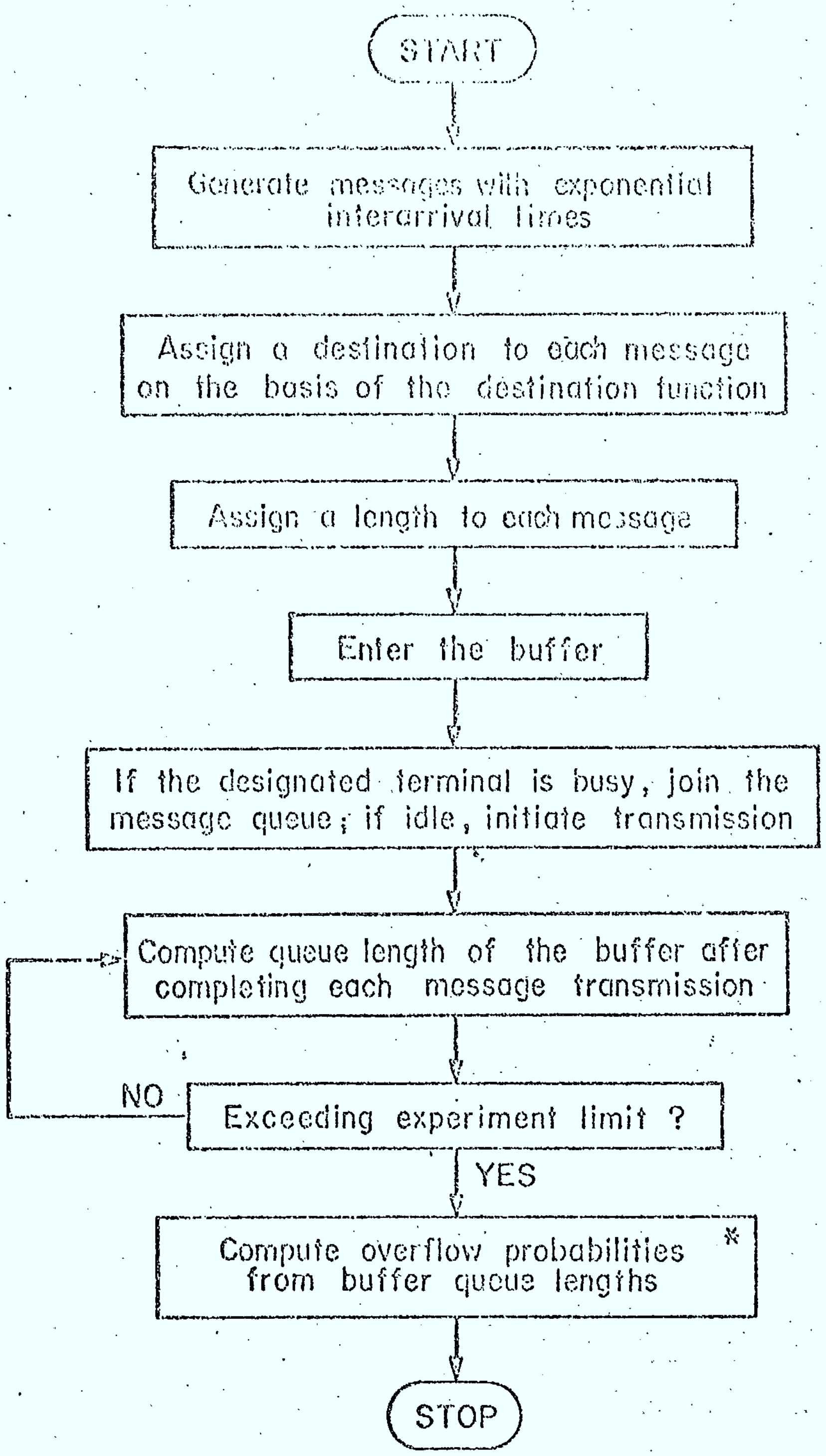
C_i : i^{th} computer

Statistical Multiplexing Systems



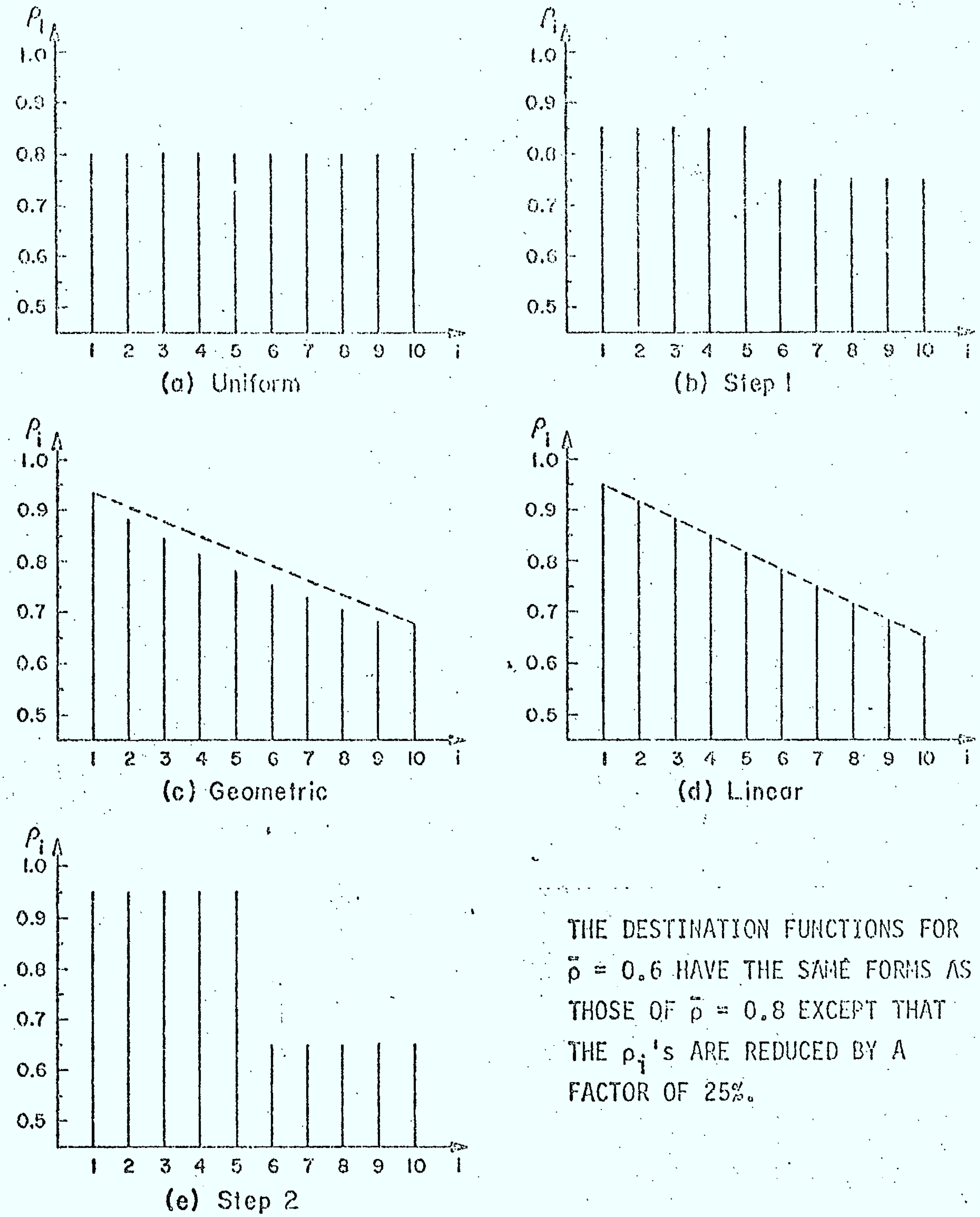
α	message inter-arrival time
β	message length
γ	destination function
μ_i	transmission rate for the i^th destination
ρ_i	traffic intensity for the i^th destination
N	buffer size
m	total number of message destinations

A Model for Demultiplexing Buffer


START
Generate messages with exponential interarrival times
Assign a destination to each message on the basis of the destination function
Assign a length to each message
Enter the buffer
If the designated terminal is busy, join the message queue; if idle, initiate transmission
Compute queue length of the buffer after completing each message transmission
NO
Exceeding experiment limit ?
YES
Compute overflow probabilities from buffer queue lengths *
STOP

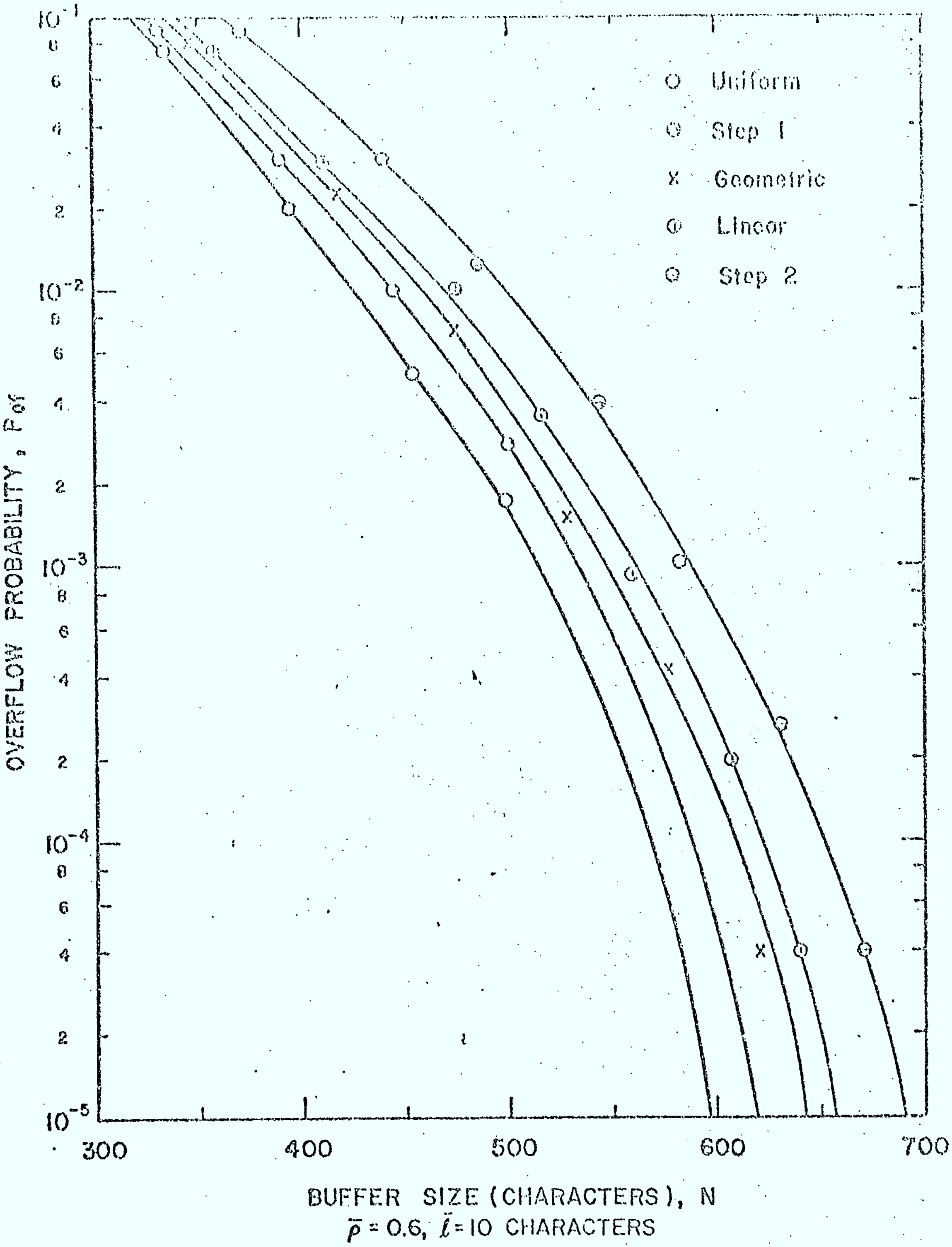
* The buffer overflow probabilities were computed from the buffer queue lengths obtained from the GPSS simulation.

A Logic Flow Chart for the Buffer Simulation Model

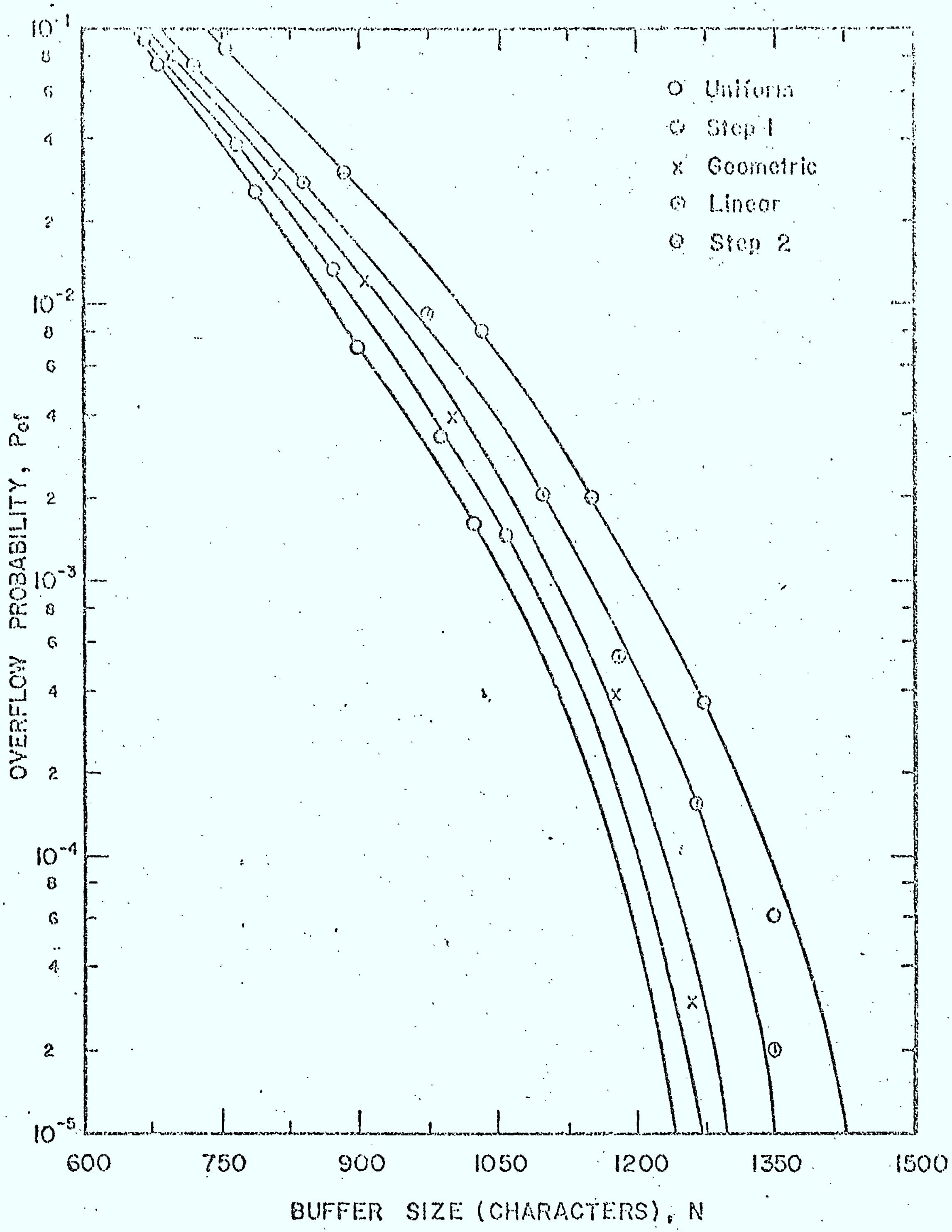


THE DESTINATION FUNCTIONS FOR $\bar{\bar{\rho}} = 0.6$ HAVE THE SAME FORMS AS THOSE OF $\bar{\bar{\rho}} = 0.8$ EXCEPT THAT THE ρ_i's ARE REDUCED BY A FACTOR OF 25%.

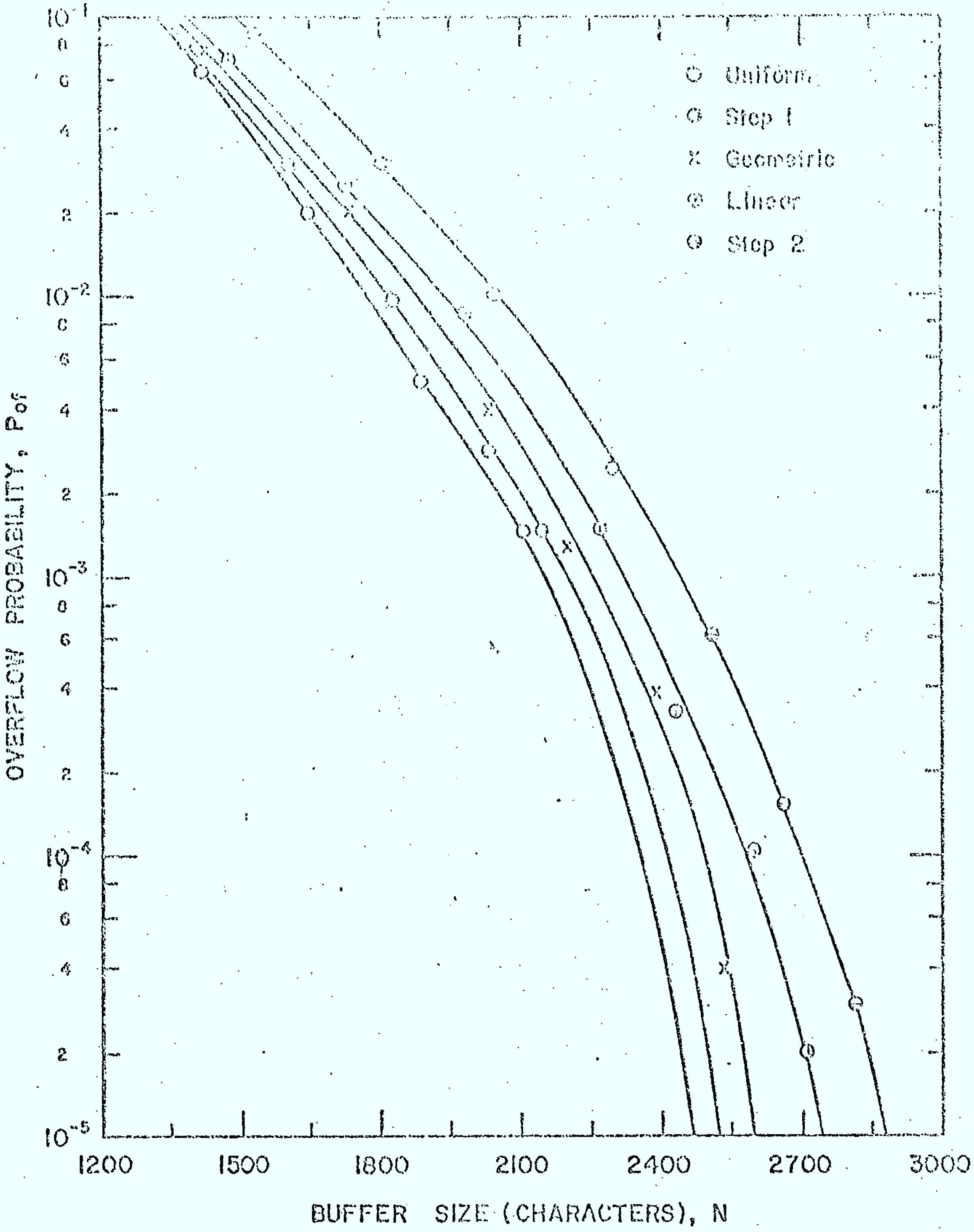
Various Types of Destination Functions

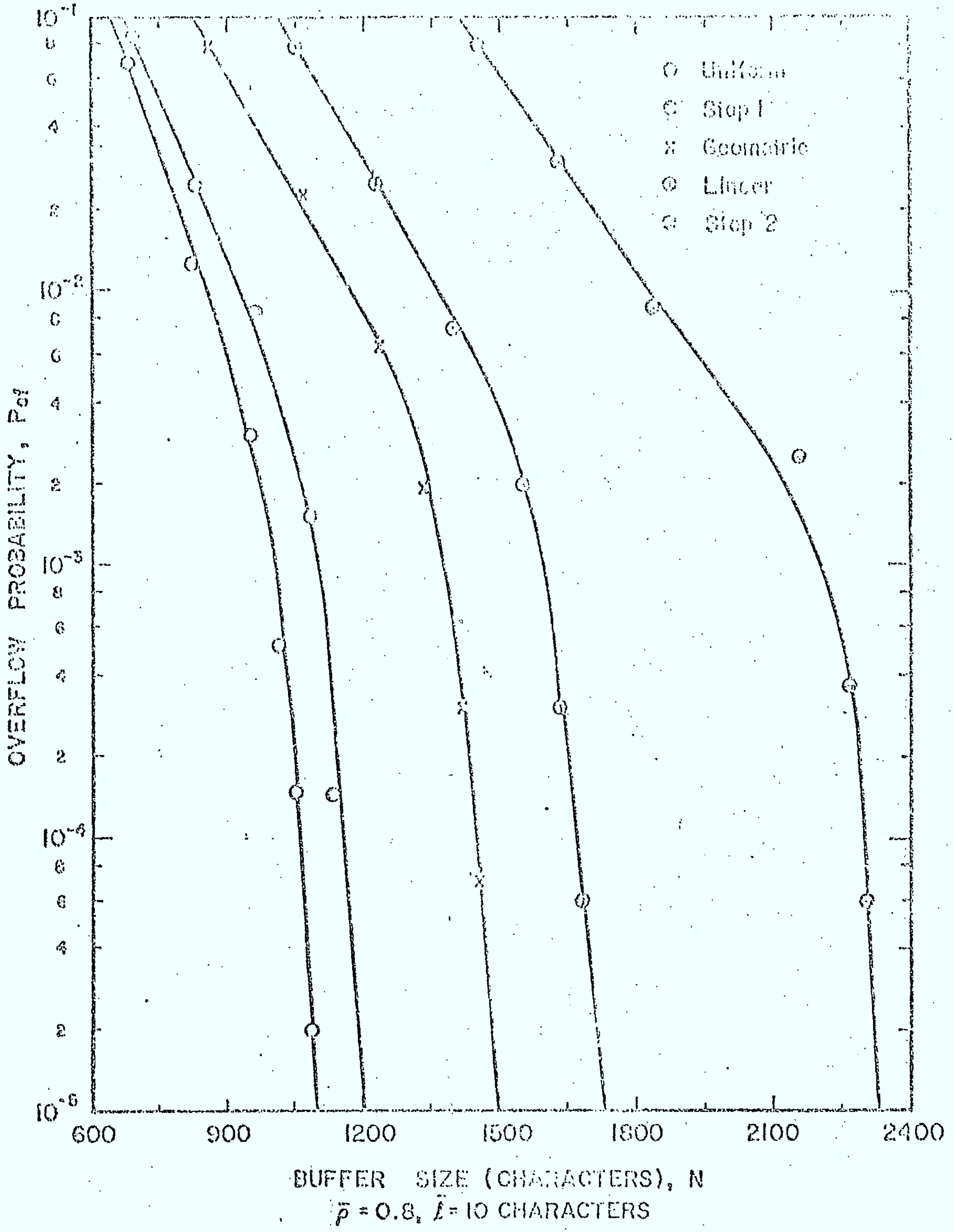


10⁻¹
8
6
4
2
10⁻²
8
6
4
2
10⁻³
8
6
4
2
10⁻⁴
8
6
4
2
10⁻⁵
300
400
500
600
700
OVERFLOW PROBABILITY, P_of
BUFFER SIZE (CHARACTERS), N
ρ̄ = 0.6, l̄ = 10 CHARACTERS
○ Uniform
⊙ Step 1
X Geometric
⊕ Linear
⊙ Step 2

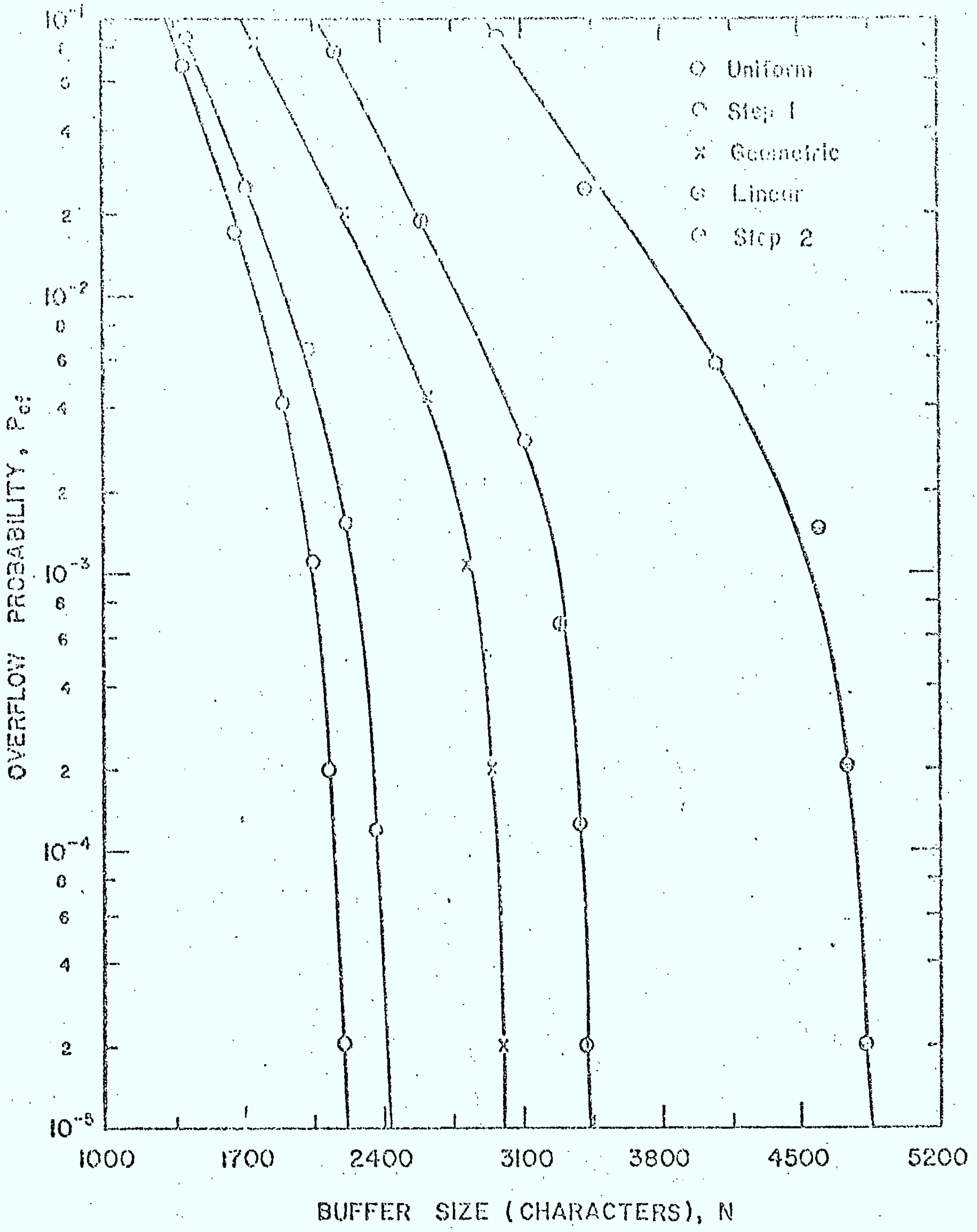


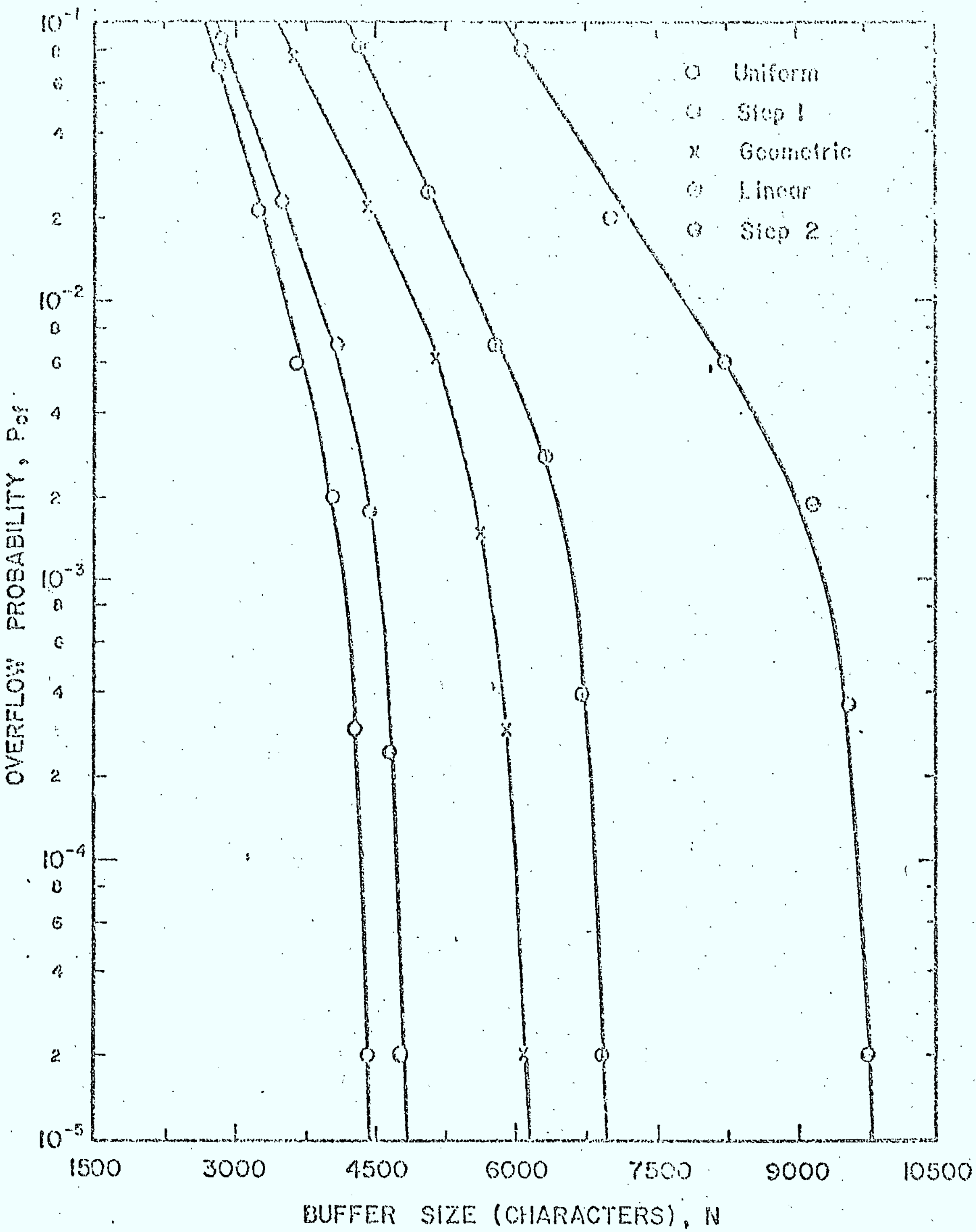
10^{-1}
8
6
4
2
10^{-2}
8
6
4
2
10^{-3}
8
6
4
2
10^{-4}
8
6
4
2
10^{-5}
OVERFLOW PROBABILITY, P_{of}
600
750
900
1050
1200
1350
1500
BUFFER SIZE (CHARACTERS), N
○ Uniform
⊕ Step 1
x Geometric
⊙ Linear
⊖ Step 2


10^{-1}
8
6
4
2
10^{-2}
8
6
4
2
10^{-3}
8
6
4
2
10^{-4}
8
6
4
2
10^{-5}
OVERFLOW PROBABILITY, P_{of}
1200
1500
1800
2100
2400
2700
3000
BUFFER SIZE (CHARACTERS), N
○ Uniform
⊖ Step 1
× Geometric
⊗ Linear
⊖ Step 2



○ Uniform
◌ Step 1
× Geometric
⊙ Linear
◌ Step 2
OVERFLOW PROBABILITY, P_{of}
10^{-1}
8
6
4
2
10^{-2}
8
6
4
2
10^{-3}
8
6
4
2
10^{-4}
8
6
4
2
10^{-5}
600
900
1200
1500
1800
2100
2400
BUFFER SIZE (CHARACTERS), N
$\bar{\rho}$ = 0.8, $\bar{l}$ = 10 CHARACTERS


○ Uniform
○ Step 1
× Geometric
⊖ Linear
○ Step 2
OVERFLOW PROBABILITY, P_c
10^{-1}
8
6
4
2
10^{-2}
8
6
4
2
10^{-3}
8
6
4
2
10^{-4}
8
6
4
2
10^{-5}
1000
1700
2400
3100
3800
4500
5200
BUFFER SIZE (CHARACTERS), N


10^{-1}
8
6
4
2
10^{-2}
8
6
4
2
10^{-3}
8
6
4
2
10^{-4}
8
6
4
2
10^{-5}
OVERFLOW PROBABILITY, P_{of}
○ Uniform
○ Step 1
x Geometric
○ Linear
○ Step 2
1500
3000
4500
6000
7500
9000
10500
BUFFER SIZE (CHARACTERS), N

Analytical Model of Half-Duplex Interconnections of Computers

JACK S. SYKES

REFERENCE: Sykes, J. S : ANALYTICAL MODEL OF HALF-DUPLEX INTERCONNECTIONS OF COMPUTERS, Bell Telephone Laboratories, Inc., Holmdel, N. J. 07733. Rec'd 2/16/68; revised 12/30/68. Paper 68TP418-COM, approved by the IEEE Data Communication Systems Committee for publication after sponsored presentation at the 1968 IEEE International Conference on Communications, Philadelphia, Pa. IEEE TRANS. ON COMMUNICATION TECHNOLOGY, 17-2, April 1969, pp. 235–238.

ABSTRACT: Computer interconnections allowing half-duplex data communications have been analyzed using a single-server dual-queue model. Storage usage for message queuing at each computer and associated message delays are considered as functions of communications parameters.

Introduction

WITH THE advent of time-sharing computers, many systems are being planned that involve the exchange of information between computers. Two examples of systems in which data communications between computers are a necessity are the following:

1) message switching systems in which messages accepted by one computer can be destined for delivery at a station served by another computer,

2) large-scale inquiry-response systems in which the data base exceeds the capacity of a single computer. When this happens, a local computer occasionally must obtain the requested information from another computer.

In these and similar systems, the computers involved are interconnected by means of data links. The characteristics of the data link chosen for a particular system depend on various factors and parameters: the volume of messages to be transmitted, the delays that can be tolerated, the cost of various types of transmission facilities, etc. For many systems, computers may be interconnected with facilities allowing half-duplex message transmission, i.e., facilities that allow transmission in either direction, but in only one direction at a time.

This paper presents a mathematical queuing model that can be used to represent two computers that communicate in a half-duplex mode. The solution of this queuing model has led to formulas for estimating two related quantities of major importance in a data communication system: 1) the average delay of messages awaiting transmission from one computer to the other, and 2) the average amount of storage used for message queuing at each computer. These formulas are expressed as functions of relevant parameters of data communications systems.

Model Description

Let us assume that in a given system, computers i and j communicate in a half-duplex mode. With sufficient occupancy of the data link, messages to be transmitted to computer j will be delayed in a queue at computer i, and vice versa. In the remainder of this paper, the queue at computer i and the queue at computer j, which are obviously interdependent, will be referred to as queue i and queue j, respectively.

In order to study the factors influencing the delay of messages and the storage used for message queuing, a single-server dual-queue queuing model has been formulated. The single server in the model represents the data link that alternately allows transmission of the messages that accumulate in queue i and in queue j. The service times in the model represent the intervals t_i and t_j required to transmit individual messages. For example, t_i is the transmission time of a message from computer i to computer j. These transmission times are independently determined by the message lengths l_i and l_j and the constant transmission rates τ_i and τ_j, since for each message $t_i = l_i/\tau_i$ and $t_j = l_j/\tau_j$. If $\bar{l}_i$ is the average length of messages transmitted from computer i to computer j, then the average value of t_i is $\bar{t}_i = \bar{l}_i/\tau_i$. Likewise, $\bar{t}_j = \bar{l}_j/\tau_j$.

The assumptions upon which this application of the queuing model has been based are as follows:

1) Poisson queue arrivals, i.e., the probability that X messages join queue i during an interval z has a Poisson distribution. Further, the rates λ_i and λ_j at which messages enter queue i and queue j are constant during the interval approximated by the model.

2) General service times, i.e., the actual distributions of t_i and t_j in a system can be independently approximated with arbitrarily chosen distribution functions $F_i(t)$ and $F_j(t)$. These functions are assumed to have means $\bar{t}_i$ and $\bar{t}_j$ and coefficients of variation $c^2(t_i)$ and $c^2(t_j)$, respectively, where $c^2(t_i) = \text{var}\,(t_i)/\bar{t}_i^2$ and $c^2(t_j) = \text{var}\,(t_j)/\bar{t}_j^2$. If, for example, t_i is assumed to be exponentially distributed, $c^2(t_i) = 1$; if t_i is assumed to be constant, $c^2(t_i) = 0$.

3) Reversal times, i.e., intervals of duration r_{ij} and r_{ji} are required to switch the transmission capability from queue i to queue j, and vice versa. For example, r_{ij} could represent one or both of the following: a) turnaround time of the data link at the end of a transmission from computer i to computer j, b) delay within computer j before a transmission to computer i can begin. Intervals r_{ij} and r_{ji} may be constants or they may have inde-

pendent and arbitrarily chosen distribution functions $H_{ij}(t)$ and $H_{ji}(t)$. These functions are assumed to have means $\bar{r}_{ij}$ and $\bar{r}_{ji}$ and coefficients of variation $c^2(r_{ij})$ and $c^2(r_{ji})$, respectively.

4) Multimessage transmissions, i.e., once a transmission from a queue begins, the data link is not reversed until that queue has been emptied completely. This means that all messages joining a queue while that queue is being emptied are also included in the current transmission.

5) Adequate storage, i.e., sufficient storage is available so that no limit need be specified for the queue length at either computer.

A typical cycle of operation for the model is shown in Fig. 1. At the end of T_i', which represents the interval required to empty queue i, reversal time r_{ij} begins. On completion of r_{ij}, T_j commences, which represents the interval required to empty queue j. T_j is followed by another reversal time r_{ji}. A new cycle then begins with the start of the transmission period T_i. During a given cycle, either T_i, or T_j, or both may have zero duration, since either or both queues may be found empty. It has been assumed that the appropriate reversal time commences immediately when a given queue is found empty. If both queues are found empty during a cycle, the cycle duration reduces to the sum of the reversal times r_{ij} and r_{ji}.

During a typical cycle, messages accumulate at each computer during an accumulation period after which the accumulated messages are all transmitted. As an example, the accumulation period for queue i is $A_i = (r_{ij}+T_j+r_{ji})$. Let $\rho_i = \lambda_i \bar{l}_i$ represent the intensity of traffic offered to queue i. An equivalent definition for ρ_i is the fraction of time that messages are being transmitted from queue i; let $\rho_j = \lambda_j \bar{l}_j$ represent the corresponding fraction for queue j. The total fraction of time that the data link is being used for transmitting messages is therefore $\rho_i + \rho_j = \rho$, where ρ must be less than one for a steady-state situation to exist. Thus, $1-\rho$ is the fraction of time that the data link is idle, i.e., no messages are being transmitted in either direction. The components of the idle time are the reversal times r_{ij} and r_{ji}. If $\bar{I}$ is the total average idle time during a cycle, the average duration of a cycle is given by

$$\bar{C} = \frac{\bar{I}}{1-\rho} = \frac{(\bar{r}_{ij} + \bar{r}_{ji})}{1-\rho}. \tag{1}$$

An equivalent expression for $\bar{C}$ is $(\bar{T}_i + \bar{r}_{ij} + \bar{T}_j + \bar{r}_{ji})$.

Outline of Derivation

Since the processing of queue i and queue j is equivalent, formulas for the average delay and the average storage usage can be derived for one queue and then applied to the other by merely changing subscripts; queue i will be considered in this analysis.

With this model, every message joining queue i (or queue j) will be delayed. The delay encountered by a particular message depends on whether the message arrived while messages were accumulating in the queue or were being transmitted from it. Delay in queue i is also a function of n_i, the number of messages that accumulated in queue i during the preceding interval A_i. Cycles of operation beginning with n_i messages in queue i $(n_i = 0, 1, 2, \cdots)$ will be denoted by $C(n_i)$.

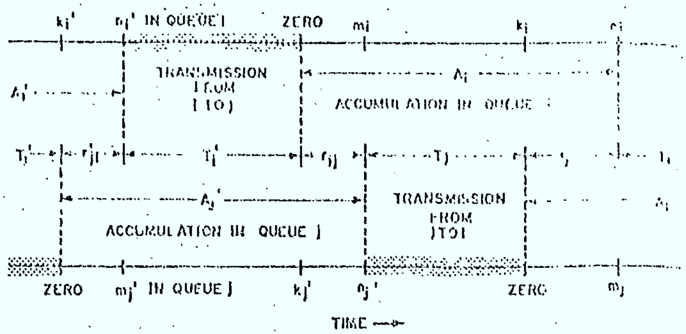


Fig. 1. Cyclical operation represented by single-server dual-queue model.

The average delay $\bar{D}_i$ of messages in queue i until the transmission of each begins can be determined as follows [1]:

$$\bar{D}_i = \frac{\bar{W}_i}{\bar{N}_i} = \frac{E[\bar{W}_i(n_i)]}{E[\bar{N}_i(n_i)]} = \frac{\sum_{n_i=0}^{\infty} P(n_i)\bar{W}_i(n_i)}{\sum_{n_i=0}^{\infty} P(n_i)\bar{N}_i(n_i)} \tag{2}$$

where

$\bar{W}_i(n_i)$ average value with respect to time of $W_i(n_i)$, sum of delays of all messages transmitted from queue i during $C(n_i)$ cycle

$\bar{N}_i(n_i)$ average number of messages transmitted from queue i during $C(n_i)$ cycle

$P(n_i)$ probability of $C(n_i)$ cycle

$\bar{W}_i$ average value of sum of queue i message delays during cycle of operation

$\bar{N}_i$ average number of messages transmitted from queue i during cycle of operation.

In order to evaluate the quantities appearing in (2), it is helpful to realize that the values of $\bar{W}_i$ and $\bar{N}_i$ would be unchanged if the following order of message transmission [2] were assumed for this analysis instead of the conventional first-come first-served order normally used in practice: the first message that joined queue i during the preceding A_i is transmitted followed by any that arrive during this initial transmission. Any messages that arrive during these subsequent transmissions are then transmitted, etc., until the "busy period" generated by this first message has been completed. At this time, the transmission of the second of the original n_i messages is begun, and a second busy period is generated. After a succession of n_i of these independent busy periods, queue i becomes empty.

Since these busy periods are identically distributed, the density function for each will be denoted by $B_i(t)$. The average duration [3] of each of these n_i busy periods will be denoted by $\overline{B}_i$, where

$$\overline{B}_i = \frac{\bar{l}_i}{1 - \rho_i}. \tag{3}$$

By analyzing this reordered transmission discipline, it can be shown that the expressions for $\overline{W}_i$ and $\overline{N}_i$ are

$$\overline{W}_i = \frac{\lambda_i E(A_i^2)}{2} + \frac{[\overline{n_i^2} - \bar{n}_i]\overline{B}_i}{2} + \frac{\bar{n}_i\lambda_i\overline{B}_i\overline{l_i^2}}{2\bar{l}_i(1 - \rho_i)} \tag{4}$$

$$\overline{N}_i = \bar{n}_i + \bar{n}_i\lambda_i\overline{B}_i = \frac{\bar{n}_i\overline{B}_i}{\bar{l}_i}. \tag{5}$$

The first term in (4) represents the average value of the sum of the delays, until the resumption of transmission from queue i, of all messages joining queue i during A_i. The second term represents the average value of the sum of the delays, while transmission from queue i is in progress, of those messages that accumulated during A_i. The last term in (4) represents the average value of the sum of the delays of those messages that join queue i while transmission from queue i is in progress.

By substituting (4) and (5) into (2), it is found that

$$\overline{D}_i = \frac{\rho_i E(A_i^2)}{2\bar{n}_i\overline{B}_i} + \left[\frac{\overline{n_i^2}}{\bar{n}_i} - 1\right]\frac{\bar{l}_i}{2} + \frac{\lambda_i\overline{l_i^2}}{2(1 - \rho_i)} \tag{6}$$

where

$$\overline{l_i^2} = \bar{l}_i^2[1 + c^2(l_i)]$$

$$\begin{aligned} E(A_i^2) &= E[(r_{ij} + T_j + r_{ji})^2] \\ &= \bar{r}_{ij}^2[1 + c^2(r_{ij})] + \bar{r}_{ji}^2[1 + c^2(r_{ji})] \\ &\quad + 2\bar{r}_{ij}\bar{r}_{ji} + \frac{\overline{n_j^2}\bar{l}_j^2}{(1 - \rho_j)^2} + \frac{\bar{n}_j\bar{l}_j^2[c^2(l_j) + \rho_j]}{(1 - \rho_j)^3} \\ &\quad + 2\rho_j\left[\frac{(\bar{r}_{ij} + \bar{r}_{ji})^2}{1 - \rho} + \frac{\bar{r}_{ij}^2c^2(r_{ij})}{1 - \rho_j}\right]. \end{aligned}$$

In order to evaluate the terms of (6), expressions are needed for $\bar{n}_i$, $\bar{n}_j$, and $\overline{n_i^2}$, and $\overline{n_j^2}$. Since the processing of queue i and queue j is equivalent, expressions need be derived for only one queue. The required expressions for queue i have been found by means of a generating function $G_{ij}(x,y)$ for the joint probability $P(n_i,m_j)$ of the following events: under steady-state conditions, at the instant T_i is to begin, n_i messages have accumulated in queue i, and m_j messages have accumulated in queue j. The expression for $G_{ij}(x,y)$ is

$$\begin{aligned} G_{ij}(x,y) &= \sum_{n_i=0}^{\infty}\sum_{m_j=0}^{\infty} P(n_i,m_j)x^{n_i}y^{m_j} \\ &= \sum_{\substack{n_i=0 \\ n_i'=0}}\sum_{\substack{m_j=0 \\ m_j'=0}} P(n_i,m_j \mid n_i',m_j')P(n_i',m_j')x^{n_i}y^{m_j} \end{aligned}$$

where $P(n_i,m_j \mid n_i',m_j')$ represents the following conditional probability: at the beginning of T_i in a given cycle of operation, n_i messages are in queue i and m_j in queue j, given that n_i' and m_j' were present at the corresponding instant in the previous cycle. The expression for this conditional probability can be written in terms of the probability of independent events in a manner similar to that used by Leibowitz [4]. After performing the necessary summations and integrations, the following expression results:

$$G_{ij}(x,y) = \psi_{ji}(\alpha)\psi_{ij}(\gamma)G_{ij}[\Phi_i(\delta),\Phi_j(\beta)] \tag{7}$$

where

$$\psi_{xy}(s) = \int_0^{\infty} e^{-st}\, dH_{xy}(t), \quad xy = ij, ji$$

$$\Phi_z(s) = \int_0^{\infty} e^{-st}B_z(t)\, dt, \quad z = i, j$$

$$\begin{aligned} \alpha &= \lambda_i(1 - x) + \lambda_j(1 - y) \\ \beta &= \lambda_i(1 - x) \\ \gamma &= \lambda_i(1 - x) + \lambda_j[1 - \Phi_j(\beta)] \\ \delta &= \lambda_j[1 - \Phi_j(\beta)]. \end{aligned}$$

Moments of n_i and m_j can be obtained by finding successive derivatives of $G_{ij}(x,y)$. For example,

$$\begin{aligned} \left.\frac{\partial G_{ij}(x,y)}{\partial x}\right|_{\substack{x=1 \\ y=1}} &= \left.\frac{\partial}{\partial x}\sum_{n_i=0}^{\infty}\sum_{m_j=0}^{\infty} P(n_i,m_j)x^{n_i}y^{m_j}\right|_{\substack{x=1 \\ y=1}} \\ &= \sum_{n_i=0}^{\infty} n_iP(n_i) \\ &= \bar{n}_i = \frac{\lambda_i(\bar{r}_{ij} + \bar{r}_{ji})(1 - \rho_i)}{1 - \rho}. \end{aligned} \tag{8}$$

Furthermore,

$$\left.\frac{\partial^2 G_{ij}(x,y)}{\partial x^2}\right|_{\substack{x=1 \\ y=1}} = \overline{n_i^2} - \bar{n}_i. \tag{9}$$

By using the relationship shown in (9), the following expression for $\overline{n_i^2}$ has been obtained:

$$\begin{aligned} \overline{n_i^2} &= \frac{\lambda_i^2(1 - \rho_i)^2}{(1 - \rho)^2}[\bar{r}_{ji}^2U + \bar{r}_{ij}^2V + 2\bar{r}_{ij}\bar{r}_{ji}] \\ &\quad + \frac{\lambda_i^2(\bar{r}_{ij} + \bar{r}_{ji})Z}{(1 - \rho)^2} + \overline{n}_i \end{aligned}$$

where

$$U = \frac{\{(1 - \rho)[1 + c^2(r_{ji})\{(1 - \rho_j)^2 + \rho_j^2\}] + 2\rho_i\rho_j\}}{1 - \rho + 2\rho_i\rho_j}$$

$$V = \frac{\{(1 - \rho)[1 + c^2(r_{ij})] + 2\rho_i\rho_j\}}{1 - \rho + 2\rho_i\rho_j}$$

$$Z = \frac{\{(1 - \rho_i)^2\rho_j\bar{l}_j[1 + c^2(l_j)] + \rho_j^2\rho_i\bar{l}_i[1 + c^2(l_i)]\}}{1 - \rho + 2\rho_i\rho_j}.$$

In order to obtain the average storage usage $\overline{S}_i$, an expression is required for $\overline{M}_i$, the average number of messages in queue i, including the one being transmitted.

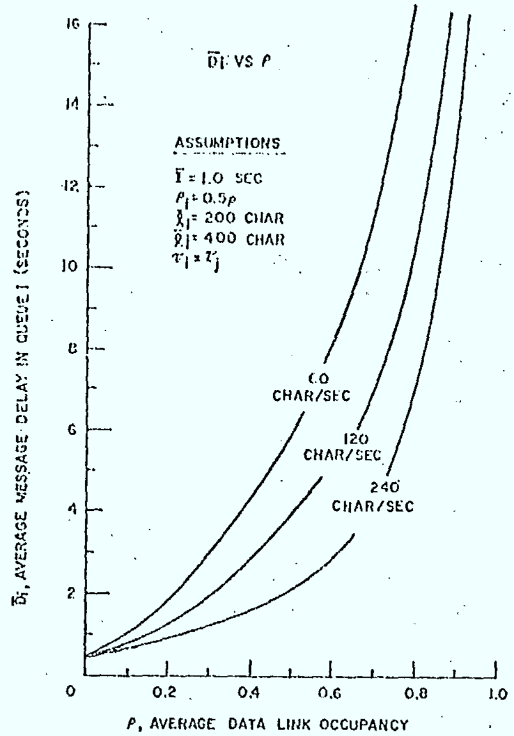


Fig. 2. Effect of transmission rate on average message delay in queue i; transmission time $\bar{l}_i$ not included.

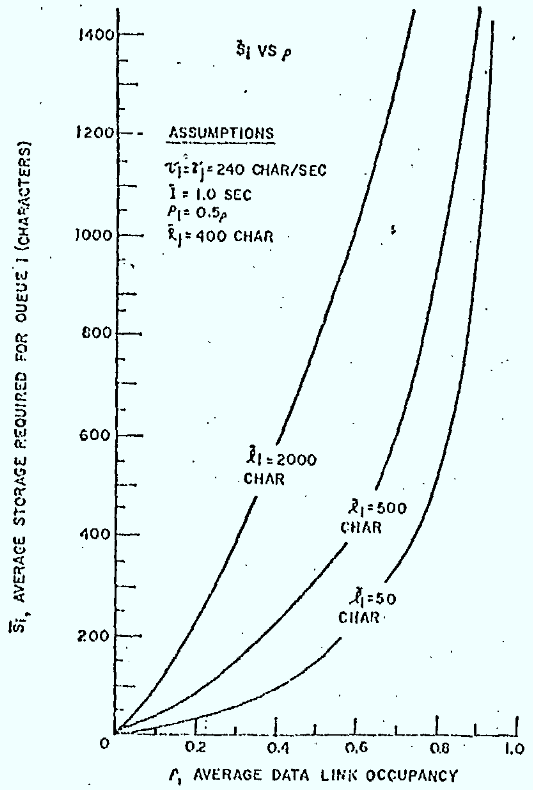


Fig. 3. Effect of average message length on average storage used for queue i; message being transmitted included.

The expression for $\overline{M}_i$ follows directly from $\overline{D}_i$, i.e.,

$$\overline{M}_i = \lambda_i[\overline{D}_i + \bar{l}_i] = \lambda_i\overline{D}_i + \rho_i. \tag{10}$$

It follows that $\overline{S}_i$ in characters is equal to $\bar{l}_i\overline{M}_i$.

Results

The equations presented in the previous section can be used to obtain results such as those shown in Figs. 2 and 3. For these figures it has been assumed that $c^2(l_i) = c^2(l_j) = 1$ and $c^2(r_{ij}) = c^2(r_{ji}) = 0$.

Fig. 2 shows how the average queue delay $\overline{D}_i$ is influenced by various transmission rates τ_i. It should be remembered that doubling τ_i and τ_j but keeping ρ constant results in the transmission of twice as many message characters. As ρ approaches zero, $\overline{D}_i$ approaches one half of $\overline{I}$, the sum of the average reversal times. Fig. 3 shows the storage usage values $\overline{S}_i$ as a function of ρ for three values of the average message length $\bar{l}_i$.

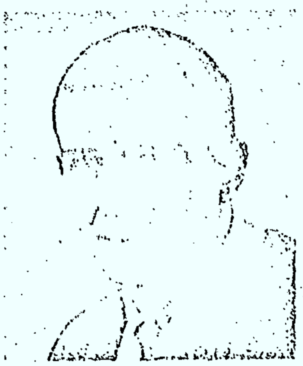
Many other figures could also be shown. For example, either $\overline{D}_i$ or $\overline{S}_i$ could be plotted as a function of $\overline{C}$, the average interval between successive transmissions of polling characters by the "master" computer. Another figure could show how $\overline{S}_i$ increases as a function of the total average reversal time $\overline{I}$.

Acknowledgment

Discussions with R. B. Cooper, D. P. Heyman, K. T. Marshall, and Grace Murray were helpful during the preparation of this paper.

References

[1] D. P. Heyman, "Optimal operating policies for stochastic service systems," Operations Research Center, University of California, Berkeley, Rept. ORC 66-31, 1966.
[2] L. Takács, *Introduction to the Theory of Queues.* New York: Oxford, 1962, p. 32.
[3] J. Riordan, *Stochastic Service Systems.* New York: Wiley, 1962, p. 63.
[4] M. A. Leibowitz, "An approximate method for treating a class of multiqueue problems," *IBM J. Res. Develop.*, vol. 5, no. 3, pp. 204–209, 1961.



Jack S. Sykes was born in St. Louis, Mo., on July 17, 1938. He received the B.S.E.E. degree (with high distinction) from the University of Arizona, Tucson, in 1961, and the S.M.E.E. degree from the Massachusetts Institute of Technology, Cambridge, in 1962.

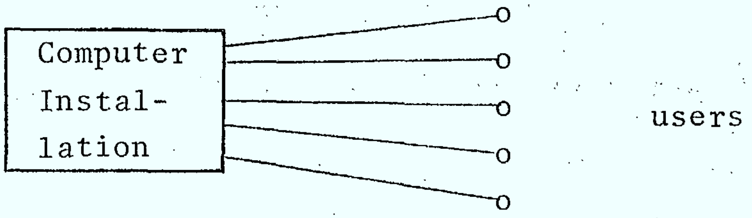
In 1961 he became a Member of the Technical Staff of Bell Telephone Laboratories, Inc., Holmdel, N. J. As a Systems Engineer, he has done analytical studies on various types of data communications systems. He is currently a member of the Computer Communications Studies Department.

Mr. Sykes is a member of Tau Beta Pi, Phi Kappa Phi, Sigma Pi Sigma, and the American Scientific Affiliation.

Communications Processor Hardware

Communications Processor Hardware

As has been pointed out in the volume 1 on Computer Considerations, a time-sharing system can basically be viewed as an operating system, a CPU with associated memory, peripherals, software and most notably a collection of terminals that remotely interface with it.



Communications Processor Functions

It is obvious, then, that part of the process to take advantage of the computer power in the installation must be associated with collecting the data at the terminals and assembling the information for time-shared processing. This is what is called communications processing. Briefly then the communications processor is that part of the computer installation that performs the following functions:

- assembles and identifies data words that are received from the remote terminals,
- detects transmission errors,
- does code conversion (i.e. to handle different types of terminals),
- assembles terminal messages in associated buffers in memory

- edits terminal messages as activation characters are detected (i.e. rubout, backspace, escape, delete line),
- notifies scheduler of status of tasks i.e.
  EOA - End of Address (beginning of message)
  EOT - End of Text (end of message).

A time shared installation implements these functions in ways that vary between two extremes. These are:

- a device called the terminal controller (T.C.) the first function above and the CPU performs all other functions <u>or</u>

- the T.C. performs the first function and a separate computer performs all other functions notifying the host computer of the status of various users (terminals). The T.C. is the common control element central to the effective running of a time shared installation.

Terminal Controllers

Terminal controllers are essentially multiplexers with buffers associated with the output channels, address and data registers associated with the input and a single interrupt line connected to the host computer. Schematically a terminal controller is shown in Figure 1.

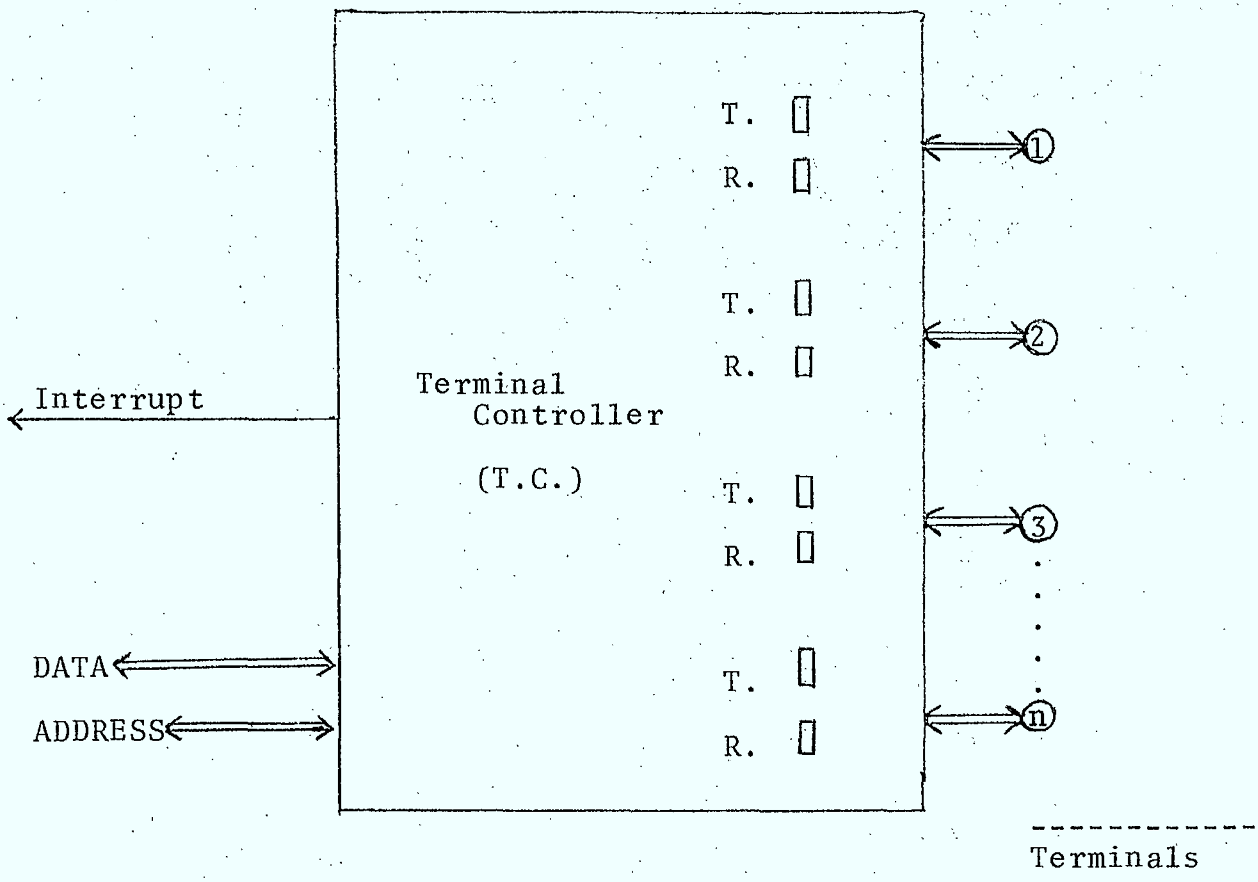


Figure 1.

In Figure 1, each I/O line to a remote terminal is connected to a pair of Transmit (T) and Receiver (R) registers. When a character is received in a register the hardware loads the address register appropriately, transfer the character to the data register and raises the interrupt to notify the communications processor that a character has been received. It is the responsibility of the CPU to "get" the character before another character arrives at the T.C.

Types of Terminal Controllers

There are basically two types of terminal controllers, namely non programmable and programmable.

In a non programmable TC (e.g. IBM-270X series), each "port" is hard wired to handle a single data type and speed; for example for

parallel data transfer
serial asynchronous data
serial synchronous data.

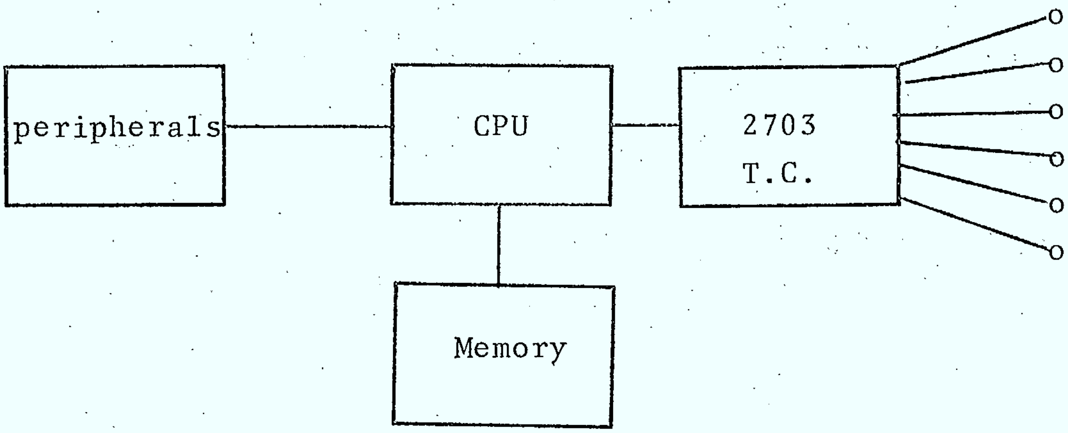
In a programmable TC (e.g. IBM-3705), each I/O line may be different in data rate and transmission mode.

Communications Processor Configurations

There are also three basic communications processor configurations, which are described below using schematics.

Type 1: Computer-Terminal Controller

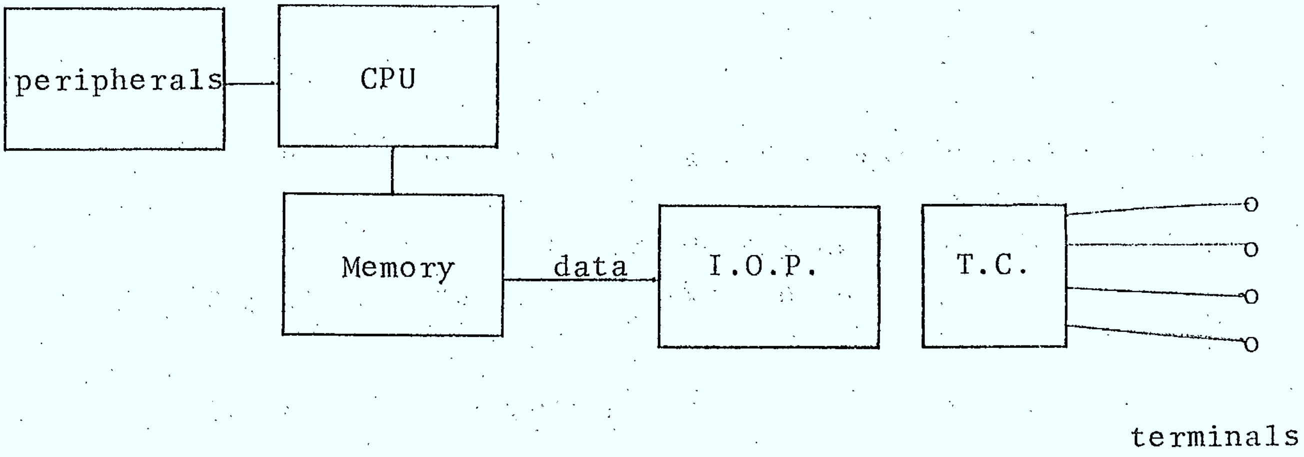
e.g. IBM 360/50 and 2703 T.C.



The scheduler in the 360/50 is interrupted by the 2703 each time a character is received. It must then get the character, buffer it, do any necessary code conversions and check for activation characters.

Type 2: Computer - I/O Processors T.C.

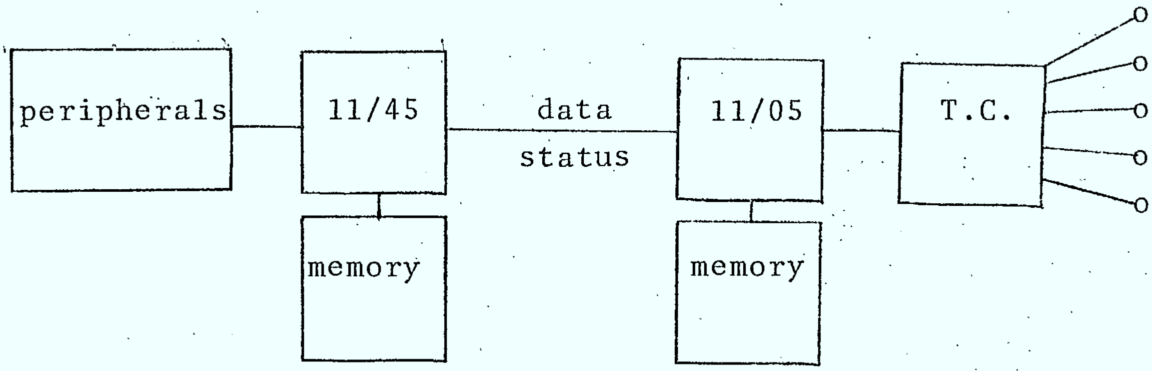
e.g. CDC-6600 and T.C. with I/O processor



In this configuration a device called the I/O processor takes some of the functional load off the CPU. It is a very primitive computer that can be programmed to read and write data directly into the core of the computer. The I/O processor associates terminaladdresses with buffers in core but leaves all further communications processing to the main CPU.

Type 3: Computer - Communications Computer - Term. Controller

e.g. PDP-11/45 and 11/05 T.C.



The communications processor functions are all handled by the 11/05. Since data associated with terminal I/O is stored in the 11/05, a "handshaking" discipline must be set up between the 11/45 and 11/05, to notify the scheduler in the main processor of terminal status. Both data and status transfer occur between the two computers. The preceeding is only a brief outline of the hardware architecture of various time-shared computers.

References for Communications Processor

Hardware

1-e/ vi

Communication Control By Computer - An Introduction

MICHAEL J. TOWNSEND
sub. of GTE Information Systems, Inc./Tempo Computers Div.
Huntington Beach, Cal.

ABSTRACT

It is the author's objective to introduce the communications-oriented reader to the importance of computer-controlled communication: how it is accomplished, its extreme economy if properly applied, and wherein lie the major applications of this important technology.

A BIT OF BACKGROUND

The growth of data communications, in terms of equipment installations and carrier facility usage, has been thoroughly documented, debated, and analyzed. Not as well considered, however, has been the changing relationship between data-communications technology and the data-processing world. Yet, changes in that relationship are largely responsible for data communication's explosive growth.

Early computer applications were designed around straightforward *batch* tasks, for the most part requiring little movement of information into or out of the computer's immediate environment. Since those early applications, two decades have passed and billions of dollars have been spent. Now computer hardware and software are enormously more powerful and sophisticated, and the tasks to which computers are being applied are tremendously more complex.

This complexity takes many forms, but of interest here is that now there are greater and greater demands for remote data collection; for delivery of computer outputs to a wider and wider audience; and for the availability of *real-time* responsiveness.

This is the reason for the growing reliance on data communications by data-processing system designers. Large computers have become the majority users of data-communications technology. But, as indicated, the relationship is changing: increasingly, computers are being used to manage the communication process, as well as to reap its benefits.

Sometimes this communication-management function resides within the user computer. In other cases, computers are dedicated to communication management. Dedicated computers can stand by themselves, as in conventional message-switching systems, or they can be specialized appendages to some larger computer and be either locally or remotely located.

Whatever the case, it is the involvement of the computer — its programs, peripherals, and computational power — in communication control that permits precise, automatic coordination of information flow. As data-processing systems grow more complex, this control becomes more and more necessary to the system designer's objectives.

Remember that computers are intricate devices, and that their care and feeding is a burden not to be assumed lightly.

As a result, the aggregate transmission capability of modems connected to the Bell System has risen from about 250,000 characters per second (cps) in 1962, to nearly 10,000,000 cps today. The forecast is for a capacity of 100,000,000 cps by the end of 1975.

Computer-controlled communication has itself evolved considerably in the past decade. Starting with the pioneering adventures in replacing *torn-tape* and selective-calling functions, the technology has advanced to the point where literally thousands of computers are now used exclusively in the communication-management process. Such applications will grow nearly ten-fold in the next five years.

CONTEMPORARY FACTORS

What's presently happening in the data communications and computer industries?

Growth

Foremost in people's minds, of course, is the enormous growth referred to previously. Modems and acoustic couplers are presently being installed at the rate of about one every working minute of the year. Other communication expenditures vary in proportion to the amount of transmission capability.

Less well understood, however, is that data rates are increasing over most channel types also. This is a crucial point because, as data rates increase and available bandwidth is more efficiently utilized, communication-control problems increase. Contrast, for instance, the difficulty in communicating with a conventional 110-baud teleprinter terminal with that of a 1200-baud CRT display terminal.

The more complex the communication process, the more expensive the mechanization of communication control becomes, and the more the need for cost-effective use of computer flexibility. Thus, as complexity increases the demand for programmable control grows.

Carriers

As the demand for more data communication equipment and circuits grows, the carriers — common and specialized — are struggling to keep up. Already, experts are forecasting that by the end of this decade the amount of data traffic on the nation's telephone networks will exceed the amount of voice traffic.

As proposals for various types of specialized carrier services multiply, the common carriers themselves are hastening to add new services, equipment, and capabilities of their own. In this sense, at least, competition in this industry is beginning to benefit the ultimate user. The dark cloud surrounding this silver lining, however, is the effect increased traffic loads — both data and voice — are having on existing facilities. Planned some years ago, a number of telephone facilities are already strained to capacity.

The resulting delays, error rates, and outright breakdowns in areas of traffic concentration have tended to inhibit the growth of communication applications. Gradually, these problems will be rectified, of course, but progress in the telephone industry takes longer than in the computer industry, by a significant margin.

. . . in many cases overall EDP system organization relegates communication functions to a relatively low priority.

Large Computer Design Trends

Just as early common-carrier planning largely ignored future data-communication requirements, so did early computer designs. Although the computer industry is rapidly acquiring the special skills necessary to cope with real-time applications, in many cases overall EDP system organization relegates communication functions to a relatively low priority. Inevitably this leads to a low efficiency, both in the utilization of the network connected to the large computer and in the execution of communication functions within the computer and its terminals.

This is *low efficiency by design default*, and can usually be traced to a fundamental characteristic of large computers. For the sort of tasks most large computers are used, the bigger the computer the lower the cost of doing a given job. While there are all sorts of qualifications to this maxim, the tendency remains to design and install larger and larger computers, so that their centralized resources embrace as much of the logic mechanization of a data-processing system as feasible.

Lumping everything together in a massive central-processing unit frequently results in low efficiency in the execution of those functions not central to the designers' intent. Unfortunately, communication-control functions are all too often in this category.

System Integrity

As communication applications grow; as terminals, modems, and the like are spread throughout the United States; as computers get bigger and bigger; the problem of system integrity becomes of paramount concern.

. . . the entire life-blood of a company may flow through its computer and communication systems . . .

In the days when the EDP system provided a helping hand to an otherwise manual accounting department, failure of that system was less than a corporate catastrophe. Those days are fading fast; frequently, the entire life-blood of a company may flow through its computer and communication systems, and outages for more than a few seconds or minutes may spell disaster.

Computers differ from communication networks in types of failure; the problems of system integrity are far harder to solve when the computers are used to control the communication process. System designers are finding it necessary to be more and more preoccupied with questions of reliability, maintainability, error detection, fault recovery, critical element redundancy, data-base privacy, etc.

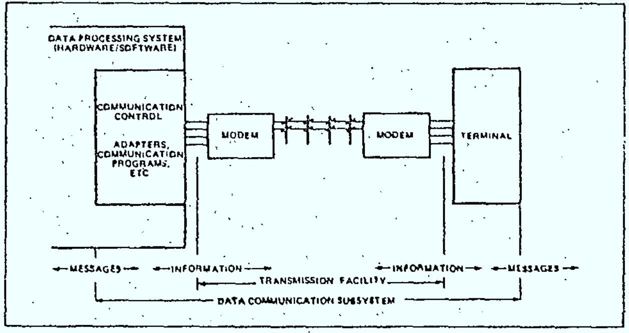

DATA PROCESSING SYSTEM (HARDWARE/SOFTWARE)
COMMUNICATION CONTROL
ADAPTERS, COMMUNICATION PROGRAMS, ETC
MODEM
MODEM
TERMINAL
MESSAGES
INFORMATION
INFORMATION
MESSAGES
TRANSMISSION FACILITY
DATA COMMUNICATION SUBSYSTEM

Fig. 1 Teleprocessing system organization.

ORGANIZATION OF TELEPROCESSING SYSTEMS

Large data-processing systems that use data communications perform a variety of functions, only some of which relate to the actual control of communications. Understanding where communication functions stop and other EDP functions begin — in other words, the true extent of communication involvement by the data-processing system — is vital to this article's subject.

(Continued on page [illegible]

COMMUNICATION CONTROL

(Continued from page 34)

Figure 1 shows a routine arrangement: a *data-processing system*, a *transmission facility* consisting of modems and (in this case) telephone lines, and a *terminal*. The entire complex is termed a *teleprocessing system*.

The data-processing system itself is shown with a major portion emphasized: communication interfaces and control units, interconnecting channels, communication monitor and application software, etc. This part of the data-processing system, together with the modems, lines and terminals, is the *data communication subsystem*. The key point in Figure 1 is that messages flow across the boundaries of the data-communication subsystem, into and out of the non-communication portion of the data processing system, and into and out of the terminals.

By contrast, the transmission facility itself deals only with *information* — in the fundamental sense, 1's and 0's — to be transferred from one part of the data-communication subsystem to another. The distinction is that messages are organized information — organized for transmission purposes, and for their subsequent non-communication use.

From this come the following fundamental definitions: we speak of *transmission* as the process of transferring information from one point to another. When we wish to organize and manage that process in order to move messages from one point to another we are engaging in *communication*, and all the system elements that contribute directly to that objective are part of the *data-communication control function* of the teleprocessing system.

The simplifications in Figure 1 do not alter the basic definition of communication control. For example, in place of the terminal at one end of the transmission facility there might well be another data-processing system, with its own set of communication interfaces, programs, etc. Also, transmission facilities typically are far more complex than that shown. Switched connections, multidrop configurations, etc., add transmission complexity but do not alter the function. (It has been through ignoring the vital distinction between transmission and communication that many early mistakes were made in applying and planning for on-line, real-time systems. These require an intricate orchestration of computer hardware, operating systems, application programs, transmission facilities, terminals, communication executives, human operators and the like; they are not achieved by *hooking modems to the accounting department's computer.*)

COMMUNICATION CONTROL FUNCTIONS

With a teleprocessing-system organization as described above, there are limitless opportunities to use the data-processing system's computer to control the communication process — or to employ dedicated computers to the same end. In either case, if computers are to be used, what will they do?

The Hierarchy of Communication Procedures

All communication processes, whatever their degree of complexity and means of implementation, involve a nested procedural hierarchy. This is illustrated in Figure 2.

The inner procedure in this nest is *message transfer* and is, of course, the whole point of data communications. But, before message transfer can take place a *circuit* must be established that provides electrical connection between two or more parties, each potential communicants.

Having provided a connected circuit, it is also necessary to establish a *link*, or logical path, between two particular parties who wish to communicate. Since the circuit can (in principle) bring together any number of parties for potential communication, the link relates a particular pair by providing positive identification, each to the other, and defining the terms under which communication is to proceed, e.g., who sends and who receives.

Clearly, before two different parties can communicate, an existing link must be dis-established and another established. If the process requires a party or parties not part of the existing circuit, they must be disconnected and new ones connected. All of this may seem obvious, but it is emphasized here for an important reason that may not be apparent immediately.

Computers . . . are exasperatingly intolerant.

When communication processes are not automated, human operators are responsible for circuit connection and, sometimes, link establishment. More importantly, they are also responsible for coping with failures, breakdowns, and other aberrant behavior of the network during these processes.

Fortunately, humans are remarkably tolerant of aberrant behavior and are capable of amazing amounts of compensative action. Computers, on the other hand, are exasperatingly intolerant. In fact, they are only capable of dealing with unexpected behavior of a portion of the teleprocessing system when a programmer or engineer has anticipated the problem in the first place, something of a contradiction in terms.

Converting unconscious or routine human reactions into precise computer routines is a tricky business, requiring considerable experience and effort. The great majority of energy expended in automating communication control is devoted to coping in advance, as nearly as possible, with strange or faulty behaviour of the communication subsystem.

Since much of this concern is concentrated in the two *outer* procedures in Figure 2, a clear understanding of the hierarchy is essential.

Communication-Process Control

Within the framework of the activities described above, a number of functions must be performed by a computer in order that it control the total communication process. A number of other functions may be performed, depending on the application and the extent of control exercised by the computer.

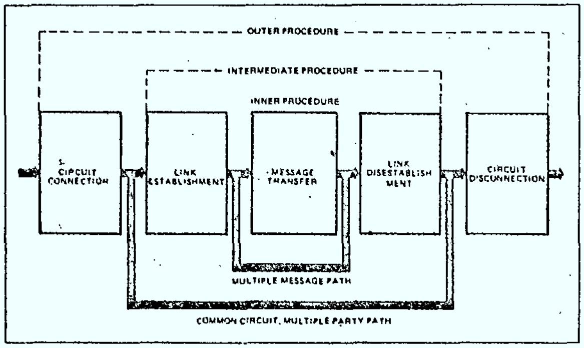


Fig. 2 Communication procedures heirarchy.

Table 1 contains a list of the functions generally associated with communication control. While not exhaustive, the list is at least indicative of the many dimensions of communication control. The list is arranged so that the simplest, most fundamental functions are at the top and the most complex are at the bottom. As one might expect, those functions at the top are common to nearly all communication systems, while those close to the bottom are more specialized, more variable from one application to the next, and less likely to appear in simpler systems.

TABLE 1
COMMUNICATION-CONTROL FUNCTIONS

LOW-LEVEL FUNCTIONS

Interface Signalling
Bit Synchronization
Character Synchronization
Character Assembly/Disassembly
Connection Control
Polling/Selection/Contention
Terminal Control
Message Assembly
Error Detection and Control
Code Conversion
Data Compression/Expansion
Job-based Routing
Network Performance Monitoring/Analysis
Terminal Testing

HIGH-LEVEL FUNCTIONS

Address/Based Routing
Queueing/Dequeueing
Prioritization
Reconfiguration Control (Network, Computer)
Fail-Soft Functions
Time Control/Recording
Format Control
Format Conversion
Communication File Maintenance

Each function in Table 1 is explained below,

- *Interface signalling* refers to handling the many individual signals at the interface between the transmission facility and the computer or terminal. In the more complex situations, there can be as many as a dozen individual control signals per circuit.
- *Bit synchronization* is the process of detecting the timing of the flow of information across the transmission-facility boundary. Asynchronous operation requires that this be done in the computer or terminal; synchronous operation implies that the transmission facility is responsible for bit-rate timing detection.
- *Character synchronization* involves the detection of character boundaries in incoming data.
- *Character assembly/disassembly* provides a conversion from the serial nature of the transmission facility to the parallel organization of the computer and the terminal.
- *Connection control* includes dialing and answering, whether automatic or manual. It can also involve intricate manual patching or line-switching operations for dedicated or private circuits.
- *Polling* is the process of sequentially addressing a series of terminals on the same circuit, to permit them to transmit. *Selecting* (or *calling*) is the reverse process. *Contention resolution* is required when two or more parties have the opportunity to communicate over the same circuit at the same time.
- *Terminal control* includes the functions necessary to exercise the various capabilities of whatever terminals are in use. For example, in a CRT display terminal these might include cursor control or hard-copy printouts.
- *Message assembly* involves the accumulation of characters, blocks, records, or other message components until a complete, coherent message has been received and verified.
- *Error detection and control* refers to both routine and nonroutine sources of errors: random line hits, for instance on the one hand, or total line dropouts or misconnections on the other hand. Properly executed, error control can involve highly sophisticated recovery procedures, depending on the error and the extent to which it has propagated through the teleprocessing system.
- *Code conversion* is a necessity in any extensive teleprocessing system; there are a dozen major transmission codes presently in use. Curiously, the most widely used code is also the oldest: 72 years older than the first electronic computer!*
- *Data compression and expansion* involves the removal, and subsequent restoration, of redundant data; it permits more efficient utilization of transmission facilities, although frequently at the expense of more complex communication-control equipment.
- *Job-based routing* permits messages to be routed to different locations or resources according to the task to be performed on them, or according to the function that produced them.
- *Network-performance monitoring and analysis*

gives the teleprocessing system user the information necessary to optimize the use (hence minimize the cost) of his communication network: traffic statistics, line-performance histories, connection performance, etc.

- *Terminal testing* lets the user spot trouble and, in some cases, deduce its appropriate cure from a central site. This is especially important where the network is extensive, or where the terminals are far from a competent source of diagnosis.
- *Address-based routing* — the essence of message switching — causes messages to be dispatched, routed, logged, or stored according to self-contained addresses or directory information.
- *Queueing* is the process of arranging messages in transmission order, particularly when they are produced by diverse job functions or received from diverse sources within the teleprocessing system.
- *Prioritization* involves applying priority criteria to the *queueing* function, above, so that the more important messages get transmitted or otherwise acted on first.
- *Reconfiguration control* is necessary for systems containing redundant elements, and capable of automatically operating around a failed component. This involves fault detection, reconfiguration, recovery and, finally, system restoration.
- *Fail-soft* (or *Fail-safe* — Ed.) functions are used to keep portions of a teleprocessing system, such as terminals, operating when a major element has failed, e.g., the main computer itself.
- *Time control and recording* permits accurate auditing of communication transactions, and is vital to successful recovery from certain types of failures and errors.
- *Format-control functions* allow the data-processing system to guide data-entry procedures at remote terminals, by interacting in a variety of ways with the human operators as they enter the data.
- *Format conversion* involves the restructuring of messages prepared in one format into another. Depending on the formats involved, this can be extremely difficult but diversity of tasks and users in a system may make it unavoidable.
- *Communication-file management* is employed in the most extensive communication systems to organize, manipulate, and maintain the correspondingly extensive files of communication transactions, in order to permit efficient and accurate retrieval, updating, and disposal.

Within a computer system it is important to make a distinction between *low-level* and *high-level* functions. In the latter category lie those functions whose implementation in the computer usually involves substantially higher expense than the former category: more extensive facilities, and far higher design costs.

Since the high-level functions tend to be more specialized, commonality of design and implementation from one system to the next is much less feasible. Thus, high-level communication control tends to be a custom-design project, and therefore tends to cost more, have more undetected design flaws, be less completely documented and understood, and be harder to change.

With the significant increase in investment represented by moving from the low-level domain to the high-level domain of automated communication control, there comes a heightened concern for system reliability and integrity. Hardware redundancy, automatic fault detection, dynamic reconfiguration, etc., are tools frequently used to preserve the benefits of the high-level of communication control. These, in turn, serve further to complicate the computer system and add to its cost.

WHY USE COMPUTERS?

Early communication-system controllers were strictly non-programmable; in fact, some were (and still are) mechanical. What are the reasons for changing from these conventional schemes to the use of computers for controlling communications?

Essentially, the reasons fall into two categories: (1) concentration of resources, and (2) flexibility.

Concentration of Resources

Communication-control and data-processing functions tend to be highly repetitive: in multiple-circuit teleprocessing systems, what you do for one line must pretty well be done for all of the rest of the lines, or perhaps each message or each character. Since such functions are repetitive it is possible to devote a single physical resource to their performance, executing each very rapidly in order to handle the demands of a large number of lines, characters, or messages. This, of course, is just what most computers are designed to do. As a result the system designer has at his disposal a physical resource more expensive than a nonprogrammable controller for a small number of lines, characters, or messages, but much less expensive for the larger jobs, where nonprogrammable approaches represent significant duplication of hardware.

This important cost relationship is shown in Figure 3, where the cost of communication control by two techniques in a hypothetical application is related to system size or complexity. System complexity might be measured in a number of ways: number of lines, throughput, amount of *high-level* control activity required, etc. Not all measures of system complexity result in this relationship, but it happens often enough to be enormously useful.

Clearly the potential application of communication controllers in real communication systems must be scrutinized individually to determine

COMMUNICATION CONTROL

(Continued from page 38)

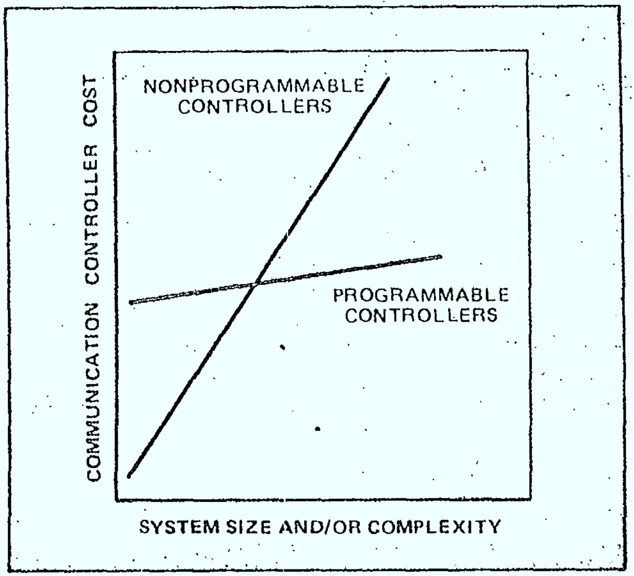


Fig. 3 Comparative costs, programmable versus non-programmable communication controllers.

where they might lie on a diagram such as Figure 3.

In evaluating the cost of any communication controller, programmable or otherwise, the total expense must be assessed. In the case of programmable devices, this must include program maintenance and the relatively high cost of acquiring and maintaining in-depth familiarity. For nonprogrammable controllers, the cost of the device's relative inflexibility and difficulty in expansion (see below) must be assessed.

Computer Flexibility

The number of ways the flexibility of the modern digital computer can be utilized in the control of communication is almost without limit. Some of the most prevalent uses are outlined below.

- Terminal Flexibility. Computers permit communication users to select from a wide variety of terminal types, communication procedures, data rates, codes, formats, transmission techniques, etc. By contrast, nonprogrammable controllers are inevitably restricted by their initial design to a particular set of terminal types.
- Network Expansion and Modification. Since computers concentrate most of their hardware cost in the central processor and related peripherals, the expense of adding lines, changing or upgrading terminal types, etc., is relatively small compared to nonprogrammable controllers.
- Fail-Soft Effects. Control of communications by dedicated computers permits caretaker operation of a network, including its terminals, when the large EDP computer has failed or is otherwise out of service. These effects can include the transmission of standby messages to terminal operators, the collection of input data on behalf of the large computer, the provision of alternate access to data bases, etc. These functions may be performed by either local or remote communication computers.
- Error Control; Traffic Protection. Programmability of communication control permits the use of more elaborate and sophisticated forms of error control and fault recovery — or even the provision of error control where none previously existed, as in the substitution of a concentrator for a hardwired multiplexer.
- Communication Management. About the only practical way of keeping track of the condition and quality of communication lines in a multi-line system is by means of a computer. Its ability to store events and analyze error statistics is essential to developing, promptly, an adequate picture of communication-line performance.
- Large-Computer Economies. With a programmable communication controller (particularly a front-end controller; see below), a number of economies can be achieved in the large EDP computer with which it is associated. These economies depend on a judicious partitioning of communication functions between the dedicated computer and the large machine, in order to optimize the behavior of the complete system. No two systems are the same, of course, but some of the savings in the large computer that can be realized include: interrupt-processing-load relief; memory savings; channel-traffic savings; and background-processing-load relief. In this last example are included important economies that can be realized by relocating *high-level* communication-control functions to a dedicated communications computer.

TYPES OF COMMUNICATION CONTROLLERS

At one time, communication control by computer was almost synonymous with *message switching* and, indeed, several contemporary communication-oriented computer designs are based on time-honored message-switching principles.

Nevertheless, the 1970's will see the advent of the small dedicated computer in communication control applications. Figure 4 indicates the forecasted growth of such installations.

Among this growing population there are five distinct categories of use, explained below.

FRONT-END COMMUNICATION CONTROLLERS. These are computers connected directly to a large EDP machine, usually by means of one of the latter's input/output channels or ports. Among other things, front ends provide the user with increased terminal flexibility, communication operation, and the opportunity to absorb many communication functions normally performed, sometimes very inefficiently, by the large machine.

(Continued on page 62)

COMMUNICATION CONTROL (Continued from page 60)

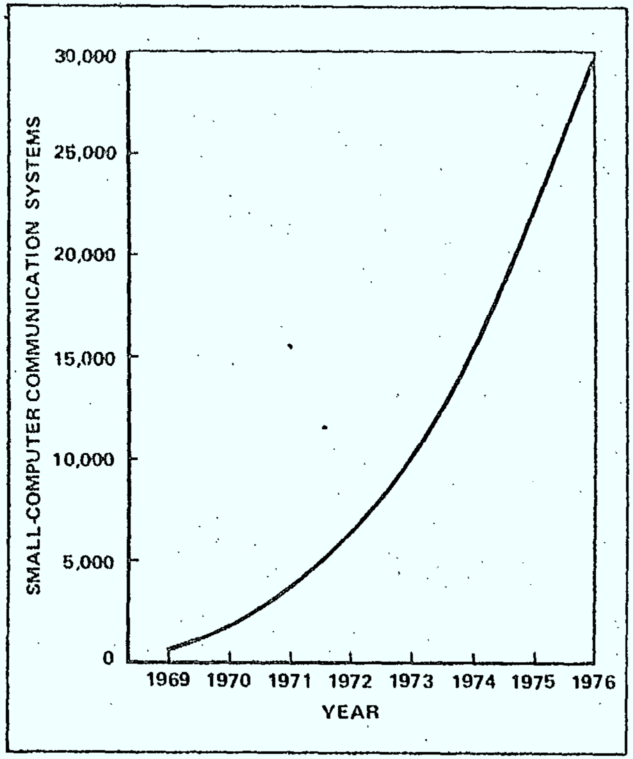


Fig. 4 Estimated installed base of small-computer communication systems, 1969-1976.

REMOTE-DATA CONTROLLERS. These are also called *intelligent-batch terminals*, and are designed to replace ordinary remote-batch terminals to provide data preprocessing, higher-performance peripheral devices (printers, card readers, etc.), faster data rates, or other enhancements to conventional batch terminal operation.

REMOTE COMMUNICATION CONTROLLERS. Computers are sometimes located at sites remote from the large EDP computer, for purposes of providing an intermediate level of communication control on behalf of the large machine. These usually are justified by the resulting savings in network costs and in the ability to extend error control to remote locations. Concentrators fall into this category, as do such relatively new applications as *intelligent* CRT display terminal controllers.

STAND-ALONE COMMUNICATION CONTROLLERS. These include the various types of message switches presently in use and yet to be installed. Typically such systems require extensive planning involving multiple processors, considerable high-level functional performance, extreme measures in the protection of traffic, etc. They are *stand-alone* systems in that message switching is their primary purpose, and while there may be connections to one or more large EDP machines, the relationship is more remote than the previous categories.

LINE-SWITCHING SYSTEMS. More and more often, telephone companies and other carriers (and some industries) are experimenting with the use of computers in line- and circuit-switching applications, where the computers' flexibility provides enhanced, more customized switching functions.

WHAT TO WATCH OUT FOR

Despite the fairly enthusiastic picture painted in this discussion, the use of computers to control communications is not without a set of unique problems.

> **. . . don't use a computer for communication control if the job can be handled efficiently in some other way.**

KEEP IT SIMPLE. Computer applications are the easiest thing in the world to go overboard into, and unless the user is careful, he can get involved in a cyclic justification that goes something like this: *"I know communication is complex and my needs unpredictable, so I'll use a computer to control the whole thing, taking advantage of its flexibility . . . but now that I have a computer, I'll have to expand and complicate my network, terminals, etc., in order to take best advantage of the computer."*

A good rule of thumb is this: don't use a computer for communication control if the job can be handled efficiently in some other way — making necessary allowances for the requirement and cost of the humans that interface with the system, the need to expand and otherwise hedge against the future, etc.

Remember that computers are intricate devices, and that their care and feeding is a burden not to be assumed lightly.

WHERE'D HE GO? A persistent problem in any end-user-oriented application of computers, communication control included, is the problem of the vanishing salesman: once the concept is born, a procurement made, and a delivery and installation accomplished — where is the necessary support?

Be objective in evaluating your total needs and your own strengths and weaknesses, and deal only with a computer or system vendor capable of providing *whatever you lack*, not just hardware or perhaps, an initial turnkey system.

WHERE'D THEY GO? Another persistent problem in the small computer industry (well, for that matter, in the large computer industry, too!) is that of the disappearing vendor. Deal only with vendors whom you have good reason to believe will be in business — the SAME business — for the life of your product.

Despite your best efforts, careful investigation of track record, the most exhaustive and objective specifications, you are best assured of a communication-control computer you will be satisfied with if you obtain it from a vendor who will depend on your satisfaction for his future business — from you and from others.

In some systems the central processor spends too much time fielding communications inputs. Here HAL BECKER of Honeywell's Information Systems division talks about the special functions of a system specifically tailored to ease communications

Communications processing for large data networks

In large scale teleprocessing systems, the communications network can impose much overhead on a central processor trying to cope with random incoming data. Often the central computer spends more time receiving information than processing it. One solution is to add a second computer to handle the communication inputs. But the price/performance ratio of the total system usually worsens. A communications computer must be designed and optimized for its particular task.

Just what functions should a communications computer perform? An evaluation indicated the following key tasks:

- free-standing administrative message switching
- front-end communications processing for remote batch systems
- remote concentration

An examination of specific methods and experiences for fulfilling these functions will demonstrate the major concepts that go into a communications-oriented processor.

Early in the development phase, attempts were made to define the above communications functions and other related tasks, and to design a specific computer for each. But this approach was later abandoned because the functions overlap and because the communications processor often performs two or more tasks concurrently.

In order to provide a single system that could be applied to any data communications task, and to accomplish an optimum price/performance ratio, a modular, building-block approach was adopted. This resulted in a system composed of four basic subsystems: communications, processor, memory, and peripheral I/O.

The communications subsystem accommodates all five standard line classifications:

- low-speed asynchronous
- medium-speed asynchronous
- medium-speed synchronous
- low/medium-speed parallel
- high-speed synchronous

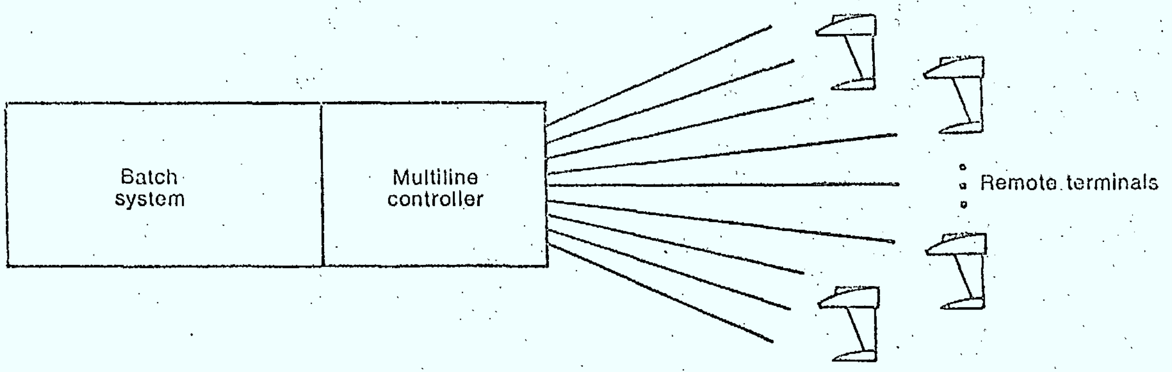
The basic element of the communications subsystem is a plug-in unit that will accept combinations of channels up to a maximum of 32 full-duplex, low-speed asynchronous; 16 full-duplex medium-speed (asynchronous or synchronous); or 16-parallel channels.

Bit rate, code level, and where needed, synchronization characters may be selected by the system software. This flexibility enables the system to provide a given grade of service to the terminal environment with fewer channels.

For example, a low-speed asynchronous channel may interface with a Model 35 Teletype operating at 110 bps at one time, and with a Model 37 Teletype operating at 150 bps at another. The ability to vary channel parameters eliminates the often encountered problem of a terminal being denied access because the only open channels in the system are fixed to operate at a different speed.

Each channel in a communications module has loop-back capability. Should the processor detect faulty operation on a channel, it can connect the transmit and receive sides of the communications channel, and send out a character sequence on the transmit side to see if the same sequence is observed on the receive side. This action enables the system to determine whether the fault is in the channel. If it is, the processor will remove the channel from service and send an appropriate message to the network supervisory station. The system can accept up to eight communications modules giving a maximum of 252 full-duplex, low-speed, asynchronous channels; 126 full-duplex, medium-speed channels; or combinations of the two.

The processor subsystem uses an instruction set of about 100 commands—many of them character-processing instructions—to facilitate the character-by-character processing function, an inherent requirement in the data communications environment. Since the system memory is word-oriented, the need to pick up a word, shift or mask off the unwanted character(s), and position the desired character so it can be processed is eliminated. Diagnostic commands which permit the processor to exercise malfunctioning elements of the system and determine what corrective action is required are also included in the

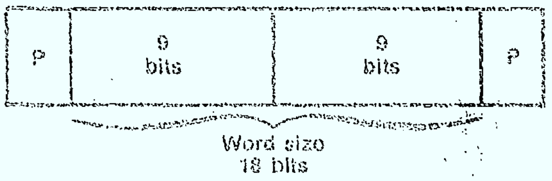


In early data communications systems, a single batch processor was used to process data, and through multiline controllers, handle the communications control. Increasing size and complexity of the systems soon made this unwieldy.

instruction repertoire. Binary add and subtract commands are standard, and for those applications needing more arithmetic capability, hardware multiply and divide commands are optional.

The processor contains dual arithmetic units that operate in parallel. All arithmetic functions—add, subtract, multiply, divide, shift, etc.—are performed in each unit, and the results checked for agreement before the processor is allowed to proceed. In this way, hardware malfunctions in the arithmetic unit are detected and the processor notified so that it can execute the appropriate diagnostic commands to determine if the failure is hard (permanent) of soft (transient) and take the proper corrective action. Three program-accessible working registers are provided, including a channel address register for use in connecting a specific channel to the processor logic and to facilitate channel addressing of the many channel oriented tables found in data communications programs.

Four hardware channel macro commands ease communication between the processor and the configured channel complement. A read-only-memory (ROM) is available as a processor option. It contains 265 words of predetermined, fixed, reboot/restart code that can be read into memory, under processor or manual control, and executed. The code contains the logic necessary to bootstrap the system software from either a local card reader, a local typewriter, or a remote computer over a communications channel.



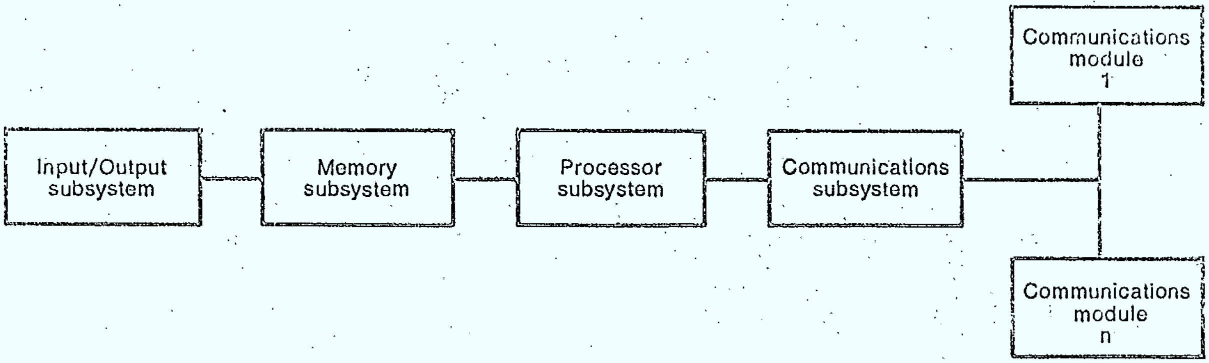
The organization of a word in memory provides for two nine-bit characters plus a parity bit for each character. Parity is checked each time a character is entered or removed from memory. This provides a high degree of internal message integrity.

The memory subsystem is word-oriented, with a word length of 18+2 bits. Each word consists of two 9-bit halves (considered a character by the character-addressing instructions) and a parity bit associated with each half.

The memory access is 600 nsec, and the cycle times are 1.2 μsec for the read-restore (R/R) and clear-write (C/W) functions, and 1.8 μsec for the read-alter-write (RAW). Both cycle times include a 200 nsec logic delay period. The C/W and R/R are used for straight data transfers between memory and the working registers. The RAW cycle is used to pick up a word (from memory), alter or modify it, and place it back in the same location. An example of this would be the assembling of an incoming character from a low-speed, asynchronous channel into a computer word. The word is loaded into the arithmetic unit and shifted left one bit at a time as the character is read in, then the word is stored in its original location. This is accomplished in 1.8 μsec using the RAW cycle, as opposed to 2.4 μsec if the R/R and C/W cycles were used.

The first 1,024 words of memory are a common data bank used to store program constants, subroutine linkages, and other common, frequently referenced data. The common data bank may be addressed directly from any place in memory. Memory modules are available in units of 8k, 16k, and 32k words. One or two memory modules [illegible] configured giving a range of 8k [illegible] maximum memory size. Character parity is generated and checked on all data transfers into and out of memory, giving the system a high degree of internal message integrity. Up to four active devices may be connected to the memory subsystem. An active device is defined as either a processor subsystem or a peripheral I/O subsystem.

The I/O subsystem provides interfaces for peripheral subsystems such as card readers and punches, printers, magnetic tapes, and drums or disks; for



In its minimum configuration the system can support two communications modules, but there is no back-up or failsoft capability. Any failure in any of the subsystems would bring the system to a halt.

high-speed synchronous channels (broadband channels operating at 50,000 bps and up); and for computer interface channels connecting one communications processor to another or to a host batch processor.

Each I/O subsystem accommodates up to four peripheral subsystems, four computer interface channels, eight high-speed synchronous channels, or combinations of any of the above. The higher data rates of the high-speed synchronous channels and the similarity of their discipline to that of the peripheral subsystems make it a logical choice to configure them in the I/O subsystem rather than try to fit them into the communications module design.

The processor is an interrupt-driven system containing four distinct levels of interrupt. The four levels are:

LEVEL	FUNCTIONS	PRIORITY
4	Error	Highest
3	Realtime	
2	Dual processor error	
1	I/O terminate	Lowest

The level four interrupt is associated with the built-in error detection circuitry which continuously monitors the entire system for environmental and system errors. Environmental errors include such things as excessive cabinet temperature and primary power failure. System errors include memory parity failure, lack of memory response, and dualed arithmetic unit failure. When an error is detected, an assigned bit is turned on in an error storage register, which records the error; a level-four interrupt is generated. The responding processor immediately leaves its current interrupt level, enters level four, reads in and examines the error storage register to determine which error has occurred, runs the appropriate diagnostic routine, and decides on a course of action.

If the failure makes continued system operation impossible, the processor will bring it to an orderly halt. Sometimes the failing module can be removed from service by the processor and the rest of the system allowed to continue; in this case the processor will send the appropriate message to the system's supervisory or control station. In those cases where continued operation is impossible, the error storage register can be read out manually by service personnel. This pinpoints the error and eliminates the need to run a time-consuming system diagnostic program.

The level-three interrupt provides the system software with a realtime clock which is software accessible and counts in intervals ranging from 10 μsec up to slightly over 500 msec. The level-three interrupts are used by the system's software executive to provide the various timing algorithms necessary for the execution of the communications channels macro commands and other time dependent activities.

If two processor subsystems are configured in a single system, and one fails, it will shut itself down, turn on the assigned bit in the error storage register, and generate a level-two interrupt in the alternate processor. The processor receiving the level-two interrupt will then initiate the required system recovery routine, execute a set of diagnostic commands in the halted processor to determine the exact cause of the failure, and send out an appropriate message to the network supervisory control station.

The level-one interrupt is used to provide the processor with an indication that a change of state from busy to not-busy has occurred on one or more high-speed synchronous channels, peripheral subsystems, or computer interfaces. This eliminates the need for the software to periodically "peek-out" to determine if a disk unit, for example, has finished its seek and is ready for the ensuing read or write command.

If the processor is not in any interrupt level it is in an implied level zero where all the spare time or nonrealtime processing is performed.

Four hardware macro commands are included in the instruction set to facilitate the interface between the processor(s) and the communications channels. These commands greatly reduce the processor overhead required to perform the realtime line and terminal disciplines needed by the communications

SYSTEM PERFORMANCE CHARACTERISTICS

The determination of the maximum continous I/O rate for large systems with heavy peripheral loads in a single processor, single I/O subsystem configuration is achieved as follows:

Since this is a two port configuration (one for the processor and one for the I/O subsystem), the total available memory time must be split between them. Logic in the memory allocation hardware will prevent it from allocating more than one memory cycle to any given port so long as there is a request for access up on another port. Assuming that the I/O subsystem has a device connected to it capable of accepting and using every possible access granted to it, the memory allocator will then alternate between the two ports.

The processor's average memory access time is about 1.3 μsec (since some portion of its cycles are R/R or C/W at 1.2 μsec and the remainder are RAW at 1.8μ sec) and the I/O subsystem uses a constant 1.2 μsec access time. Since there are two active devices requesting memory cycles, the 200 μsec logic delay times (included in the cycle time) can be overlapped giving a total of (1.3 + 1.2) —(2x.2) = 2.5 — .4 = 2.1 μsec elapsed time to grant each active device one memory access. This figure divided into the total available memory time gives the total number of memory access granted to each port every second:

$$\frac{1{,}000{,}000}{2.1} = \text{Approximately 476,000 accesses/port/second}$$

The average processor instruction requires approximately 2.2 accesses so the processor will execute about:

$$\frac{476{,}000}{2.2} = \text{216,363 instructions per second,}$$

While the I/O subsystem can maintain a continuous rate of:

476,000 × 3 = 1,428,000 6-bit characters per second or,
476,000 × 2 = 952,000 8-bit characters per second

To derive the system performance figures for the System without an I/O subsystem, the following calculations were performed.

The total available memory time is 1,000,000 μsec per second, since the system is memory bound.

Evaluation of the timing algorithm for the BSCN channel macro command shows that 106,400 μsec of memory time are required for:

1. Execution of the BSCN command
2. Execution of the QJMP command
3. Execution of the software line servicing routines

Since there are no I/O subsystems in the configuation there is no impact on memory time.

The remaining memory time of 893,600 μsec per sec. (89.36%) is available for the execution of instructions in the spare time processing mode. Based on an analysis of the system software, the average instruction execution time is 2.86 μsec. This number divided into 893,600 gives the average number of spare time instructions that can be performed each second.

$$\frac{893{,}600}{2.86} = \text{Approximately 312,000 instructions/second}$$

With 100 lines capable of receiving a maximum combined load of 1,000 characters per second.

$$\frac{312{,}000}{1{,}000} = \text{312 instructions per incoming character available for processing}$$

channels. The commands are:

- BSCN bit scan
- CSCN character scan
- QJMP queue jump
- FNDC find next active channel

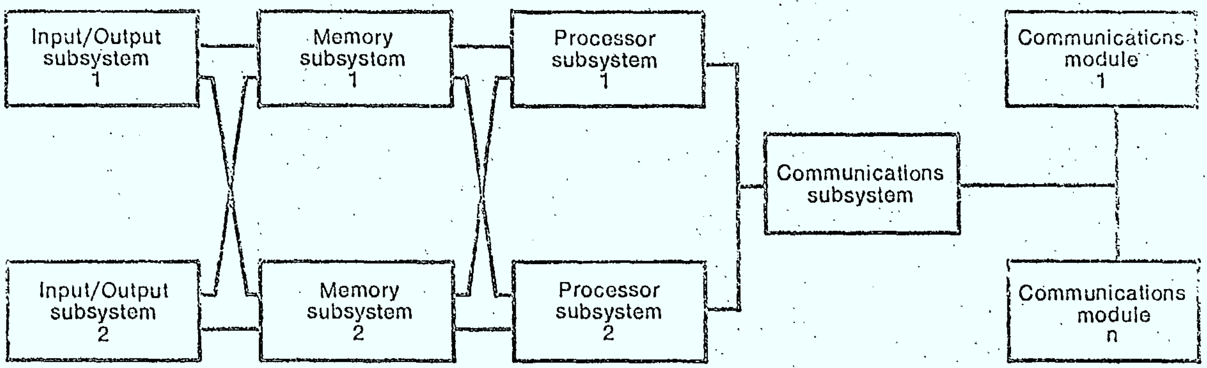
The bit scan command (BSCN) is executed once per bit time for all low-speed channels by the processor under control of the appropriate timing algorithm using the level-three (realtime) interrupt. The command assembles incoming, and disassembles outgoing, characters, bit by bit, in control words stored in the part of the memory that is dedicated to the communications channels.

The character scan command (CSCN), used for all medium-speed channels, differs from BSCN in that it is executed once per character time. It assembles and disassembles strings of incoming/outgoing characters under the control of the dedicated control words in memory.

The queue jump command (QJMP) is executed following the completion of either BSCN or CSCN. It services the queue of status words that these commands generate during their execution. Each status word indicates that software attention is required on a channel, and during QJMP, provide the processor with both the address of the channels requiring attention and a branch pointer to the specific subroutine performing the desired function.

During the execution of the BSCN and CSCN commands, only those channels requiring software attention will have status words generated and placed in the queue to be serviced by the QJMP command. This reduces processor overhead by eliminating the need to examine each channel with software to determine if it needs attention.

The find-next-active-channel (FNDC) command is used to provide a basic or skeleton macro capability for those channels that do not fit within the framework of the BSCN or CSCN commands. When executed with this class of channels FNDC determines the address of the next channel requiring software attention and supplies it to the processor which then



In the optimum configuration, with dual processors, memories communications and I/O subsystems, the system provides either full back-up capacity or a system operation that essentially doubles the throughput rate of a single system.

services the channel. It is also used following a level-one interrupt to identify the peripheral subsystem channel(s) that have changed state.

In its smallest configuration, the system includes a processor, memory (8k words), and one or two communications modules. This configuration could tolerate the failure of either communications module and still function. Failure of either the memory or the processor subsystems would bring the system down. Dualing the processor and memory subsystems with independent, identical processors, both capable of addressing any portion of or all the channels in the communications modules and the entire complement of memory (up to 65k words), results in a system which may be controlled by one processor with the other idling in a standby mode ready to take over in case of a failure, or they may both control a portion of the system with the understanding that if one fails the system runs at a reduced or degraded level.

Dualed memories provide either backup in case of system failure where the entire software package will fit in one memory, or will provide larger memory capacity, with the understanding that if one memory fails the system will run at a reduced level of performance. The memory addresses in this configuration are contiguous, i.e., one memory contains the low-order addresses and the other the high-order addresses. In the event of a hard or permanent failure of the module containing the low-order addresses, the responding processor will execute an instruction causing the remaining memory to assume the low-order addresses. This assures the restart and recovery routine that the low-order memory is operative.

If the instructions for processor I are all contained in memory I and similarly for processor II and memory II, the instruction execution rate and the total system throughput will approach twice that of a single processor, single memory system as the memories are in parallel operation.

An evaluation of the performance and throughput characteristics of the processor summed up in answers to the following questions:

- What is the total amount of memory time available per unit of time?
- What percent of this time is needed to handle the realtime functions imposed by the particular configuration of communications channels?
- What percent of this time is required to handle the I/O subsystem load, if any?

Assuming a single memory, single processor system with no I/O subsystem, and with 100 full-duplex lines running at a continuous 110 bps (10 characters), the system performance figures will provide 312 instructions per incoming character for processing through the system.

For those systems with heavy peripheral I/O requirements, such as large message switchers with disk storage subsystems, a determining of the maximum continuous I/O rate reveals that for a single memory, single processor, and single I/O configuration the performance figures will be:

- 216,363 instructions per second
- 1,428,000 six-bit characters per second or,
- 952,000 eight-bit characters per second.

Dualing this configuration and segmenting the software results in a system capable of executing in excess of 483,00 instructions per second while supporting a continuous I/O rate of almost three-million 6-bit or 2,000,000 8-bit characters per second.

The system provides a network processor capable of handling data communications needs ranging from relatively small remote concentrators to large batch processing front-end communications processing. Failsoft capability is optional for those applications requiring it which allow the relatively small user, without failsoft requirements to escape paying for unneeded capabilities. ✠

The numbers and calculations used in this article were derived from the actual performance data of a DATANET communications processor.

By the mid '70s, most computer installations will hook into some kind of communications network. DR. DIXON DOLL, private consultant, puts overall communications network design in perspective, and outlines the key parameters that influence system costs

Planning effective data communications systems

All the factors that make up a total data communications system must be considered early in the network's planning stages. It makes little sense to commit to specific equipment, transmission line, or network structure before all communication requirements of a particular application have been thoroughly studied. Rapidly advancing technology, increased demand for common carrier service, and a fast-changing regulatory climate complicate the data communications picture.

Planning a data communications system can be carried out in a systematic and objective way by working with a checklist of the important hardware and software parameters that affect decisions. Major breakdowns on this checklist would include:

- structure of the network
- types of remote terminals and related items
- transmission speeds of the communication links
- communications control and concentration methods
- system reliability needs

With these factors in mind, a planner can effectively deal with vendors and common carriers, and understand how to put together the best system for his particular needs.

Although many pitfalls and problems arise in planning an online telecommunications network, it need not be an unmanageable, awesome task. Essentially, the analyst or project leader must be able to understand and interpret the consequences of any potential layout. Some communications-based computer systems will contain few information sources and sinks. Here a combination of analytical techniques and intuition can go a long way. However, in the larger networks, the systems analyst may turn to computational tools provided by various hardware vendors and systems engineering firms specializing in network design.

System performance is a key criterion in the acceptance or rejection of alternative network layouts. Since performance will be closely related to the geography and traffic characteristics of the application, the first planning step is to estimate the following:

- number and location of remote terminals
- information flow patterns
- types of messages
- message volumes by terminal
- urgency of messages
- reserve capacity needed for future growth
- accuracy

These factors must be thoroughly assessed before making any major equipment decisions. In a sense, they are constants in the design problem since they don't depend on implementation.

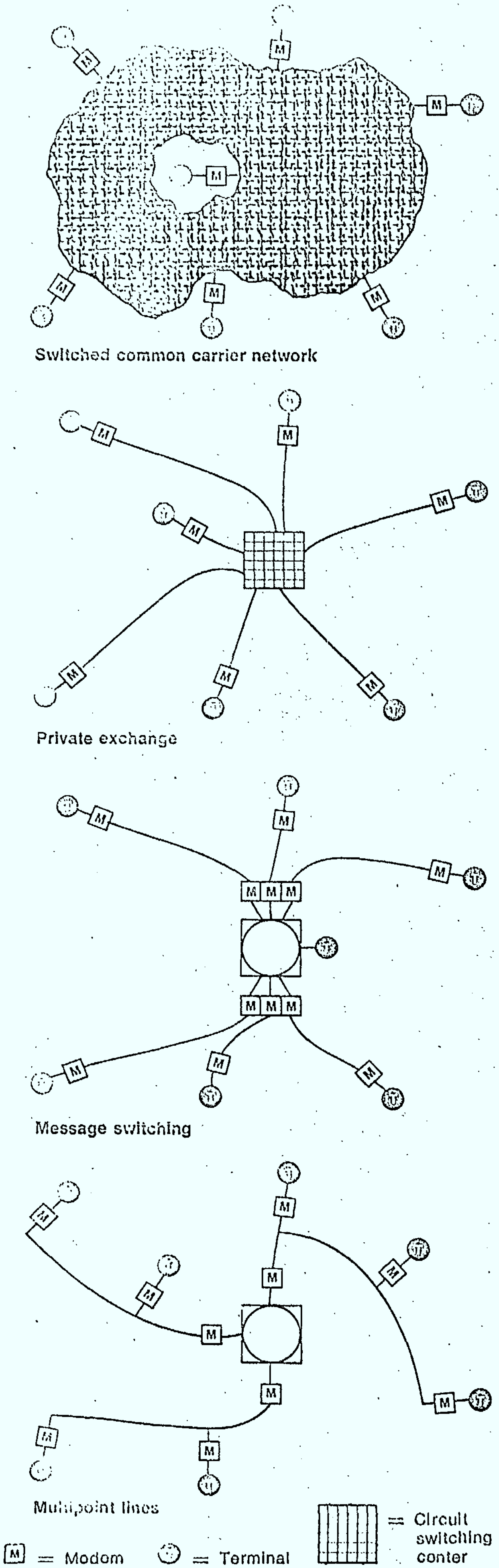
Geographic separation of remote terminals and a data processing center is what makes network design important. When all the terminals are clustered within a few miles of a computer site, most of the economic justification for sophisticated concentration schemes vanishes. Here transmission line costs will usually represent a negligible fraction of total system costs.

Where terminals and computers are widely scattered, the best network layout depends primarily on the traffic flow between data sources and sinks. Most applications have traffic patterns where many points communicate with many points (distributed) or with one central point (centralized) or have some combination of these patterns.

In any case, the analyst can select either switched or dedicated lines to interconnect points. Switched lines are often the best choice for applications with distributed traffic patterns. Here a common carrier network operates like the ordinary dial-telephone network. The common carrier equipment establishes a connection between any two stations, and maintains this connection only for the duration of a single call. The circuit is then disconnected and both stations are free to communicate with any other station in the network.

For centralized applications, the common carriers offer special tariffs to encourage use of their switched networks. For example, the Bell System's Wide Area Telecommunication Service (WATS) enables calls to be placed either into or out of (but not both) a central location, on either an unlimited or measured-time basis at established monthly rates.

Dedicated lines provide a permanent communication path between terminals, whether or not the terminals are active. For distributed traffic patterns, costs for a completely interconnected private line network are relatively high, caused by the large number of modems and long lines. Less expensive alter-



Switched common carrier network

Private exchange

Message switching

Multipoint lines

In distributed systems, data is transmitted to and from many similar terminals with no special flow pattern. Circuit switching message switching, multiplexing, and multipointing schemes cut line and equipment costs.

natives would be to use either a circuit switching or message switching center or multiplexer at some point within the network.

A circuit switching center is essentially a private exchange where electrical connections are automatically or manually made between incoming and outgoing circuits. A message switching center on the other hand, contains computational and storage capability. Here messages may be held and concentrated before being forwarded. The message switching center will require more modems than a circuit switcher, but often provides better features for increasing throughput and easing control.

Time and frequency division multiplexing are significant concepts that are widely used to cut communication costs (see June DPM, page 34 and pages 31 and 46 in this issue). Multiplexing applies, for example, whenever many low-speed terminals at a remote site must be connected to a computer. It reduces costs by dividing a transmission line's capacity into many low-speed subchannels whose effective cost per channel is substantially below that of a separate low-speed line connection. As an example, 24 Teletype terminals can be multiplexed to a 600-mile voice-grade line that leases for about $800 a month; yet one full duplex Teletype line of the same length leases for about $600 a month.

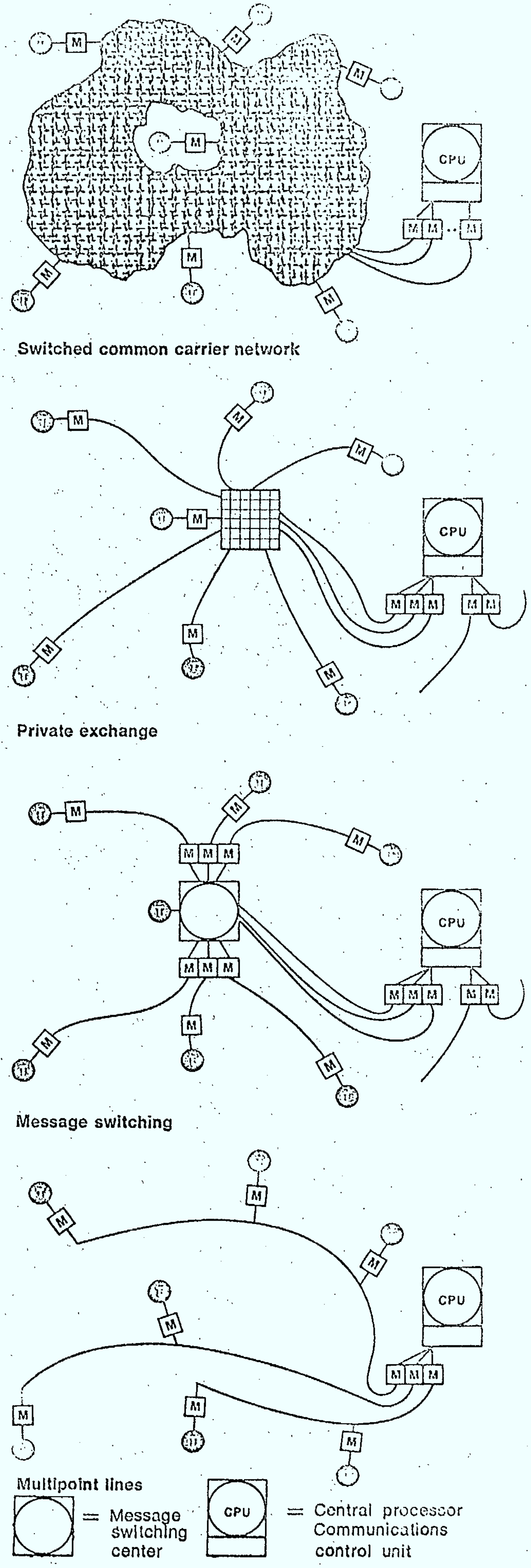
Another way to reduce costs is to connect two or more terminals to the same dedicated line, called a multipoint or multidrop line. These terminals must then take turns on the line, like users of a party-line telephone. Addressing and control requirements at the switching center are more detailed, but line and modem costs are reduced.

Terminals for sending and receiving data come in many forms and degrees of sophistication. Among these are the card reader/punch, Teletype terminal, printer, CRT display, batch station, and the programable data terminal that includes a small processor to do some local computation.

While the selection of terminal types depends primarily on the specific requirements for remote data entry to the computer, the terminal type does affect the performance and cost of the overall communications network. For example, a remote terminal with buffering and computational features permits more data processing to be done at the remote site. This frequently reduces online communication time with a consequent reduction in line charges. However, for a great many applications, a lower-cost, lower-speed teleprinter proves adequate.

Communication over lines more than a few hundred feet long usually requires a modem (or dataset) at each end of the line (see page 31). This modulator/demodulator imparts incoming digital data onto

In centralized systems, a computer acts as a major source and sink for information. Terminals communicate only indirectly with each other. Switching, multiplexing, and multipointing still reduce costs.



Switched common carrier network

Private exchange

Message switching

Multipoint lines

M
T
CPU
= Message switching center
= Central processor Communications control unit

a carrier signal suitable for the transmission lines. The modem at the receiving end removes the carrier from the line signal, restoring the original data.

Modems are provided by the common carriers and by a wide variety of independent companies; their operating speeds, costs, and functional capabilities cover a large range of applications. On a voice-grade telephone line, transmission speeds ranging up to 9,600 bps can be achieved. Most modems that transmit data at 2,400 bps or faster also require special circuit conditioning to widen the usable bandwidth of the telephone line.

A terminal usually has a fixed, maximum operating speed so that the companion modem must have a sufficiently fast clocking rate. For unbuffered terminals like Teletypes, the modem will always operate at the same speed as the terminal. Thus the terminal dictates the transmission speed in these cases. However, buffered terminals give the system analyst more latitude in selecting the line speed.

A buffer is basically a data memory, whose storage capacity may be a word, a line, or—in the case of tape cassettes—may range up to hours of accumulated data. Since the buffer permits transmission over the communication line at a faster rate than the data is entered or received by the terminal operator, it offers economics resulting from the use of fast lines for short intervals.

Regardless of whether terminal equipment dictates transmission speed, the planner will usually have to choose from three basic grades of transmission service for communication lines: low speed, voice grade, and broadband.

Low speed transmission ordinarily includes all speeds less than 600 bps, such as Teletype terminals and most other typewriter-like devices. The communication lines of the ordinary telephone network are referred to as voice-grade lines; they can be operated at a wide range of speeds, depending on the modem used. Voice-grade lines are usually used with communications equipment such as multiplexers, message switchers, and buffered terminals. The standard speeds for voice-grade operation are 600 X N bps, where N is an integer from 1 to 16. Broadband speeds go higher than 9,600 bps. Examples are AT&T's private line services known as Telpak A, B, C, and D, having capacities of 12, 24, 60, and 240 voice-grade lines.

Private microwave systems are opening new applications for broadband transmissions; common-carrier broadband services are economical only for large corporations with high communications budgets and traffic volumes. Whenever justifiable, broadband suits computer-to-computer data transfers well since typical internal speeds of most medium and large mainframes far exceed the current practical speed limits of the voice-grade line.

For communications control, either contention or polling procedures may be used to monitor access to the transmission lines in the network. With contention control, terminals request access to a line. If the line is free, the connection is made; if not, the terminal must wait. The communications control unit must build up a list of transmission requests for each line, and process these requests in some predetermined order, such as a first-come, first-served basis.

In polling, the communications controller asks each remote terminal if it has anything to send, making these queries in a predetermined sequence. Polling is generally performed under control of vendor software, provided as part of the overall operating system. Some software packages for polling are IBM's QTAM (Queued Telecommunication Access Method), or BTAM (Basic Telecommunications Access Method), and RCA's MCS (Multichannel Communications System). The software package maintains a list of valid terminal addresses; the order of entries in this list determines the sequence in which queries are made.

Polling is more suitable in large networks where tight control of line usage is essential for efficient performance. It generally yields better performance than contention does, but requires costlier hardware and software. Polling is also better suited for applications where transmitted messages have different levels of priority. If a terminal tends to submit many high priority messages, then the polling procedure may be designed to query it more frequently than the other terminals. The query sequence, often called the polling list, can be readily modified under program control.

A disadvantage of polling is that the remote terminal can't send data at any time, but only when asked. For most networks, some combination of polling and contention works best. RCA claims its MCS package, mentioned earlier, can combine the two procedures for different lines in the same network.

The exchange of information between remote terminals and the central computer calls for much code conversion, error checking, message editing, message-block assembling, multiplexing, and related processing operations. These functions may be done by the central computer itself, or by a special communications control unit, located either near the CPU or at a remote site. Assigning these functions to a special communications control unit frees the central computer from constant interruptions, enabling the processor to devote more time to actual computation.

Examples of these special communications control units are the IBM 2701-2-3 series, the Collins C-System, the GE DATANET series, the Comcet 60, and the Sanders Associates Sandac 200 (see page 51). Recent wide-spread availability of relatively inexpensive minicomputers has led to their use as communications control units (page 43).

In a broad sense, reliability includes line transmission errors caused by circuit noise and by failure of system components, such as lines, modems, and concentrators.

Transmission errors from circuit noise inevitably occur when sending data over almost any communication link. System-wide impact of such errors can be minimized. Two common ways to deal with errors are to ignore them, leaving correction up to a human console operator, or to implement block codes and parity schemes that automatically detect errors. With these latter methods, errors that crop up at a receiving station initiate a control signal that travels back to the sending terminal and requests a retransmission. On the other hand, if no error appears, the next block is transmitted. These methods for error detection and retransmission require extra control logic and buffering hardware in the terminal. On noisy lines retransmissions lead to much overhead, reducing the net information transfer rate significantly.

More powerful codes than those solely for error detection are available. These codes can find and *correct* errors in data blocks without the need for retransmission.

A common misconception is that throughput over a communications link must *always* be sacrificed for these extra error-protection bits. In some applications, however, the continuous sending of from 5% to 50% redundant bits will improve the effective error rate so substantially that net improvements in message throughput actually result.

In any event, the systems analyst must carefully check modem and terminal equipment for methods that guard against transmission errors. Once he selects a particular terminal, he usually has no choice on error control techniques. These schemes are embedded in most terminals. Some latitude is afforded by modem manufacturers who offer families of devices with varying degrees of susceptibility to noise.

The planner-analyst will want to consider the least expensive modem that has an acceptable error rate for the desired transmission speed on a given link. Actual throughput will depend on:

- modem clocking or data rate
- lengths of message blocks
- error rate on communication line
- control signaling time between message blocks
- code used by the terminals

To protect against prolonged outages of system components, the analyst will have to look at alternative transmission paths, backup equipment, and possibly temporary changes in the entire network structure. The flexibility afforded by ordinary switched telephone networks makes them an attractive alternative to fall back on during prolonged outages.

Changing to the switched network may temporarily degrade system-wide throughput. It total integrity and continuity of system operation are paramount, an additional option is to purchase or lease extra equipment such as modems, communications links, and concentrators, to be switched on when the primary devices fail.

General References

TEXTBOOK REFERENCES

1. R. Watson — Time Sharing System Design Concepts
McGraw Hill 1970 ($12.50)

2. Harry Katzan Jr. — Advanced Programming
Van Nostrand Reinhold 1970 ($15.00)

3. J. Martin — Teleprocessing Network Organization
Prentice Hall 1970

4. DATAMATION — A Catalogue of EDP Products and Services (1971)
Available from Datamation, 1301 South Grove Ave.,
Barrington, Illinois 60010. ($35.00)

5. AUERBACH — On Time Sharing (1967)
available from Auerbach Info. Inc.,
Philadelphia, Pa 19109. ($14.00)

6. James Ziegler — Time Sharing Data Processing Systems
Prentice Hall 1967 ($13.00)

7. Douglas Parkhill — The Challenge of the Computer Utility,
Addison Wesley. ($8.00)