A SIMULATION STUDY OF A LOCALISED COMPUTER NETWORK

Thesis submitted to the

UNIVERSITY OF ASTON IN BIRMINGHAM

for the degree of

DOCTOR OF PHILOSOPHY

Zbigniew Ziemski

OCTOBER 1977

ZBIGNIEW ZIEMSKI PH.D. 1977

A SIMULATION STUDY OF A LOCALISED COMPUTER NETWORK

SYNOPSIS

An investigation is carried out into the design of a small local computer network for eventual implementation on the University of Aston campus. Microprocessors are investigated as a possible choice for use as a node controller for reasons of cost and reliability. Since the network will be local, high speed lines of megabit order are proposed.

After an introduction to several well known networks, various aspects of networks are discussed including packet switching, functions of a node and host-node protocol. Chapter three develops the network philosophy with an introduction to microprocessors. Various organisations of microprocessors into multicomputer and multiprocessor systems are discussed, together with methods of achieving reliable computing. Chapter four presents the simulation model and its implementation as a computer program.

The major modelling effort is to study the behaviour of messages queueing for access to the network and the message delay experienced on the network. Use is made of spectral analysis to determine the sampling frequency while Exponentially Weighted Moving Averages are used for data smoothing.

Keywords: simulation, packet-switching, network,
microprocessor, fail-soft computing

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. M. Walker for his guidance and supervision throughout the period of this research. I am indebted to Mr. K. Bowcock for making available the facilities required to carry out this research. I would like to thank Mr. J. Hollingworth for the discussions with him and for his useful suggestions.

Finally, but not least, thanks are due to Miss S. Gaufroid for her very patient understanding and encouragement throughout the period of research and for typing the thesis under very difficult conditions.

CONTENTS

Page

Chapter Three: Network Philosophy

Chapter Four: Network Model and Simulation Program

Chapter Five: Effects of Parameter Changes in the Node

LIST OF FIGURES

Page

Standard Network

<u>Standard Network - 1 processor, 2 memory units per node</u>

<u>Standard Network - 1 μsecond memory/processor</u>

Standard Network - 128 word memory module

Standard Network - 3 hosts/node

Standard Network - 256 word packet

Standard Network - mean short message = 2 packets
- mean long message = 20 packets

Standard Network - short/long message in ratio 9:1

Standard Network - Mean number of generating hosts = 2

CHAPTER ONE

INTRODUCTION TO COMPUTER NETWORKS

Computer networks are the product of the seventies and will undoubtedly take over from the time-sharing industry of the previous decade. Their effect on society and the economy will be more profound than any other network developed to date.

A computer network can be defined as "an interconnection of dependent or independent computer systems which communicate with each other in order to share certain resources such as programs and/or data; load sharing; and reliability reasons."

Functioning computer networks have been in existence for several years since the early sixties. They include CYBERNET, MERIT and OCTOPUS but perhaps the most sophisticated and ambitious computer network in existence is the ARPA network.

A network can be divided into two parts: one part consisting of the computers which provide the computational services of the network - the "HOSTS"; and the other part consisting of those computers which service the communication needs of the network - the "NODES."

Computer networks are set up as a message service to enable any computer on the network to submit a message destined for another computer in such a way that the message will be delivered quickly and correctly. As the two computers are communicating there will be messages going back and forth similar to the types of messages between a user console and a computer on a time-shared system. It is in effect an ultra high speed postal system with little storage or buffering capability.

A typical usage of a network might be the preparation of a program on one computer, transmitting it to another computer for processing and finally transmitting the results back to the first computer for output on a line printer.

Within a network two types of message-switching may occur: circuit-switching and packet-switching. Circuit-switching is the classic approach where a complete path is established between the two parties for as long as they wish to communicate and is comparable to the telephone system.

Packet-switching is a method of working similar to the store-and-forward technique used for telegraph message-switching in which the communication called a message is broken up into smaller units called "packets." Packet-switching is particularly suitable for data transfers involving intermittent short bursts of data with relatively long pauses between bursts. Packet-switching will be discussed in greater detail in Chapter 2.

1.1 Types of Networks

Centralised Networks are often called "star" networks because the various computers are interconnected through a central unit as shown in Figure 1.1.

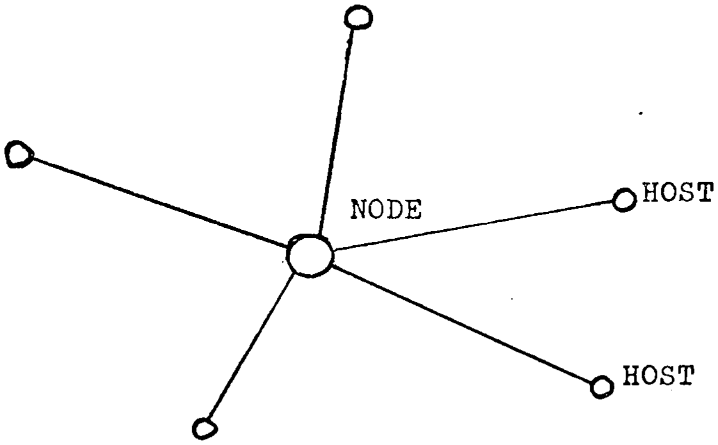


Figure 1.1 Centralised or star network

Figure 1.1 shows a centralised network as a set of point-to-point connections. An alternative structure is a multipoint or multidrop line where several terminals or computers may use one dedicated line.

This type of network requires that the capabilities of the central controlling unit far surpass those of the peripheral units or it requires that the central computer does little more than switch the various messages between the other computers connected to it. As may be seen, the major disadvantage of a centralised network is the vulnerability of the network to the failure of the central computer i.e. should the central computer incur a fault the entire network ceases to function.

A distributed (or decentralised) network overcomes the disadvantage of the centralised network by having no central computer. The responsibility for communication is shared among all the nodes in the system as shown in Figure 1.2.

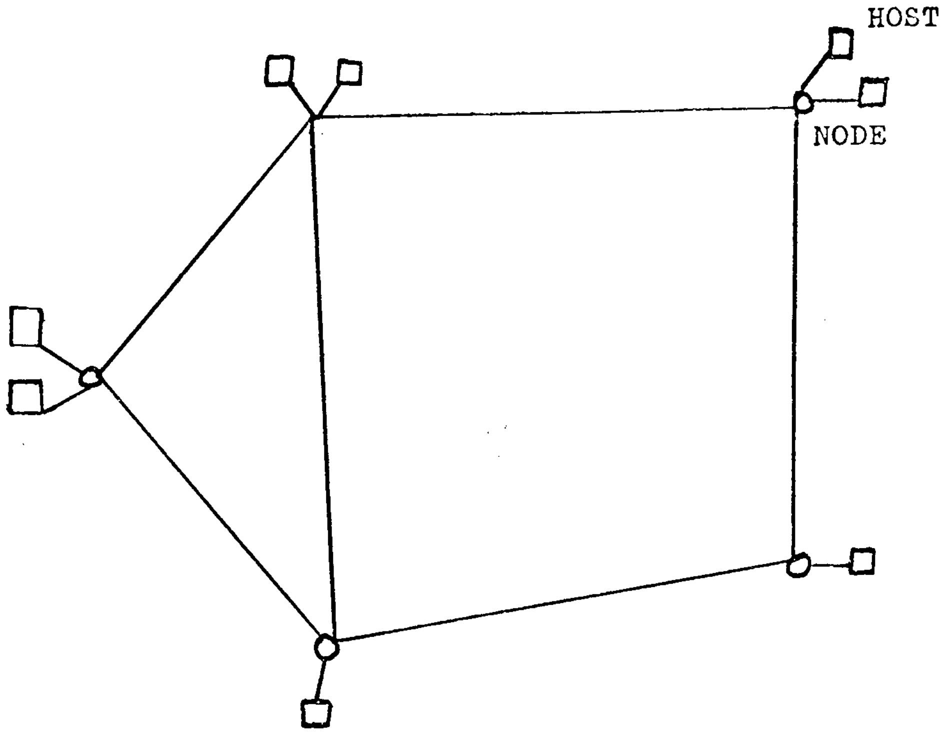


Figure 1.2 Distributed Network

A message may have to pass through several nodes before reaching its final destination. The network is made more reliable by ensuring that each node is connected to at least two others. In the event of a connecting link failing communication may always continue along an alternative path. Even if a node fails, unaffected nodes can continue to function as long as the link remains operable. ARPA is a distributed network but is not fully connected as the cost would be prohibitive, whereas MERIT is an example of a fully connected distributed network.

In a ring network, a ring or loop-type network is formed by a set of nodes. Any terminal or host computer wishing access is connected to one of the nodes as shown in Figure 1.3.

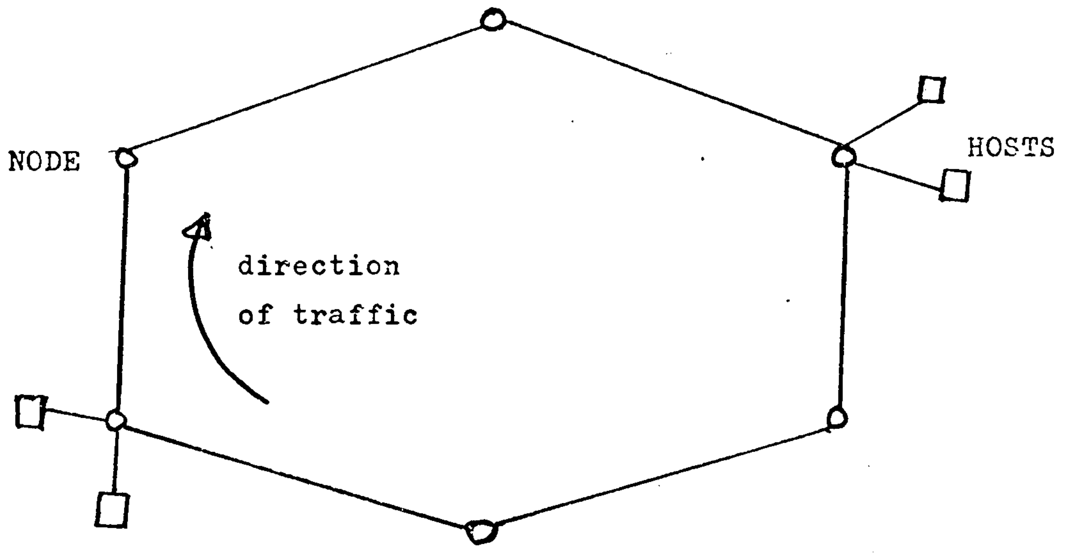


Figure 1.3 Ring Network

The nodes bridge their input and output lines with a shift register. The channel capacity of the ring is multiplexed into a series of time slots e.g. a 20 kilobits/second channel is divided into 20 slots each of 1000 bits. The time slots all flow in the same direction from node to node. All incoming messages are then put into a free slot as it comes around. A ring-switched network may consist of several rings. Neighbouring rings would be interconnected by a switching processor. Although ring networks are easy to design and cheap to build they have low reliability. Hayes and Sherman [1] discuss ring networks in greater depth.

1.2 Existing Networks

1.2.1 ARPA

The Advanced Research Project Agency (ARPA) funded network is probably the most sophisticated network in existence [2-15]. Its primary goal is to make available the resources of the network to all users. Other design aims of ARPA are:

1) A communications cost of less than 30 cents per 1000 packets (≙ 1 megabits).

2) Average packet delays under 0.2 seconds through the network.

3) Capacity for expansion to 64 IMP's without major hardware or software redesign.

4) Average total throughput capability of 10-15 kilobits/second for all hosts at an IMP.

5) Peak throughput capability of 85 kilobits/second per pair of IMP's in an otherwise unloaded network.

6) Transparent communications with maximum message size of approximately 8000 bits and error rates of one bit in 10^{12} or less.

7) Total network traffic 700-800 kilobits/second for a 20 IMP network.

ARPA is a distributed network of heterogeneous computers and operating systems. Local computers (HOSTS) are linked to the network via Interface Message Processors which are generally called IMP's. IMP's are modified Honeywell DDP-516's with 12 k memory - 6 k memory is required by software support, the remaining memory is used for message and queue storage. Each node can store approximately 77 packets. Terminals can use the network directly via Terminal Interface Processors (TIP's) [7,11]. Figure 1.4 shows the general layout of a section of ARPA. The network provides store-and-forward communications. Internodel communications are provided via 50 kilobit full duplex leased lines. Reliability has been achieved through efficient error-checking of each packet and the provision of two separate links from each node to

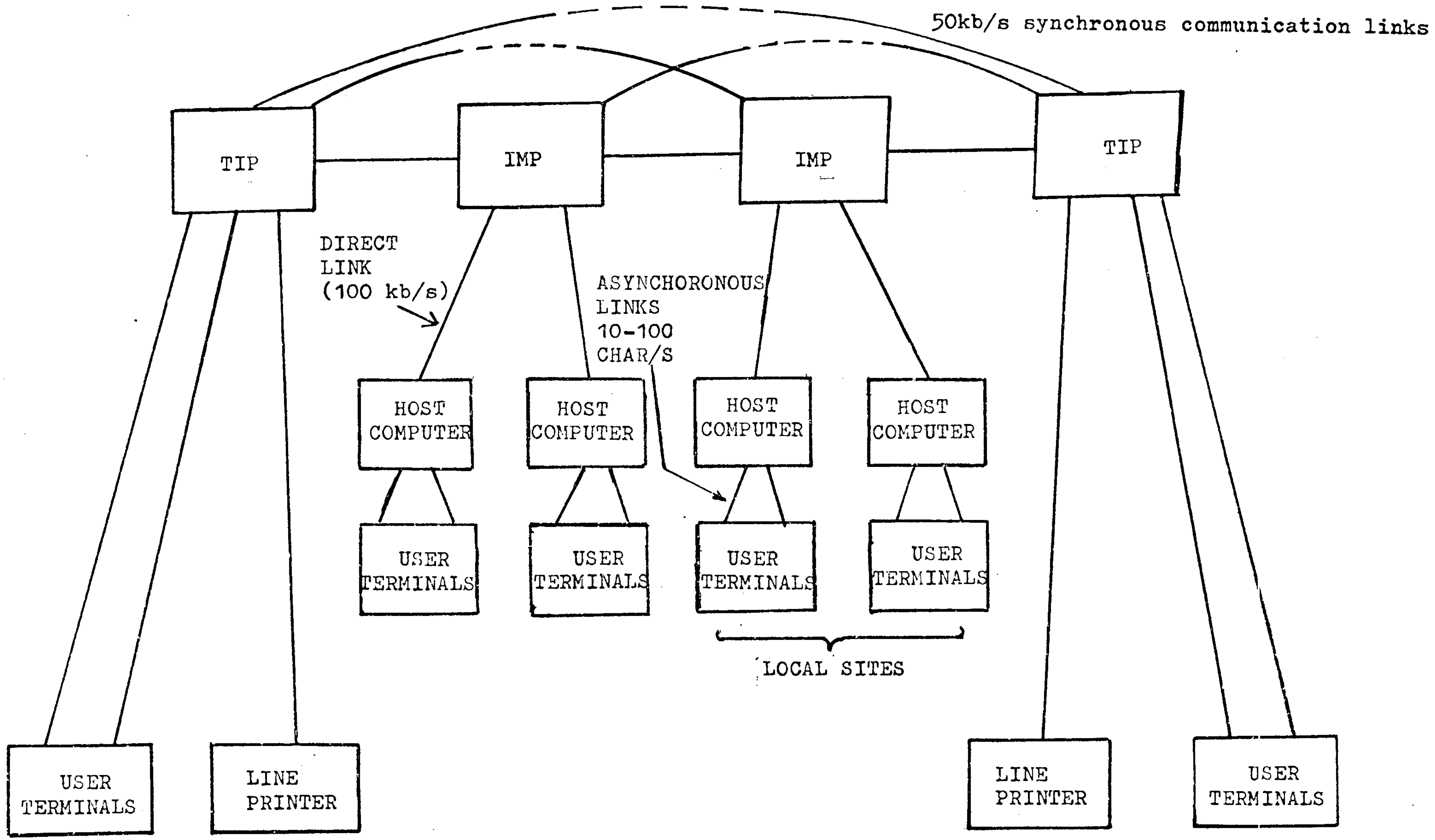


Figure 1.4 Functional Units of ARPA Network

protect against total link failure.

Each Host computer has a Network Control Program (NCP) whose function is to establish links, terminate links and control the flow of traffic.

When an IMP receives a message from a Host it breaks it up into "packets." Packets have a maximum size of 1008 bits, and each message consists of up to a maximum size of 8095 bits. Packets are then independently routed to the destination IMP where space has been reserved for reassembly before transmission to the receiving host.

As each packet is passed from IMP to IMP to reach its destination, the sending IMP retains a copy until an error check is carried out at the receiving IMP and a positive acknowledgement is sent back. On receipt of a packet an IMP must determine whether a packet has reached its destination or whether it needs to be transmitted further by checking the destination address.

Each IMP has the facilities for detecting communication failures, transmitting idling packets during the absence of normal traffic, and gathering performance statistics.

ARPA currently has over 40 nodes and over 80 hosts spread across America to Hawaii (via satellite link) and a few locations in Europe as shown in Figure 1.5. The hosts range from PDP-11's to the ILLIAC 4 which are incompatible both in software and hardware.

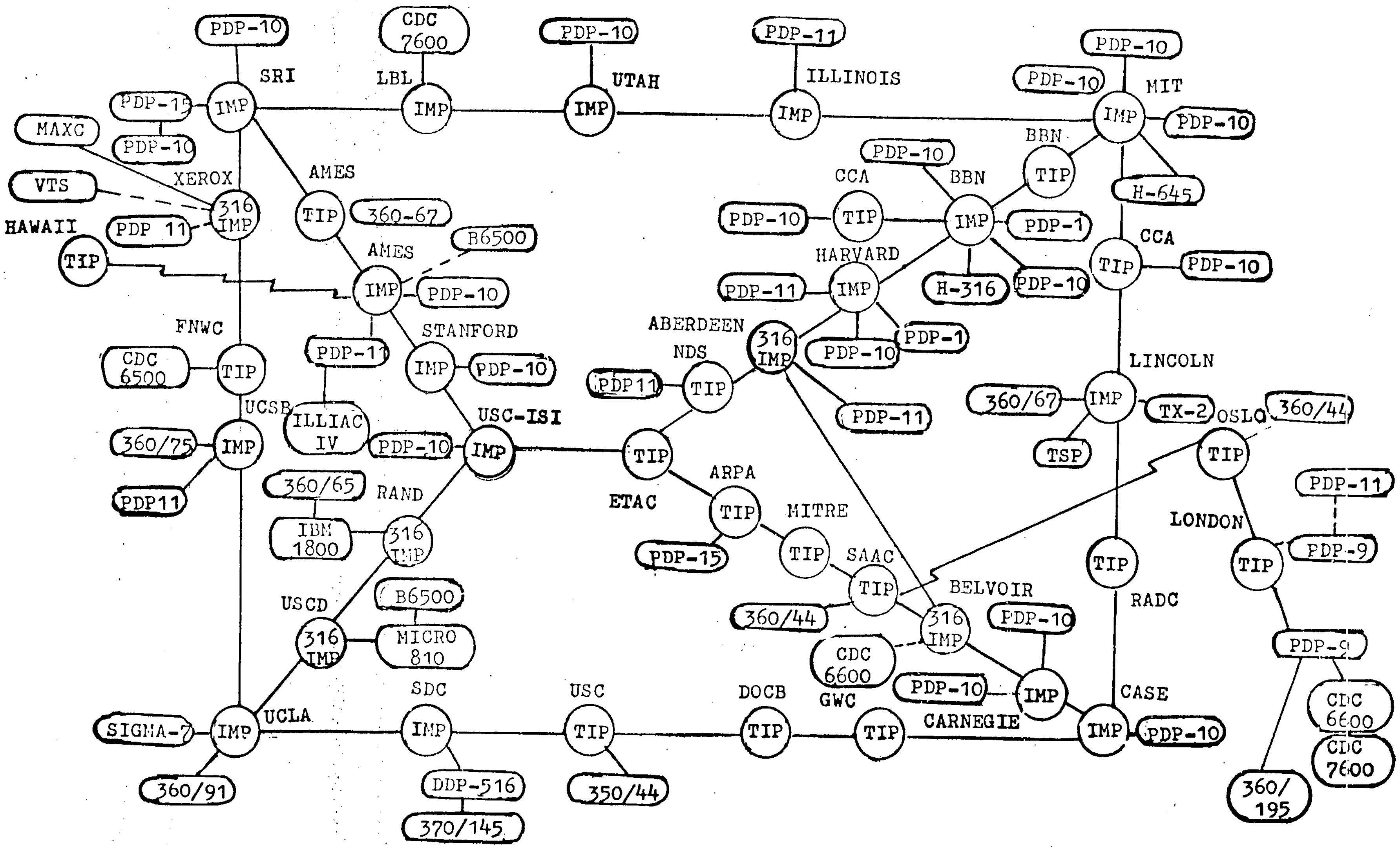


Figure 1.5 ARPA network - logical map of hosts and nodes, July 1973

1.2.2 Cybernet

CDC's CYBERNET [2,16], although not as sophisticated as ARPA needs to be mentioned since it was one of the first commercial networks offering its facilities to the public. It was built to connect CDC's existing computer centres to provide the following advantages:

1) Better reliability, users have access to an alternative computer in the event of a breakdown.

2) Greater throughput by allowing local work to be transferred to a less busy site. It also allows better load balancing with machines in different time-zones.

3) Better manpower utilization; allowing users to access one anothers programs and data bases.

4) Enhanced computer utilization through users choosing the best resources rather than local ones for the task in hand.

Cybernet is a distributed store-and-forward network composed of heterogeneous computers, mainly CDC 6600's and CDC 3300 linked by wideband lines across the U.S.A. The CDC 6600's, which constitute the primary computing element, are referred to as "centroids"; while the CDC 3300's serve as front end loaders and concentrators to the 6600's and are referred to as "nodes." Interactive and remote job submissions are supported by terminals and satellite computers. Cybernet communications employ switched,leased and satellite communications.

However, Cybernet cannot reconfigure itself and relies essentially on hand-established terminal to computer, and computer to computer connections. Although alternative paths do exist, line failure in general necessitates human intervention.

1.2.3 DCS

The Distributed Computer System (DCS) shown in Figure 1.6 is an experimental computer network being developed and constructed at the University of California at Irvine [2,17,18]. Its aims are: low cost, reliability, expansion capability and modest software development. However, the primary aim is to investigate the nature of distributed architecture in general. It is intended primarily to service mini to midi computers.

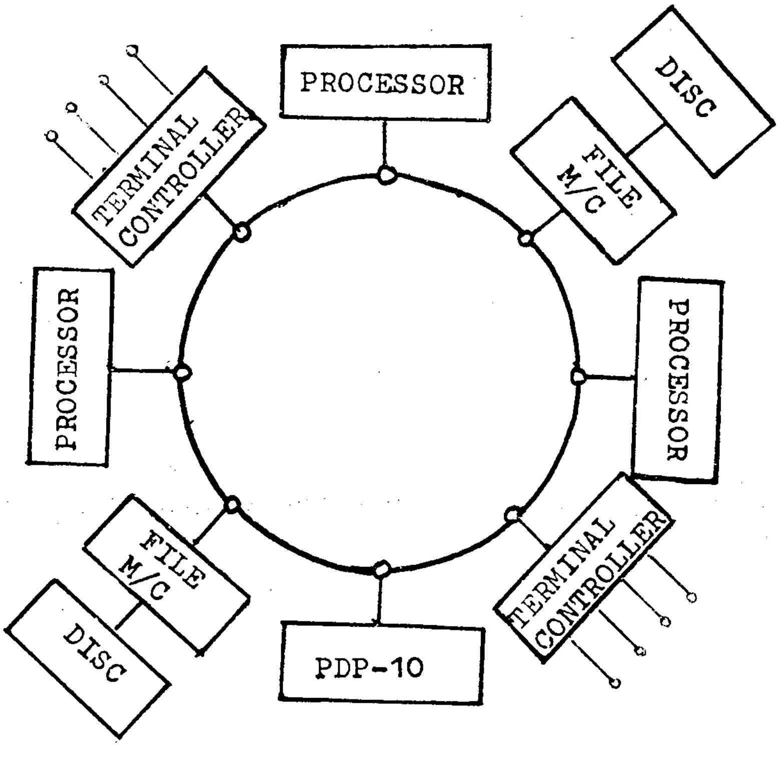


Figure 1.6 DCS network

The communications architecture is based on the Bell System TI technology (Pulse Coded Modulation on wideband of order 1.5-6.2

megabits) and fixed-length messages. The coaxical cable will initially give 2 megabits per second digital transmission but could be increased to the 6 megabit limit. Host computers are connected to the network via "ring interfaces" not computer nodes. DCS supports three classes of ring interfaces:

1) Computer support which could be a front end machine.
2) Terminal support.
3) Network of ring support.

Messages are sent to a process name and not to a real processor address. The process is identified by a general classification such as language file etc. Within each class are subclasses such as Basic, Fortran, PL/1 etc.

1.2.4 MERIT

The Michigan Education Research Information Triad (MERIT) is a tripartite effort between the three largest universities in Michigan: Michigan State University, University of Michigan, and Wayne State University [2,19,21,22]. Its objective is similar to ARPA: namely resource-sharing.

Merit is a distributed network consisting of three nodes having three heterogeneous hosts which are connected to the network via Communications Computers (CC). The CC is a modified DEC PDP-11/20 with 16k 16 bit words of memory. The CC acts as a store-and-forward system enabling an alternative path to be chosen should a line fail. The Communications Computer Cperating

System (CCOS) requires 8k of memory, the remaining memory being used for message storing. Figure 1.7 shows the MERIT network layout.

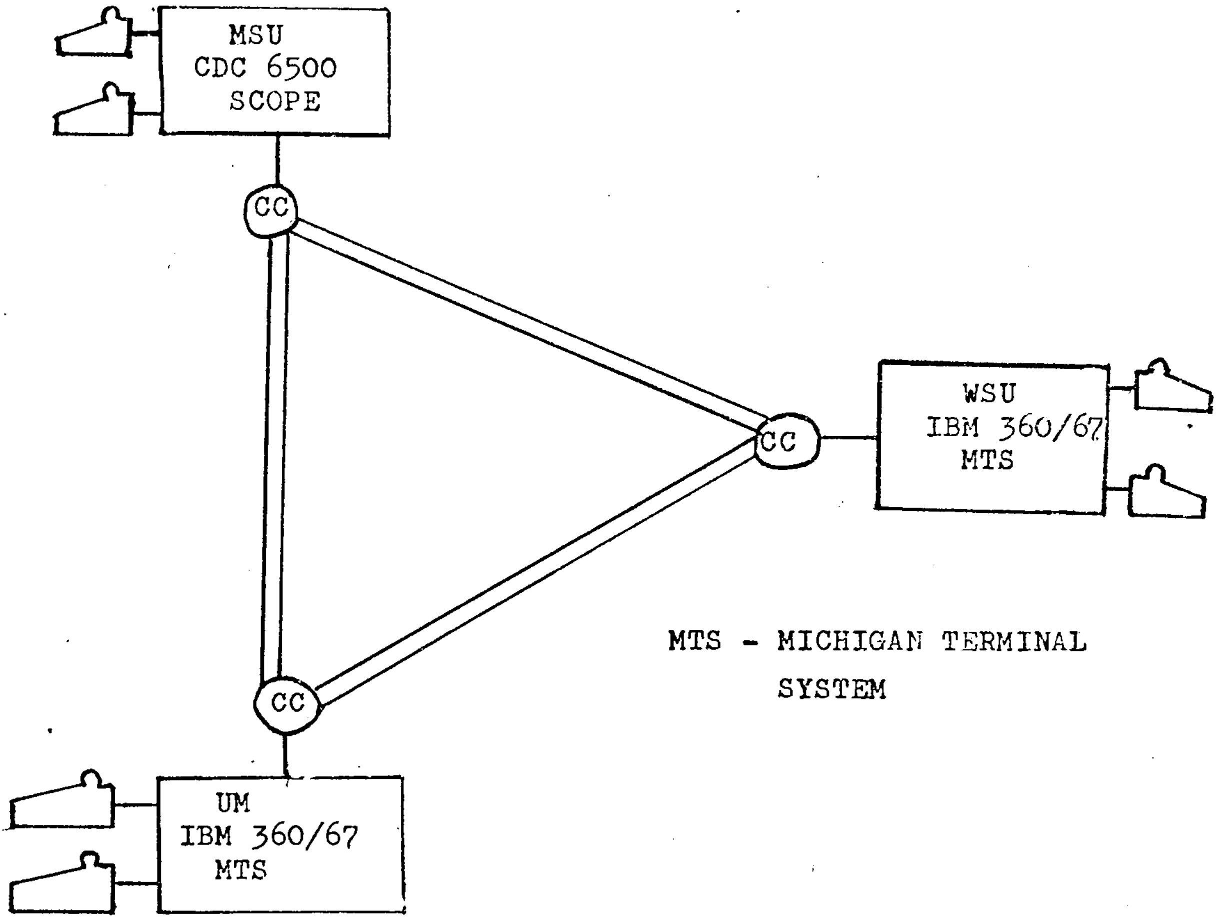


Figure 1.7 Merit Computer Network

Inter CC communications is provided initially by a group of 2000bps voice-grade lines for reasons of economy, low load and by the fact that they exist. As with ARPA after traversing each path the message is error-checked and an acknowledgement is sent for an error-free receipt.

The host/cc interface is capable of independently transmitting a variable-length data record to (from) the CC memory from (to) the host computer, performing any memory alignment operation required by the different word configurations of the two processors. The host software in addition is simplified by the interface providing a multi-address facility permitting the host to treat the CC as several peripheral devices. Thus each user/task requesting use of the communications resource is allocated a dedicated pseudo-device.

1.2.5 Octopus

Octopus is a heterogeneous network developed at the Berkeley Laboratory of the University of California which became operational in 1964 [2,22]. The primary computer power is provided by two CDC 6600's, two CDC 7600's and a CDC STAR. All these "workers" operate as time-shared facilities. The network supports a centralised data base and a large variety of I/O devices which give the user a single access point to all computers.

The workers are interconnected via 12 megabit hard-wired lines and the communication system utilizes a store-and-forward protocol. The topology of OCTOPUS is shown in Figure 1.8. The system is best viewed as two independent, superimposed networks:

1) File Transport Subnet which consists of a centralised network of the worker computers connected to the Transport Control Computer (a duplexed PDP-10) and the central memory system (disc, data cell, and photo store).

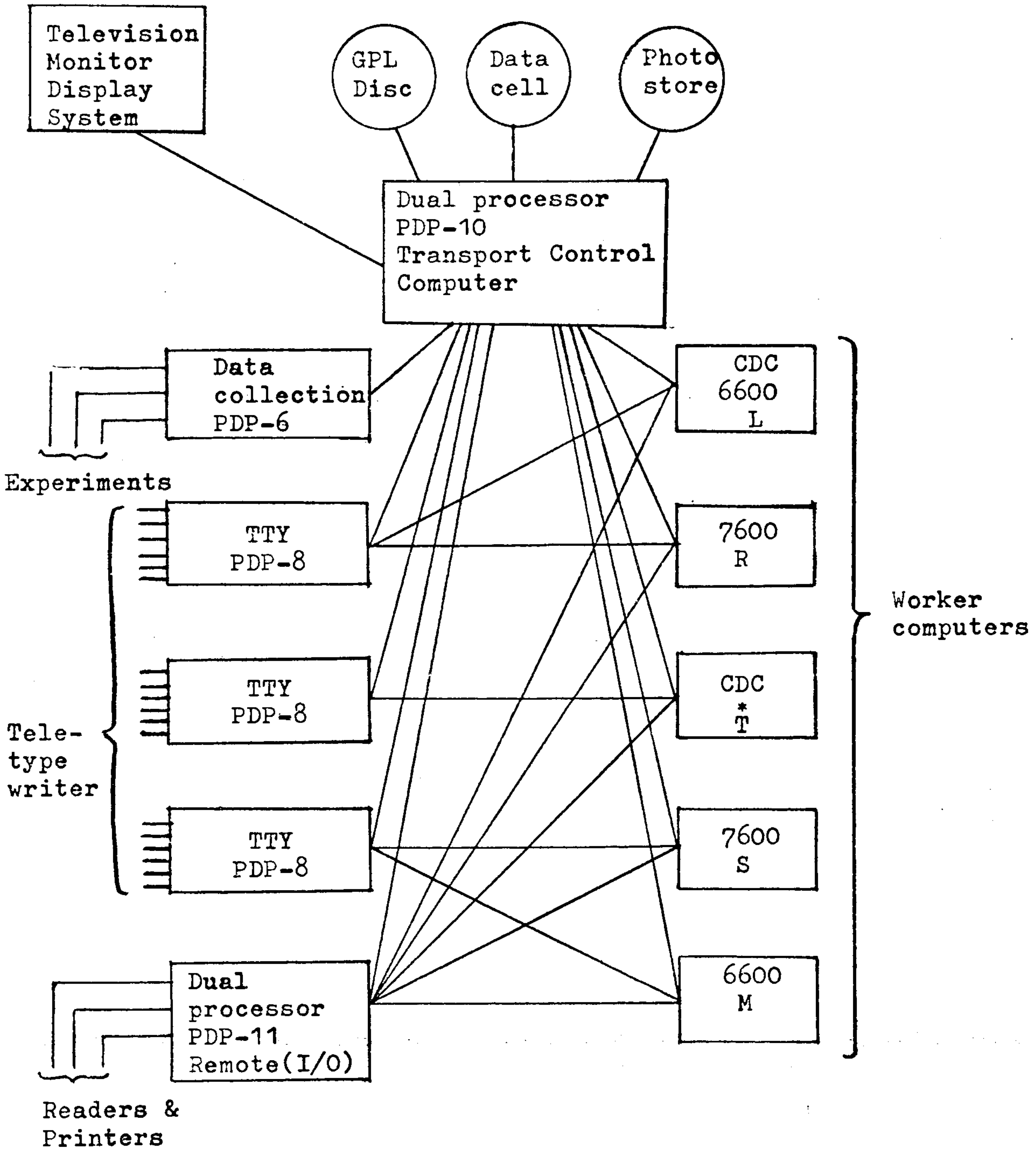


Figure 1.8 OCTOPUS network

2) Teletype subnet which is a distributed network consisting of the worker computers, 3 PDP8's (each of which can service up to 128 T/T's and the Transport Control Computer.

A graphic display capability with distributed monitors is provided by the Television Monitor Display System (TMDS). A third subnet exists to support remote I/O via a duplexed PDP-11.

1.2.6 TSS

The Time Sharing System (TSS) network is a distributed network of homogeneous computers developed in 1967 between IBM and some of its 360/67 customers [2,23]. The 360/67's use the IBM TSS/360 operating system. Some of the 360/67's operate as nodes for a local network consisting of 360's which appear as devices to the network not hosts.

The nodes are interconnected by 2000bps voice-grade lines and 40,800 bps leased lines. The voice-grade lines are interfaced to the IBM 360/67 by IBM 2701's and 2703's.

The communication software operates as an ordinary user program resident in the host computers which has to provide all programs such as store-and-forward, error-checking etc. This approach has the advantage of minimising extensive changes to the TSS/360 operating System. However, the penalty paid is that the communication software has to contend for resources on the same basis as any other program. This results in the communications equipment not being used to maximum advantage.

Users access the network via CAM (Computer Access Method) which is a specially developed set of procedure calls. A CAM request will check on whether a connection exists to the destination computer; if not, one is established. Messages, which may be up to 1k bytes long, are error-checked and acknowledged on receipt or a retransmission request is made.

It is primarily a research network to investigate the advantages and disadvantages of general purpose networks. Load-sharing, remote service and dynamic file access are some of the features provided. Using homogeneous computers eases implementation problems since problems of different command languages, data structures, operating systems and machines are avoided.

1.2.7 TUCC

The Triangle Universities Computation Centre (TUCC) has been operative since 1966. It is another joint venture between three major North Carolina Universities: Duke University, Noth Carolina State University and the University of North Carolina although many schools and colleges also enjoy the benefits of TUCC [2,24]. It is a simple central network supporting homogeneous IBM 360/40's and 360/75. The 360's are multiprogrammed for local batch work and to support the telecommunication requirements of the network. The network central control is carried out by an IBM 360/75 which was replaced by an IBM 370/165 in 1972 to cope with the work load.

The three primary goals that the network had to satisfy were:

1) To provide economically adequate computing facilities to each institution.
2) To minimise system programming personnel.
3) To encourage greater cooperation between the three universities in the exchange of systems, programs and ideas.

The TUCC computer connects with the local sites via single leased wide band lines of 40,800 bps half duplex operation. The network has resulted in substantially greater computer facilities through economies of scale.

The networks that have been discussed show a wide range of architecture and approach that have employed in network design. There are many other networks including ACCNET [25], DATRAN [26], NPL [27] and the British Post Office's Experimental Packet-Switching Service [28]. Figure 1.9 shows a summary of existing network features.

Computer Networks may be justified for any reason or combination of reasons as given below:

1) Load balancing

2) Avoidance of data duplication

3) Avoidance of software duplication

4) Flexibility

5) Simplification of file backup

6) Ability to combine facilities

7) Conversion simplification

8) Enhancement of file security

9) Decreased system costs

10) Improved computer efficiency.

1.3 Modelling Methods

There are two main techniques that may be used to analyse and evaluate the effects of proposed changes on network performance. However, all methods including simulation have their limitation.

The first method involves using Stochastic queuing theory [43,44] requiring the derivation of a series of equations describing the network being examined. The system being studied consists of a continuous flow of information or items which are counted in aggregate rather than as individual items. Even for simple networks the resultant models tend to become extremely complex and rather stringent simplifying assumptions must be made in order to find solutions. A number of queueing models have been devised to analyse the characteristics of networks [29,30].

The alternative to queueing theory which is also widely

	ARPA	CYBERNET	DCS	MERIT	OCTOPUS	TSS	TUCC
ORGANISATION	DIST-RIBUTED	DIST-RIBUTED	DIST-RIBUTED	DIST-RIBUTED	MIXED	DIST-RIBUTED	CENTRAL
COMPOSITION	HETERO-GENEOUS	HETERO-GENEOUS	HETERO-GENEOUS	HETERO-GENEOUS	HETERO-GENEOUS	HOMO-GENEOUS	HOMO-GENEOUS
NUMBER OF NODES	> 46	36	9	3	10	9	4
GEOGRAPHY OF NODES	USA	USA	UC IRVINE	MICHIGAN	LBL	USA	NORTH CAROLINA
MACHINE SIZE	MIXED	LARGE	MINI	LARGE	LARGE	360/67	360
COMMUNICATION INTERFACE MACHINE	HONEYWELL DDP-516	CDC 3300 PPU	RING INTERFACED	PDP-11	CDC PPU	IBM 2701	IBM 2701
COMMUNICATION PROTOCOL	MESSAGE SWITHCED	MESSAGE SWITCHED	MIXED	MESSAGE SWITCHED	POINT TO POINT	POINT TO POINT	POINT TO POINT
TRANSMISSION MEDIUM	LEASED LINES	LEASED LINES	TWISTED PAIR COAXIAL	TELPAK	COAXIAL	DDD	TELPAK
DATA RATE bps	50,000	100- 40,800	2.5 M	2,000	1.5-12 M	2000 40,800	100-2400 40,800
TRANSMISSION MODE	ANALOG	ANALOG	DIGITAL	ANALOG	DIGITAL	ANALOG	ANALOG
MESSAGE FORMAT	VARIABLE LENGTH	FIXED LENGTH	VARIABLE LENGTH	VARIABLE LENGTH	VARIABLE LENGTH	VARIABLE LENGTH	VARIABLE LENGTH
MESSAGE SIZE	8095 BITS	1024 CHARS	900 BITS	240 CHARS	1208 or 3,780,000 BITS	8192 BITS	1000 BYTES

Figure 1.9 Features of Existing Networks

used, is to simulate the network [30,31 ,32] . Simulation allows modelling of steady state and time-dependent systems with relative ease and permits analysing transient conditions. Since no explicit equations need to be derived, a simulation may handle a large number of variables thereby enabling complex models to be handled more easily. In fact, where the number of variables is large, simulation may be the only course available. In order to develop a simulation model it is necessary to know the distribution of the various processes.

Simulation also permits discrete change models to be modelled easily which have been used widely to study communication networks. Discrete change models conceptualise the changes in the state of the system as discrete rather than continuous. Such network models have the following characteristics

a) the system is defined by modules which operate on distinct parts of the model.

b) packets flow through the network from one module to another, at each stage a specific function is performed before being passed onto the next.

c) Each module has a limited capacity to process the packets, and therefore the packets may have to wait in a "queue" before reaching a particular module.

The main objective in discrete models is to examine the characteristics of the network and to determine the capacity of the system i.e. how many packets will pass through the network in a given time. The main computational task consists of keeping track of where individual packets are at any given time, moving them between routines, timing the moves and processing times at the modules.

Simulation techniques are limited only by the capacity of the computer but generally at a higher computational cost than queueing theory. Furthermore the results of a simulation tend to be in a form that is easier to interpret than those of queueing theory. To be of value, however, a simulation model must be accurate.

Bearing in mind that the network will be conceptualised as a discrete change model where transient conditions need to be studied, it was decided to carry out the investigation using a software simulation approach.

1.4 Objectives of Investigation

Figure 1.10 shows the location of computers at the University of Aston and the types available. The computers are drawn from a variety of manufacturers and are mainly incompatible both in hardware and software. Since many of these computers are underutilized and do not have the same facilities as other computers on site, it would be advantageous to join these computers on a network to enable other users to use the system during slack periods and allow all users to access the unique features of individual machines.

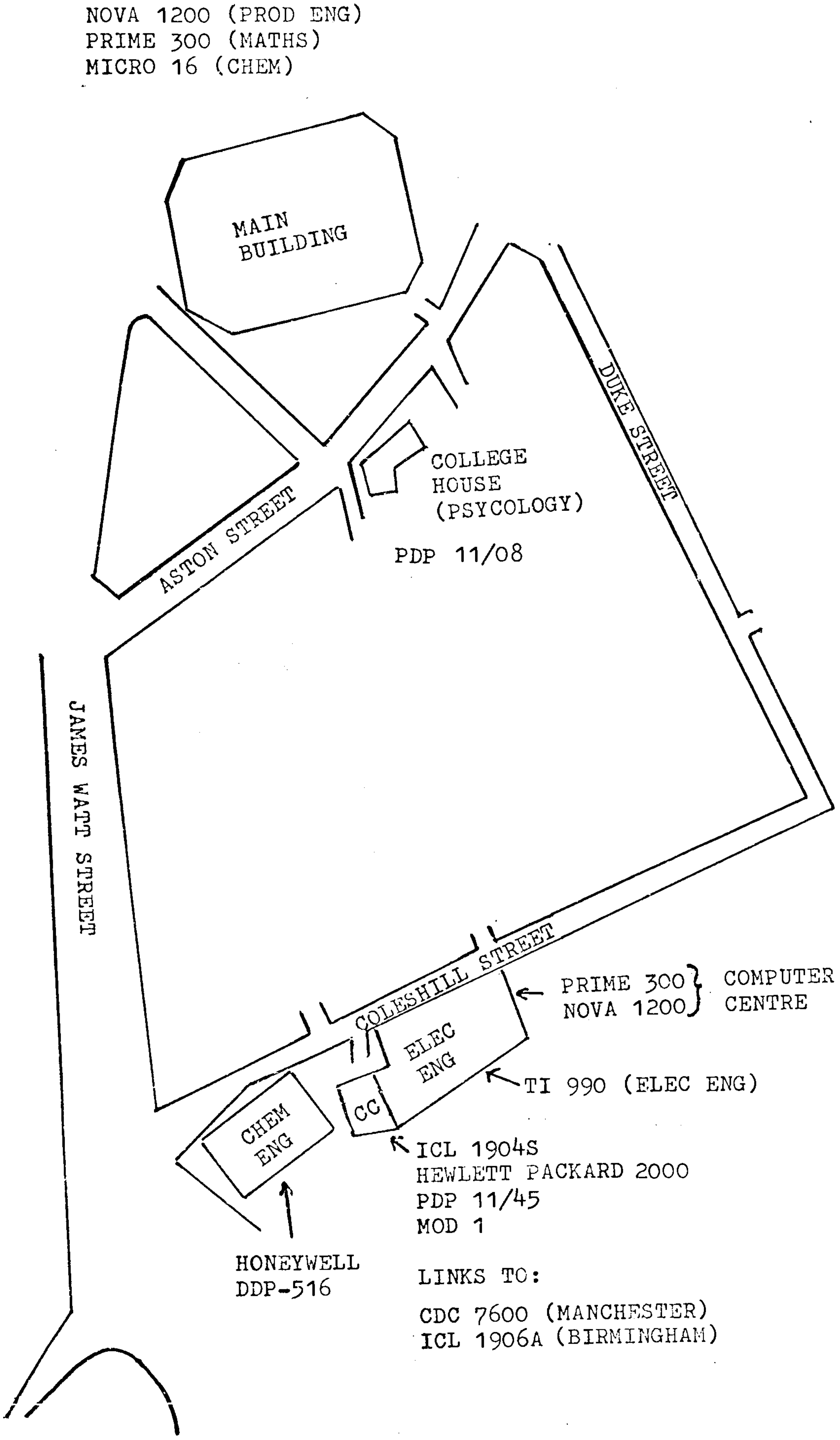


FIGURE 1.10 TYPE AND DISTRIBUTION OF COMPUTERS AT ASTON UNIVERSITY

There are a number of fundamental questions to be answered in the process of designing computer networks. Major problems are the layout and sizing of connections between nodes i.e. where should a line go and what should its capacity be? What should the packet length be? What traffic load can the network sustain? These problems are difficult to answer because there are many possible combinations to choose from. A simulation model has therefore been developed to help in the investigation of the problem.

Packet switching communication systems have two fundamental goals in processing data - low delay and high throughput. The major modelling effort is concentrated on the study of the behaviour of messages queueing for access to the network. This seems a reasonable approach since in a packet switching network, messages experience delays as they are transmitted through the network and thus the queue lengths and speed with which messages are throughput are a reasonable performance measurement.

The areas of interest include the relative capabilities of the network, identification of specific limitations of the network and may be divided into three main areas:

1) Node parameters:

   a) effect of number of processor/memory modules

   b) effect of memory/processor speed

   c) memory module size

2) High level Network parameters:

   a) Number of nodes in network

   b) Number of hosts

   c) effect of line speeds

3) Low level Network parameters:

   a) effects of increasing mean packet length

   b) effects of changing ratio between different message lengths

   c) effect of increasing mean number of generating hosts

   d) effect of increasing packet length.

CHAPTER TWO

FUNCTIONAL ASPECT OF NETWORKS

2.1 Introduction

This chapter is concerned with developing some of the ideas introduced in the previous chapter. The purpose of this is to indicate some of the problems that will be encountered in this investigation. After discussing packet-switching and packet format, the functions of a node are presented with particular emphasis on message-handling and buffering, error control, flow control and routing. Finally Host-Node protocol is discussed.

2.2 Packet-Switching

Rather than provide channels on a user-pair basis, it would be much more efficient to provide a single high-speed channel to a large number of users which share it in some fashion. This brings into effect the "large numbers law" which states that with very high probability, the demand at any instant will be approximately equal to the sum of the average demand of that population." In this way the channel capacity required to support the user traffic may be considerably less than in the unshared case of dedicated channels. The important observation that can be made is that the full-time allocation of a fraction of the channel to each user is highly inefficient compared to the part-time use of the full capacity of the channel (this is precisely the same idea as the notion of time-sharing).

2.2.1 Packet Format [33,34]

Each packet is a group of control and data bits which is independently transmitted to find its way to the destination. The control bits contain addressing and processing information and have a fixed format while the data bits may have any format providing the total data bits can be carried as multiples of 8/16 bit bytes.

As shown in Figure 2.1 a typical packet would contain the following control information:

1) Source Address

2) Destination Address

3) Type of packet

   (a) Message

   (b) Acknowledgement

   (c) Special inter-nodal control messages giving status information concerning buffers, lines, and node status

   (d) "Send next packet"

4) Packet Sequence No $\Big\}$ only for data packets.

5) No of packets in a message

In each packet after the data, a sequence of check bits are added to enable the receiving node to check whether the packet has been transmitted error-free.

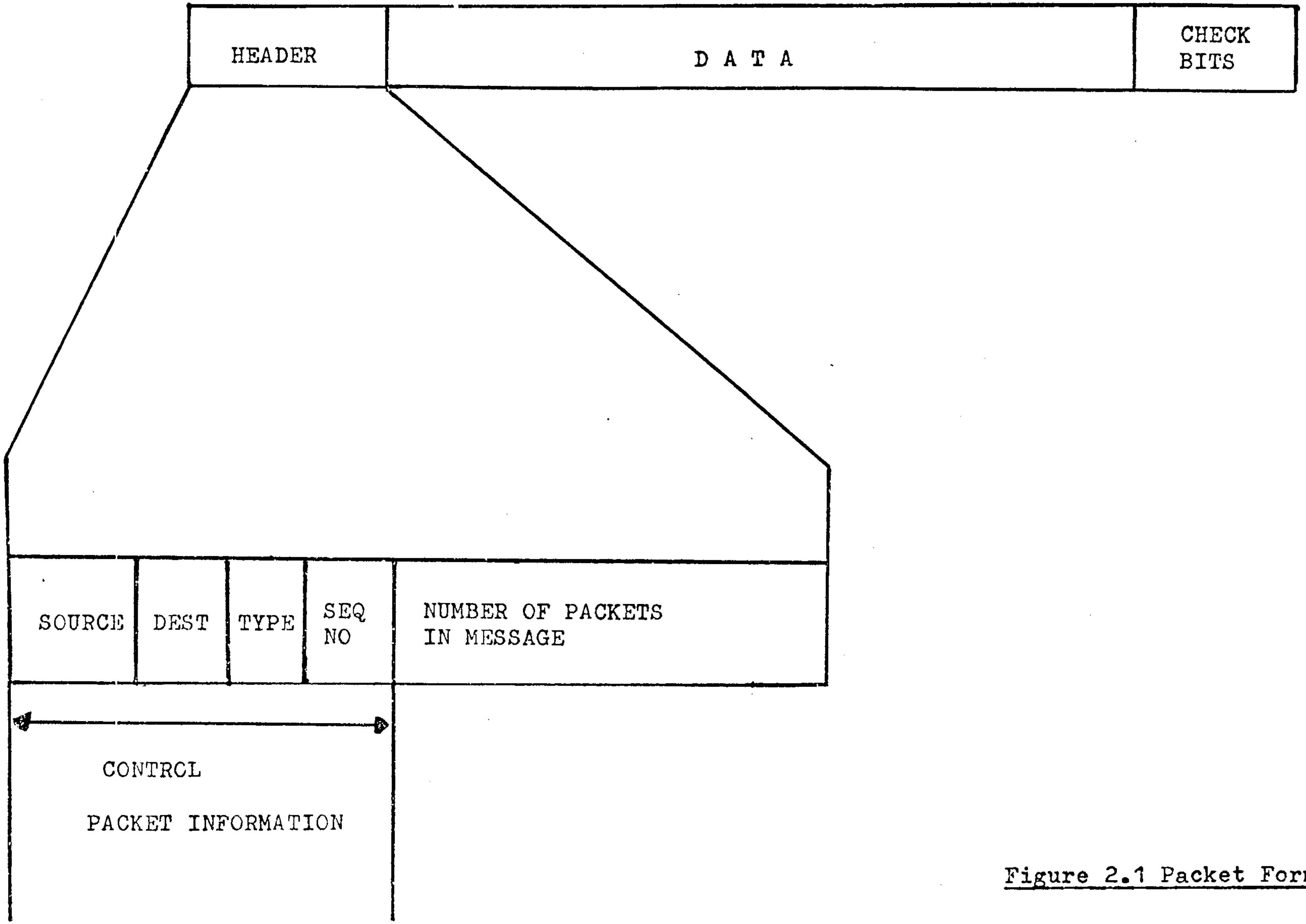


Figure 2.1 Packet Format

Packets therefore serve as the basic unit of information interchange between nodes. Their smaller size places a reduced demand on intermediate nodal storage and increases the likelihood of error-free transmission.

Consider as an example a data terminal in London wishing to use the services of a computer centre in Manchester. Using the switched DATEL services, a link would be established via the telephone network. That link would then be held for the duration of the complete transmission, even if there were periods during the call when no data was being transmitted. All conflict and allocation of resources must be resolved before the link can be established thereby permitting the traffic to flow with no conflict.

The alternative to this is packet-switching where the packet would be sent to the local packet-switching exchange (NODE). On arrival at the exchange it would be transmitted to the destination node by the most readily available route. During the transmission from source to destination there is no dedication of resources - conflicts being resolved as they are encountered. Internodal communication-lines are engaged only during packet transmission. During idle time the lines are available to other users. Should the most direct line be congested en route, the packet would be rerouted automatically through an alternative node. This ensures that:

1) Users are occupying long distant lines for the minimum of time.

2) High probability of rapid packet transfer.

3) Multiaddress or "broadcast" messages are facilitated.

4) Speed changing catered for - allows terminals of different speed capabilities to communicate i.e. source transmission rate = reception rate of destination.

5) No dedication of resources.

6) Increases likelihood of error-free transmission.

7) More effective use of data channel.

An optimum packet length exists which depends on the application and the environment. If the packet is too long, errors will be so frequent that few data can be transferred; if the packet is too short, there is an unnecessary overhead, and too many control packets will be generated during the transaction.

2.3 Functions of a node

A node has four basic functions to perform:

1) Message handling and buffering.

2) Error control is required in four situations:

    (a) Out of sequence delivery of messages at the destination

    (b) Delivery of duplicate messages at the destination

    (c) Message delivery with errors

    (d) Message not delivered.

3) Flow control.

4) Routing.

2.3.1 Message handling and buffering [4,10]

The most important function of a node is to handle traffic from the local hosts that it services and traffic from neighbouring nodes.

Communications between hosts is via sequences of messages, each of which is broken up into sub-messages called packets by the local node. Each message in ARPA can have a length of up to 8000 bits which is broken up into packets of 1000 bits length. Each of these packets is then independently transmitted by the network to the destination node which reassembles the packets into a single unit prior to handing it over to the destination host. Along the route each node retains a copy of a packet until a positive acknowledgement is received indicating error-free transmission and acceptance (e.g. the node is not too busy). Should an acknowledgement not be received within a reasonable time say 0.1 seconds it is retransmitted, possibly along another line. The examination of the packet header by the node will determine whether a packet is at the destination node or whether it needs further transmission.

When the destination node has received all the packets in a message, it must put them into the correct order, strip off the header from each packet and put a leader on the message, identifying the source host. Once the host has received the message it will issue a "Ready for Next Message", which is transmitted back to the source host where it also serves as an indication that the message was correctly received. Figure 2.2 illustrates packet-switching between host computers.

If buffer handling is made simple then fast processing will be achieved and the program size will be kept to a minimum. The number of buffers should be such that all incoming traffic can be stored to enable the lines to be used to full capacity.

Fixed buffer sizes simplify the design and speed up the handling operation. Variable packet lengths lead to inefficient utilization of network resources (buffers etc.).

The high level network avoids an extensive message-buffering problem by preventing any host sending a message to any other host that is in no condition to receive messages.

If a host is to be effective on the network, it must be willing to receive and acknowledge messages with extremely little delay. Then the major burden of message buffering is on the host computers themselves.

2.3.2 Error Control

The node has full responsibility for providing error control. Four situations can arise which the node must handle. Allowing messages to be multipacket and sent along independent routes will lead to the packets arriving at the destination node out of sequence. At the destination node the packets

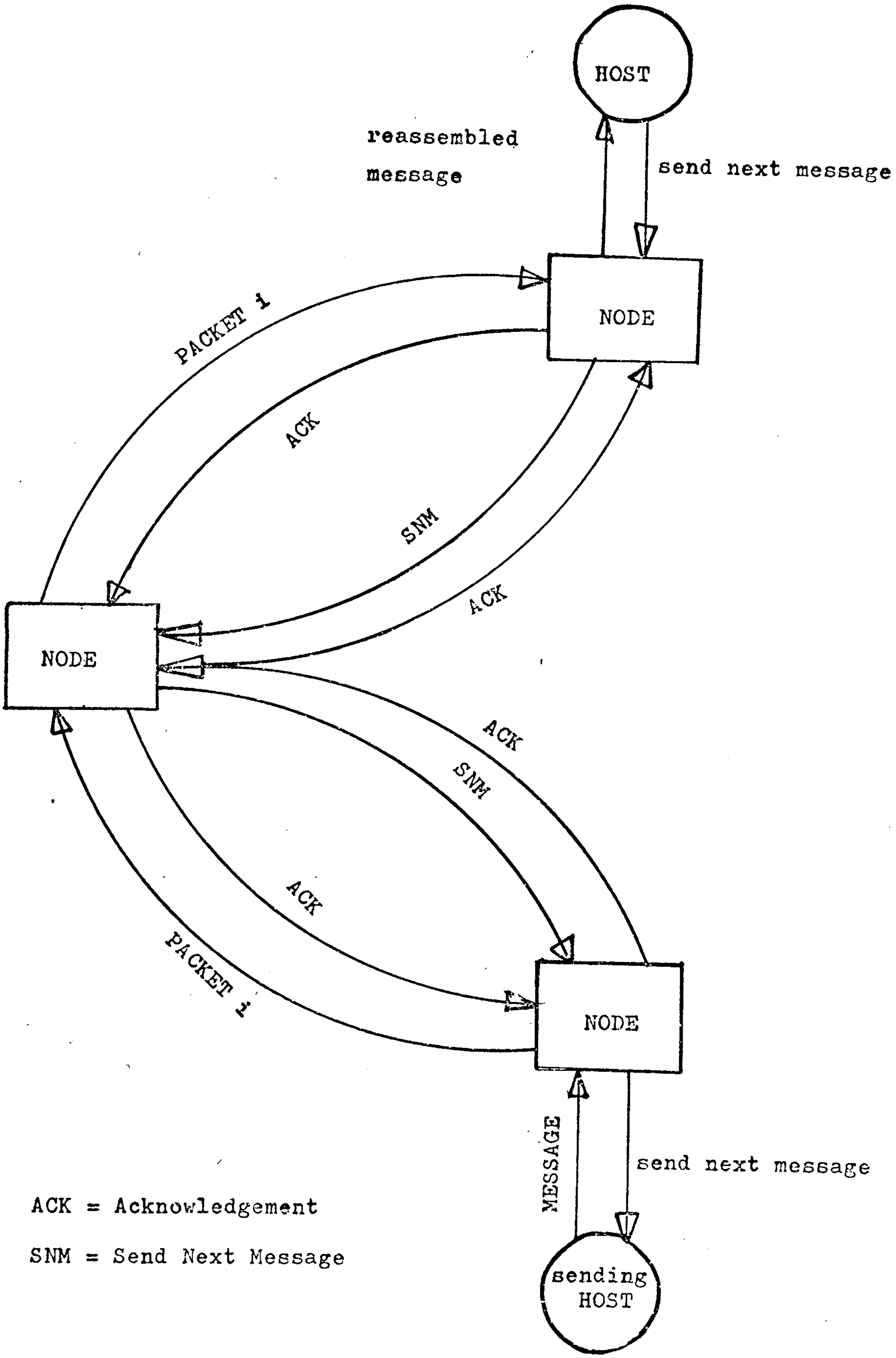


Figure 2.2 Packet Switching between host computers

must be reassembled into the correct sequence before being handed over to the destination host. Assigning packet sequence numbers would enable this to be carried out. The situation where a host pair could have several messages on the system could arise but is best avoided as this further complicates the network.i.e. with each host pair allowed one message, sequencing would occur naturally.

Should an acknowledgement be missed somewhere along the route it is possible that a duplicate packet would be retransmitted. The provision of sequence numbers for each packet in the message would enable the message to be correctly reassembled at the destination.

Noise is the primary cause of errors on communication channels. Error handling is simply achieved by error detection (e.g. cyclic redundancy code carried out on each packet) and retransmitting the packet if necessary when an acknowledgement is not received through an erroneous packet.

Each transmitted message will now be accurately delivered to the intended destination through reliable network design. Should a message fail to get through, simple end-to-end retransmission would protect against the occurence of this situation.

2.3.3 Flow Control

It is clear that any network has a limit to the amount of traffic that it can support. Should traffic rise over a certain level, it must be rejected or the network will grind to a halt. When a network reaches a situation where it must reject traffic then it is said to be "congested" or "logically deadlocked" where traffic movement has stopped. To prevent these situations occuring good flow control techniques are required. The provision of mass storage in the nodes could greatly increase the mean time to congestion. However, more storage alone cannot in general prevent congestion. The network must provide a certain amount of buffering between the source and destination host, preferably equal to the band-width of the channel times the round trip time over the channel. Flow control is necessary to prevent messages from entering the network for which there is no buffering available.

As with road traffic, congestion may be expected to start at one point in the network and spread as the queues fill and links betweennodes are blocked. The workload a network can take will be increased through good routing. Eventually, a limit will be reached where several lines or nodes block simultaneously.

Congestion control methods can be classified as "local" or "end-to-end". Local control is applied in a subnet on information passed between nodes. Neighbours inform each other of traffic delays experienced, or may request reductions or return to normal traffic over certain links.

A user who is contributing to the overload may be some distance away from the point where congestion is occurring and local control methods may have to spread some way before action is taken.

End-to-end control makes use of the notional links that exist between subscribers. Under heavy loading data rates of certain links may be reduced and new links may be refused.

Two types of deadlocks may occur known as "reassembly lockup" and "store-and-forward lockup" which occur with multi-packet messages. Reassembly lockup is the situation where the remaining packets of a partially reassembled message are prevented from reaching the destination node by other packets in the network that are waiting for reassembly space at that destination to become free. Thus the first message cannot be completed and the reassembly space freed. In the second case of store-and-forward lockup, packets interfere with each other by tying up buffers in such a way that none of the packets are able to reach the destination although the destination has room to accept arriving packets.

Figure 2.3 shows the problem schematically. Node 1 is sending

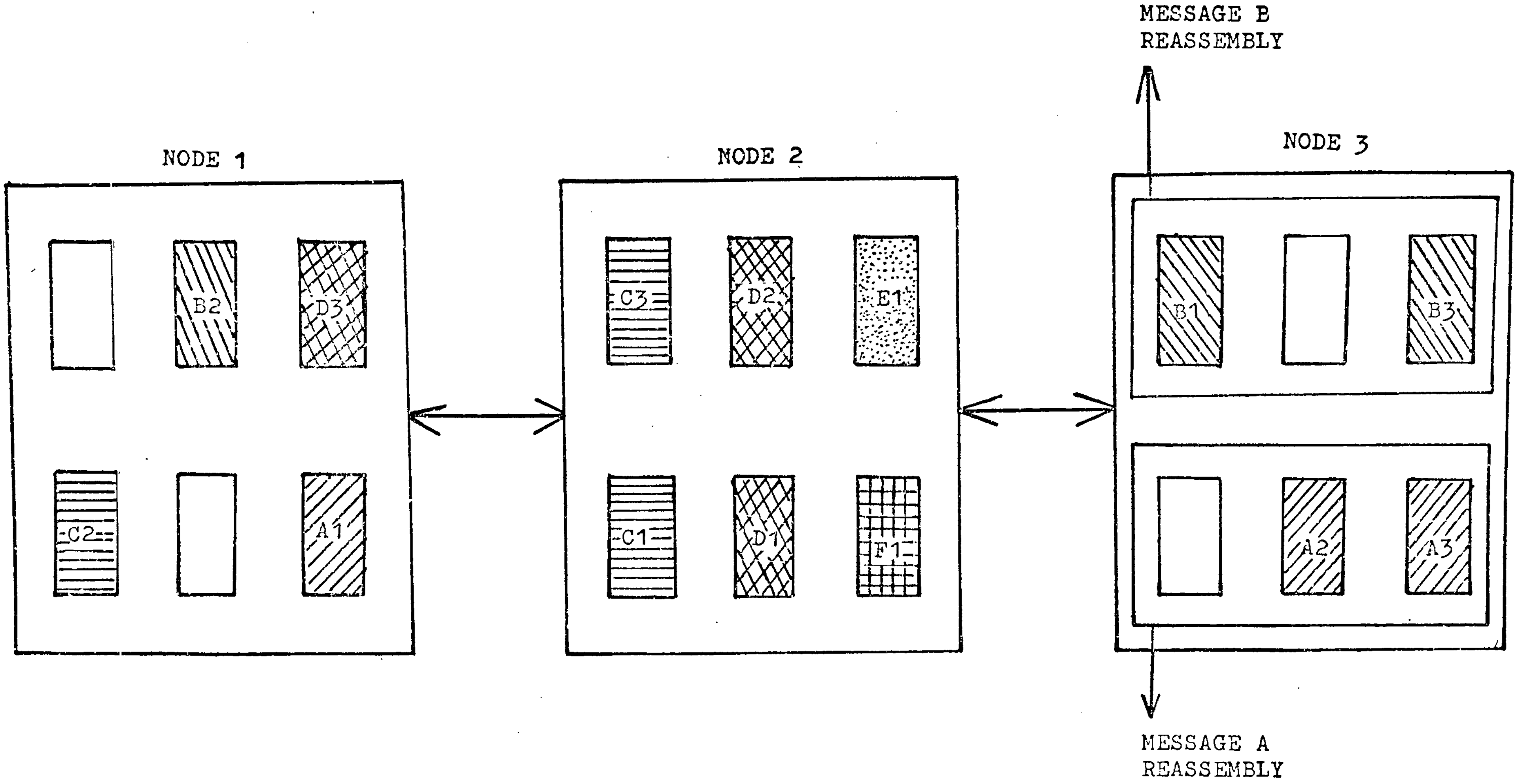


Figure 2.3 Reassembly Lockup

a multipacket message to node 3. Node 3 has devoted its buffers to partially reassembled messages A and B. Since all the buffers are tied up to messages A and B, the node can only free space when the remaining packets have been received from node 1. This is reassembly lockup. Packets A1 and B2 cannot get through since node 2 is in a store-and-forward lockup containing packets of which none are destined for node 3. Node 3 will therefore never complete its message-reassembly.

ARPA solves this problem by not permitting any multipacket message onto the network until the destination IMP has reserved reassembly storage. On receipt of the first packet from a host a control message is sent to the destination node requesting reassembly storage. When an acknowledgement has been received the IMP takes the remaining packets of the multi-packet message from the host. This strategy will ensure that message D in node 2 could be reassembled.

An incomplete message at node 3 could be discarded at tnis point, since eventually a copy of the message would be retransmitted from the source host. The source host could be informed of this move,

Another solution would be to use overflow buffers thus ensuring that one packet at least would reach the destination, The packet to be sent could be selected randomly. The receiving node would acknowledge the packet if it was useful. If no acknowledgement is received another packet would be tried.

One node could take charge in this situation and try to sort things out. However, this solution would further increase the traffic and the controller itself would be vulnerable to failure.

The problem could be further eased by only allowing one message between host pairs at any time. The acceptance of the message by the destination host would be followed by a "Ready for next message." This solution would prevent the overloading of a node of host. Of course, it is assumed that some users would be transmitting messages rather than many users transmitting single packets coincidentally.

Davies [35] suggested that congestion could be prevented by placing a limit on the total number of packets in the network. Since data-carrying packets must be created and destroyed, the balance is kept by using empty packets. Thus the arrival of a normal data-carrying packet at its destination would result in its replacement in the network by an "empty" packet. Similarly, when data is ready to enter the network, an empty packet must be found to be replaced by a new data-carrying packet.

This constant group of packets can, by analogy, be compared to a gas composed of molecules in perpetual motion. Packets will arrive and leave each node in the network at a roughly constant rate, regardless of the data traffic. All packets will eventually visit each node due to the randomness of the motion.

In order to keep the empty packets moving around the network some rule is needed. The rule should have a random element. Therefore, in its simplest form, a destination node could be chosen at random. When the empty packet arrives at the specified destination address it will be used by any data awaiting transmission. Should no data be waiting, a new destination address would be chosen and the packet would try elsewhere. When a data-carrying packet arrives at its destination, it would clearly be sensible to give data waiting at that node priority to use the empty packet rather than sending it randomly back into the network. Further efficiency could be gained by retaining a small store of empty packets at each node.

Should any traffic be offered beyond its capacity, the network would reject it until empties were created to resume normal operation.

Although the Isarithmic Network is quite attractive it does suffer from three serious problems:

1) Even though a rule exists for moving empty packets around the network, local congestion could still occur

due to the random element of distribution i.e. packets could collect at one node.

2) Some of the empty packets are required for control information thus reducing the effective number of usable packets.

3) It is as vulnerable as a central processor in a star network. If a node failure occurred the network could steadily gain or lose packets.

2.3.4 Routing [4,10,36]

Good routing strategies will ensure that message delay in the network is minimised. Message delay is the time taken to send a message from source to destination. A good routing strategy will also be adaptive to changing traffic levels and changing network topology in the event of a failure.

Each node applies some routing technique to decide the next link that the packet must travel over. In a distributed network the node will have to make a decision based on information it currently holds about the state of its neighbours, together with local information regarding the state of buffers and lines.

Perhaps the simplest strategy is fixed routing where packets from host i to host j always take the same route. The poor adaptability of fixed routing may be overcome by increasing the route reliability.

Random routing, which takes no account of the destination tends to give long average delays although they are very adaptive.

A simple strategy would be for each node handling a packet to send it along the current estimate of the shortest route to the destination. It is not enough to base a strategy solely on the local information such as internal queue lengths. However, it is a simple matter for each node to inform its neighbours of its state and from this all nodes can compute the current shortest path.

Since routing information itself suffers from a time delay, raw data such as current queue lengths are not enough to accurately characterize traffic flow. Some sort of averaging procedure must be employed in order to effectively select the shortest route and help to predict in advance where a possible traffic build-up could take place.

One scheme used on ARPA is to send packets along that line with the minimum estimated time delay to the destination. The time delay information is updated every 0.5 seconds using information from the nodes' neighbours regarding minimum time delays together with internal estimates of the delay to each neighbour. Should the traffic flow increase heavily this strategy may become inefficient due to the fact that information of queue lengths may change faster than the information can be distributed.

ARPA used a more intricate scheme to overcome the inefficiency of the last algorithm during heavy traffic. The packets are now routed down the path with the fewest nodes and which have excess capacity. If that path becomes full then the one with the next fewest nodes and excess capacity is chosen.

EPSS [36] used an Alternative-Routing strategy. A table is accessed which gives the next path that must be taken. Should this path be congested the second choice is tried and so on until all choices are exhausted. Should there be no path, failure routines are invoked. To determine routing the node is restricted to the following information:

a) Packet destination derived from packet header.

b) The node from which the packet has just been received.

c) The current queue length for each route.

d) Which of the routes are the faster intercomputer routes.

e) The packet source derived from the packet header.

(b), (c) and (d) are used to determine whether the current choice is satisfactory.

This type of strategy is deterministic in that action is taken on traffic information only when route queues exceed a preset threshold.

Figure 2.4 shows the relative merits of each of the strategies.

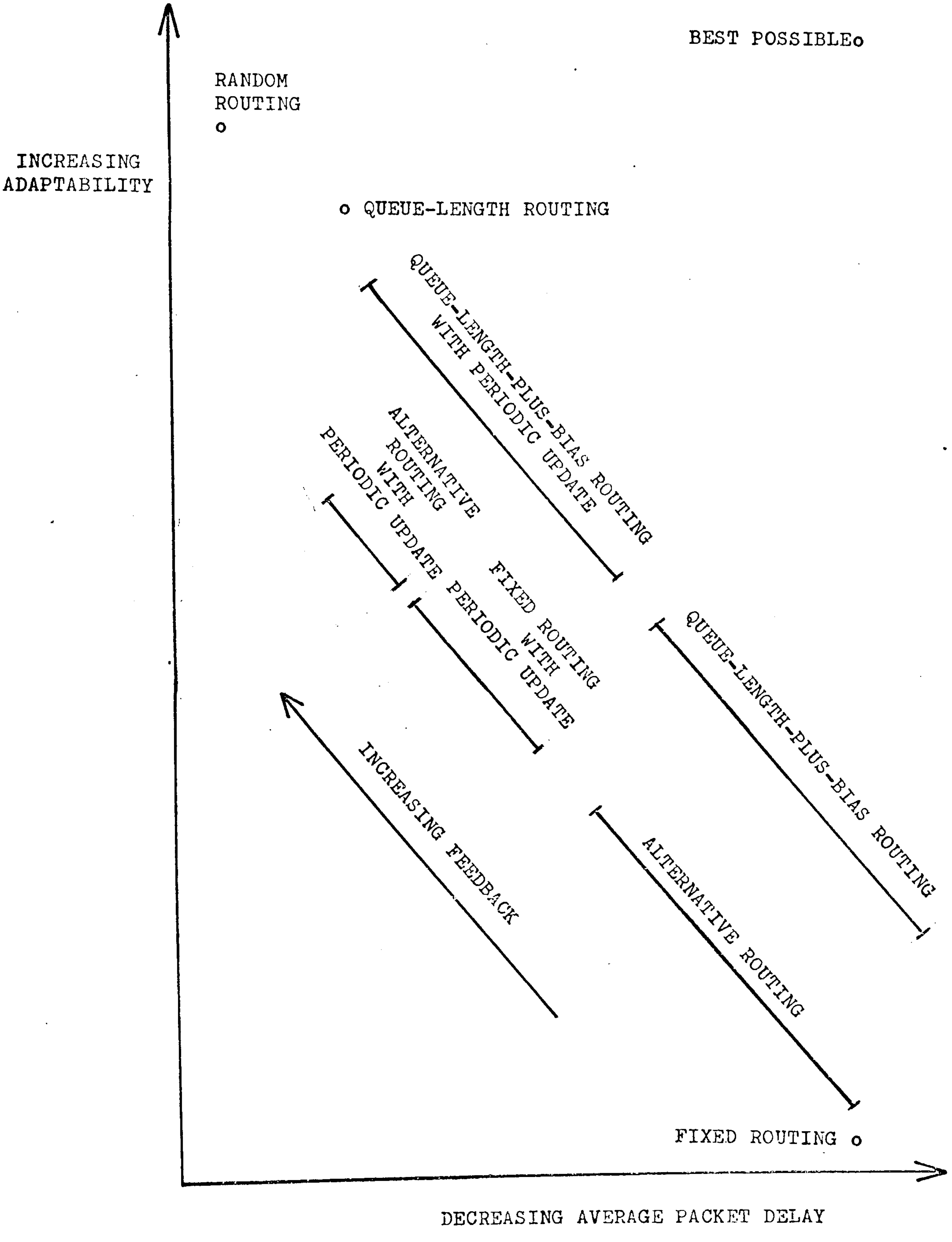

BEST POSSIBLE
RANDOM ROUTING
INCREASING ADAPTABILITY
QUEUE-LENGTH ROUTING
QUEUE-LENGTH-PLUS-BIAS ROUTING WITH PERIODIC UPDATE
ALTERNATIVE ROUTING WITH PERIODIC UPDATE
FIXED ROUTING WITH PERIODIC UPDATE
QUEUE-LENGTH-PLUS-BIAS ROUTING
INCREASING FEEDBACK
ALTERNATIVE ROUTING
FIXED ROUTING
DECREASING AVERAGE PACKET DELAY

Figure 2.4 Comparison of different Routing Strategies

Two of the strategies have not been discussed. Queue-length routing transmits packets on the route with the shortest queue, regardless of packet destination. This gives very long delays compared with fixed routing.

Queue-Length-Plus-Bias is a compromise between the short average delay of fixed routing and the adaptability of the last strategy. The route selection is based on the evaluation of a function of queue lengths and bias for each available route. The bias terms are preset constants whose values import a gross traffic-flow pattern. An example of such a function,f, is

f[(bias term for route) - k(queue length for route)],

where k is a constant. The route selected is the one whose function value has the largest numerical value. Under low traffic conditions, the system reverts to a fixed route strategy, but as traffic builds up, so the system adapts to equalise the imbalance of route usage.

Eventually, as traffic levels increase, the re-routing of packets can no longer prevent congestion, and the network must reject traffic offered to it.

2.4 Host-Node Protocol

A number of questions need to be asked regarding the relationship of the host to its node. What tasks shall be performed by each? What constraints will one place on the other? What dependence shall the node have on its host?

The following tasks must be carried out:

1) Breaking up a long transaction into message blocks so that the length of the message is within the networks constraints (8000 bits on ARPA).

2) Formatting and code-converting the message blocks into a standard format acceptable to the network.

3) Attaching a header to each message block giving address and control information.

4) Attaching a trailer with error-checking information to each message block.

5) Storing the unacknowledged messages and/or message blocks for possible retransmission.

6) Reassembling received message blocks into messages.

7) Breaking the message blocks into packets.

8) Preceding each packet with a header.

9) Adding a trailer to each packet.

10) Storing the unacknowledged packets for possible retransmission.

11) Reassembling received packets into message blocks.

12) Controlling the input rate to avoid congestion.

On ARPA tasks 1-6 are carried out by the host while the remaining functions are the responsibility of the node. ARPA[4] was guided by the following principles:

1) The node should function as a communication system whose primary task is the reliable transfer of bits from

a source to a destination. Bit transmission should be sufficiently reliable and error-free to remove the need for special precautions (such as storage for retransmission) on the part of the host.

2) The node operation should be completely autonomous. Since the node must function as a store-and-forward system, it must not be dependent on its local host. The node must continue to function irrespective of the correct functioning of the host. So the node must not depend on the host for buffer storage or program reload. Also the host must not be able to change the logical characteristics of the node.

The general philosophy of host programming adopted by many networks is that network features are extensions and additions to the operating system and not changes to develop compatible software [21]. This is the principle of host autonomy. The connection between a pair of processes appears as an I/O device in each host. This respect for individuality ensures that the unique resources of the host are not only preserved for local use but also for global use on the network. The imposition of unnecessary commonality may simplify network structure but would probably stifle interest.

CHAPTER THREE

NETWORK PHILOSOPHY

3.1 Introduction

This chapter is concerned with the philosophy of the proposed network which will be later simulated. After an introduction to microprocessors, methods of organising microprocessors into multicomputer and multiprocessor systems are discussed. An outline is given into ways of organising microprocessors to achieve better computer reliability.

The architecture of the node centres on a multi-microprocessor system operating under amaster processor. The organisation of the node together with the functions of the command and slave processes are described.

The essence of a network is its design philosophy, its performance characteristics, and its cost of implementation and operation. Unfortunately, there is no generally accepted definition of an 'optimal" network or even of a "good" network, although work has been done in this area[37,38]. A network designed to transmit large quantities of data during the night might call for characteristics in structure and performance far different from one servicing large numbers of users who are exchanging messages only during business hours.

The main functions of the node in a distributed store-and-forward packet-switching network are to establish a connection between hosts wishing to communicate, accept messages or packets from hosts or other nodes, and to route these successfully to other hosts or nodes. Since a general purpose machine may not be cost effective in such a situation a multi-microprocessor system is being investigated for use as a network node.

The advantages of using such a system include the low relative cost of components, improved reliability achieved as a result of multi-processing and faster operational speed due to parallel processing. Greater flexibility in construction is also gained by individually tailoring node hardware whilst little program change is required. This in turn eases the problems of network expansion and upgrading of nodes. The instruction set of a general purpose machine will not allow the packet handling routines to be programmed as efficiently as they could be directly in microcode. The non-volatile nature of the storage used to store microcode will ease system recovery after breakdown.

3.2 Microprocessors

Since 1971 when they first appeared on the market, microprocessors have evolved into fundamental system-building blocks. Rapid advances made in the field of large-scale-integration semiconductor process technology have also resulted in significantly improved circuit packing densities. RAM's containing 4k bits on a single chip are readily available and 16k bits chips are becoming available. The emergence of single-chip peripheral interfaces will enable system designers to construct complete computer systems with a handful of components at costs which would have been considered impossible a decade ago.

The replacement of hard wired systems by microprocessors will bring the inherent advantages of store-program control which include improved flexibility,reliability, ease of maintenance, and lower cost. Multi-microprocessors, which provide distributed processing, are a natural evolution of microprocessor-based architecture. A distributed workload will improve system throughput, increase reliability and add a further dimension of flexibility.

Microprocessors may be organised into multicomputer and multiprocessor systems. Figure 3.1 shows a multicomputer system where it can be seen that several input streams are being operated on and no integrated operating system exists. Each processor is performing a dedicated task. Interprocessor communication is primarily at the data level. In more sophisticated systems the data may take the form of commands to initiate specific actions or responses from the other processors. Each processor can be regarded as having two I/O ports: one being associated with external activity; the other internal i.e.interprocessor communications. Multicomputer systems are commonly used in larger systems where tasks are mostly independent: main CPU performs number crunching while I/O processors rapidly respond to I/O requests.

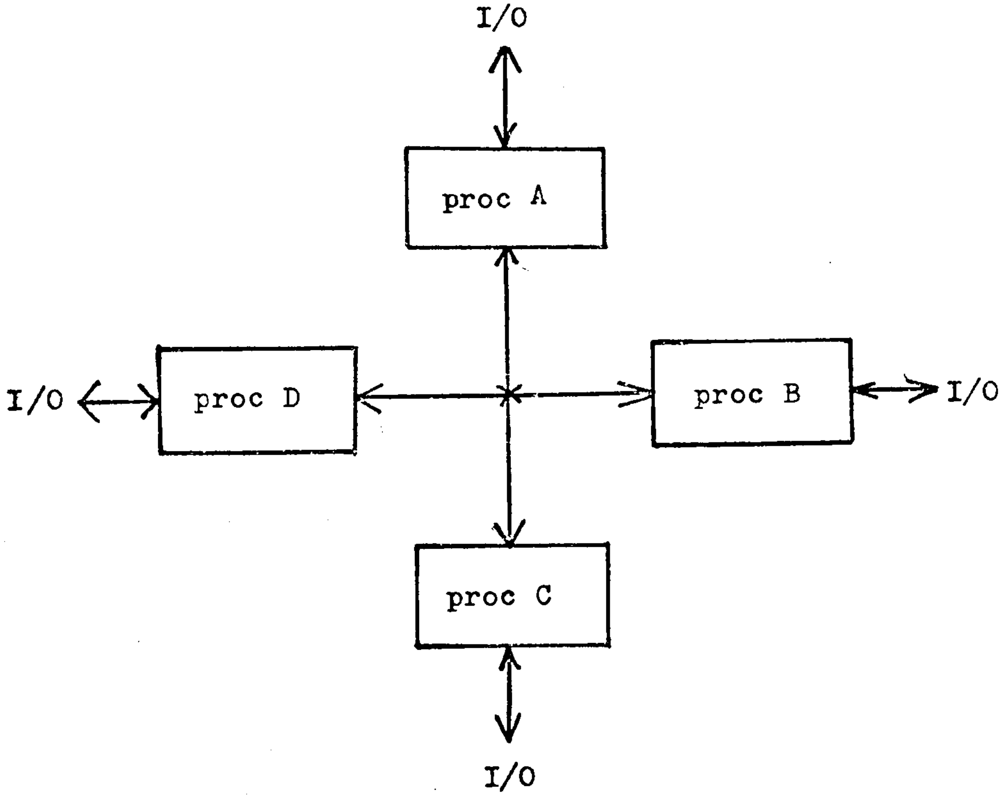


Figure 3.1 Multicomputer System

Figure 3.2 shows a multiprocessor system where several processors share tasks from a single input stream or work load. A single integrated operating system allocates hardware resources as and when required. Multiprocessors would be used in situations where high reliability is required. This would be achieved through a fully redundant system or through system reconfiguration when a fault occurs in one of the processors. Other features of a multiprocessor system would be shared main memory; I/O channels and controllers would be accessible by each processor as required. In addition each processor may have its own RAM and may have privileged access to certain resources.

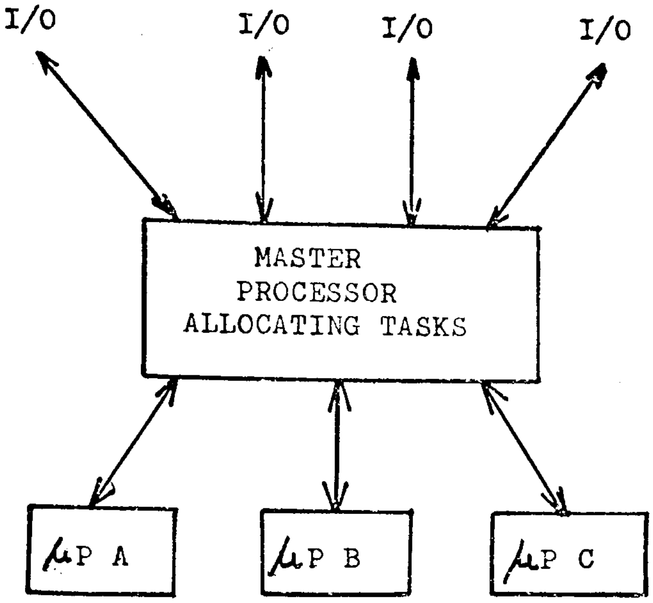


Figure 3.2 Multiprocessor System

There are many possible multiprocessor architectures from a master/slave to a ring structure. Figure 3.3 shows a master/master configuration where every processor is of equal status. This organisation is generally restricted to large computer networks such as ARPA and does not readily lend itself to microprocessor application.

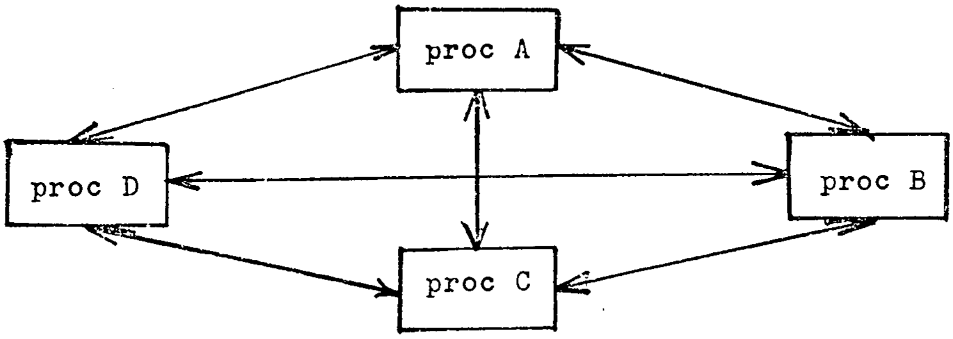


Figure 3.3 Master-Master Multiprocessor Organisation

However, the more common master/slave organisation shown in Figure 3.4 lends itself very well to microprocessor applications. All interprocessor communication goes through the master processor. This organisation enables resource conflicts to be resolved by only allowing a particular processor certain resources at any time.

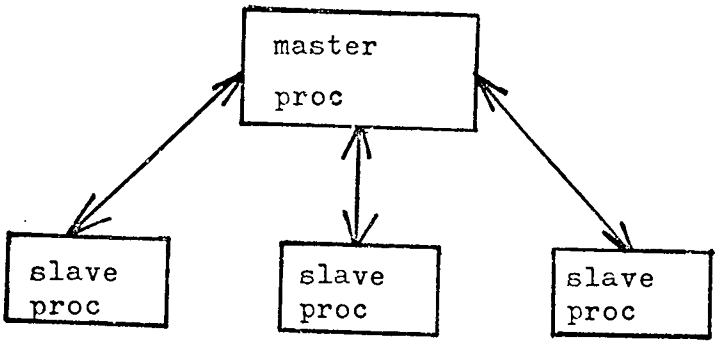


Figure 3.4 Master/slave Multiprocessor Organisation

Finally, there is a ring structure available as shown in Figure 3.5. This has many disadvantages ranging from a fault-sensitive information bus to the problem of congestion on the bus if it is also used by the processors for their own processing.

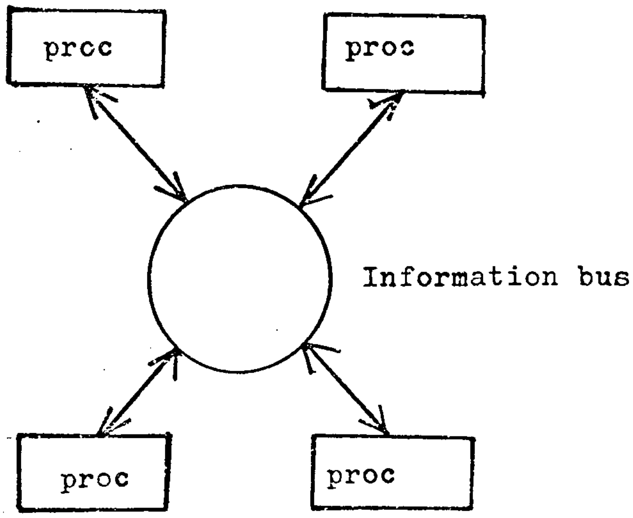


Figure 3.5 Ring Multiprocessor System

The master/slave system described is an hierarchy in which slave processors communicate with one master. Polled or interrupt driven systems tend to use radial buses shown in Figure 3.6.,

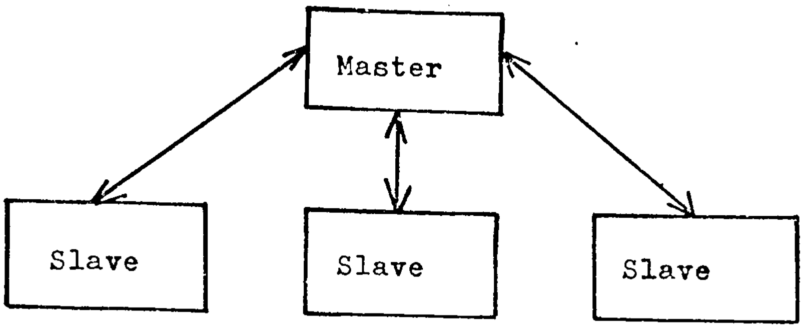


Figure 3.6 Radial bus

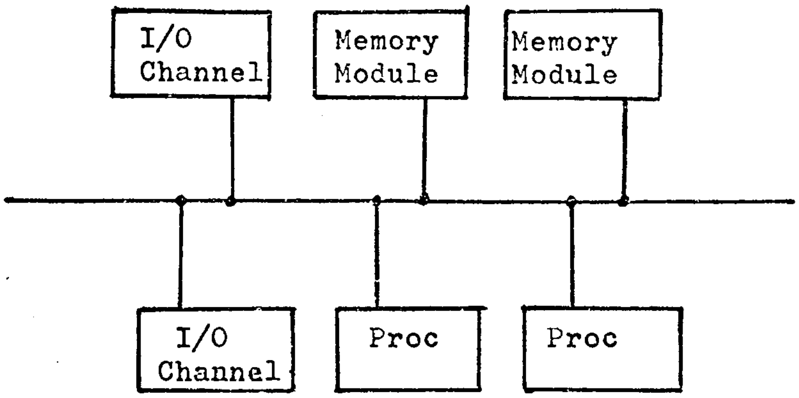


Figure 3.7 Time shared/common bus system - single bus

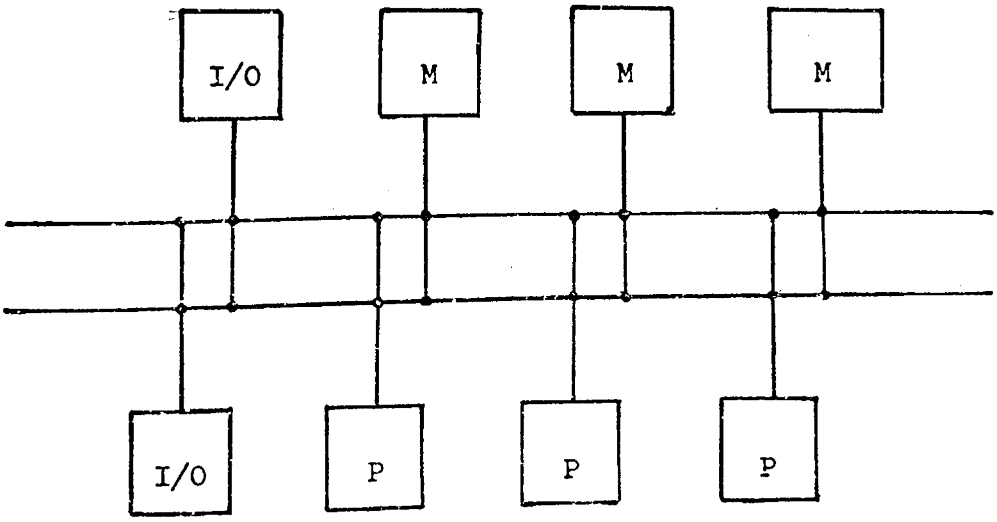


Figure 3.8 Multiple time-shared/common bus system

otherwise a common bus is used. The common bus may either be a unibus shown in Figure 3.7 or a multiple bus as shown in Figure 3.8. Such systems emphasise high information transfer. The weakness of the radial bus lies in all processor and resources being connected to the master, which becomes the weak point of the system.

The time-shared or common bus system is one of the simpler and cheaper organisations to implement. Unlike the radial system there are no continuous connections between functional units. Time-sharing or multiplexing techniques are used to enable data to flow between different units. Since there is only one path for all transfers, delays will be greater than in the multibus system. Again a single bus also weakens the reliability of the system. However, this organisation is flexible and easy to add to or remove modules. The modules are connected to the bus in parallel. The bus may be a bit, byte or word in width but the latter simplifies the control functions required.

With multibus architecture, which is essentially a crossbar system, each unit of information must be accompanied by the address of the unit for which it is destined. This organisation is much faster since more transfers can take place per time unit.

Distributed processing raises the question of memory design. It is usually advantageous that each processor has some local memory but it is debatable how much shared memory should exist in highly reliable systems. As the number of processing modules increases so too must the contention problems. Thus if a shared memory is to be used, access to it by individual processors must be restricted to a minimum. This would make local RAM'S necessary.

Shared memory is primarily required for interprocessor communication i.e. to act as a message centre where each processor can leave messages for other processors and can pick up messages left for it.

3.3 System Reliability [39,40,41,42]

There are two general methods of reliable computing: parallel processing systems and load sharing systems. The former are fault tolerant using redundancy and maintain the active structure of the system. The latter employ non-redundant fail-soft methods which "gracefully degrade" the performance of the system.

Avižienis [40] defined an operational fault as:
"a deviation of one or more logic variables in the computer hardware from their design-specified values."

Hardware faults may be of one of three types:

(1) "solid" component failure.

(2) "intermittent" component malfunction.

(3) externally caused interference with the operation of

the computer.

One of the mentioned faults will cause the program to be incorrectly executed or it may result in the program waiting for say a memory transfer that cannot take place.

Faults may be classified into three categories:

(1) Duration: transient (intermittent) or permanent.

(2) Extent: local (independent) or distributed (related i.e. the whole system will go down. The fault may be clock failure, power supply, data bus etc).

(3) Value: determinate ("stuck") or indeterminate (variable).

There are two types of fault-tolerant systems available: static or dynamic recovery. Static techniques rely on redundancy to enable single faults to be masked out by logic associated with the same function (this method is also called masking redundancy and massive redundancy). The redundancy may be provided in three forms:

(1) additional hardware (hardware redundancy)

(2) additional programs (software redundancy)

(3) repetition of operations (time redundancy).

An example of additional hardware redundancy is called triple modular redundancy whereby each module is triplicated and a vote taken at the output of the module. Hopkins and Smith describe such a system [42].

Static redundancy systems are simple to design, permit instantaneous fault isolation, and are simple to operate i.e. the operating system does not need to know about the fault masking. Against these advantages, static systems are very expensive, do not report internal failures and are very limited in the degree to which overall system reliability is increased.

The second technique of fault tolerant computing is dynamic redundancy (also called selective redundancy or stand-by sparing), where standby unit or even systems are provided to replace faulty units.

Failures must be detected with this technique and explicit action is required to remove the faulty unit from the system and replace it with a working unit. This replacement is automatic in more sophisticated systems. Reliable computing will thus continue in spite of the fact that faults may exist.

The advantages of dynamic redundancy over static redundancy are that they are less expensive and more effective in increasing system reliability. Against these advantages are the facts that dynamic systems are more difficult to design and that they are slower since time is required to detect and replace the faulty unit.

With the fail safe approach it is assumed that performance degradation would be acceptable. A highly modular structure is used with a multiplicity of all critical units so that the loss of one or more of these units can be tolerated. Large cost savings may be achieved with this technique given that some degradation is acceptable.

Borgerson [41] stated that the following characteristics were necessary for graceful degradation:

(1) A modular architecture with a multiplicity of each functional unit.

(2) The ability to rapidly detect failures and to identify the faulty unit.

(3) The ability to isolate the faulty unit.

(4) The ability of a system to reconfigure itself so that it can run without the faulty unit.

The fail soft approach was predominantly used in the 1950's and 1960's in both hardware and software. In recent years however the fault tolerant approach is being more widely used in hardware design. The change has come about with the decreasing costs of hardware and the sophistication of current technology. Combinations of these techniques are also used. [42].

If no redundancy is to be used then a multi-microprocessor system would be needed to provide the desired characteristics. The essential advantage of this system would be load-sharing i.e. all the processors would be participating in the total work load. On the detection of a fault, the faulty processor would be removed from service and the work load would be redistributed among the remaining processors. Memory, if present, must also be distributed amongst several units. System expansion is also easier and cheaper should the work load increase. Fail soft systems can be expanded using just one processor whereas the static redundancy system using triple voting would require the addition of three processors.

3.4 Network Design

Figure 3.9 show schematically the basic layout of the envisaged network. Since the network will be local to the ASTON University campus the number of nodes required will be small and hence it would be justifiable and economical to interconnect all the nodes. The fully connected network ensures reliability since an alternative line always exists in the event of a link failure and reduces the amount of buffering required. For simplicity, it is assumed that transmission lines are unidirectional so that any two nodes must be connected by at least one pair of lines, and that they are capable of transmitting at rates up to 10 megabits per second. It can be seen from Figure 3.9 that no terminal is directly connected to a node. It is assumed that a microprocessor host will be designed to interconnect the terminal to the network. The microprocessor host would provide basic editing facilities to enable more economic use of the network.

Figure 3.10 shows the basic node architecture. Each node will comprise of at least 2 processors (for "fail-soft" capability), a number of memory modules each capable of storing one packet of information, line buffers of one packet capacity for node/node and node/host communication, and a limited amount of memory for control purposes which is commonly available to all the processors but distinct from the packet storage memory.

A multibus, essentially a cross-bar bus system, is used to connect buffers, packet memory and processors. The multibus

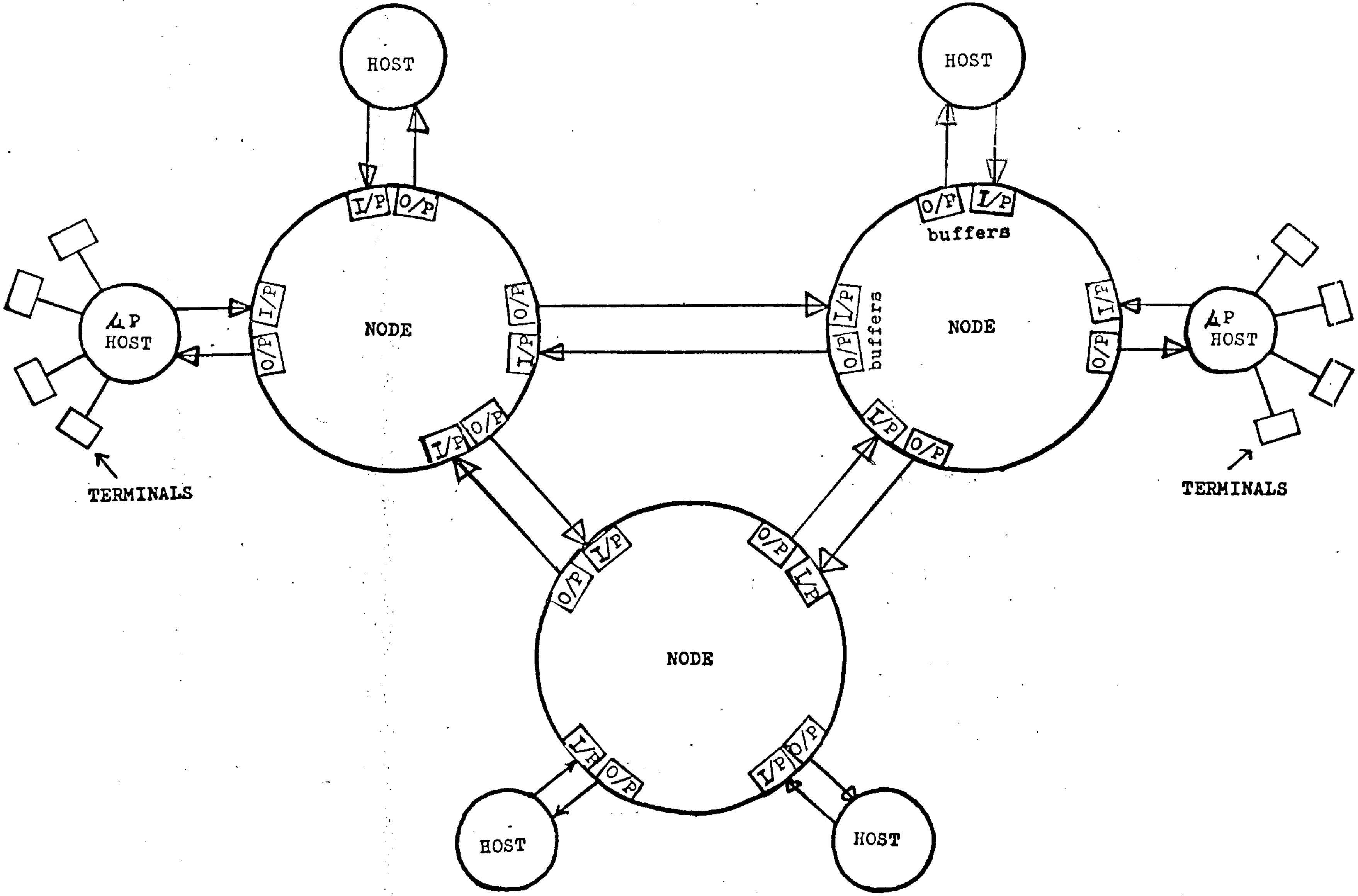


Figure 3.9 Simple Network Layout

is organised in such a way that each processor can have a source and a destination for its data independently of any other processor. In such a way processors are able to transfer data from an input buffer to memory or from memory to an outpu t buffer. The data path defining source and destination would be set up by the command processor which would make the appropriate settings in the multibus control registers.

Although all processors will be controlled by identical microcode, at any given time one processor will be in command and responsible for assigning packet handling tasks to the slave processors, maintaining status and control information, and controlling use of the multibus by exclusive access to the multibus control registers. To ensure that packet wait time and slave processor idle time are minimised the command processor (CP) will be primarily concerned with the control process. However the command processor should also be able to perform packet handling routines.

The processor in command is determined by the contents of a status register which is initially set at startup with the identification of the first logical processor. The status register controls access to the multibus control registers and routes interrupts from slave processors to the command processor.

Interprocessor communication is limited to a command/slave

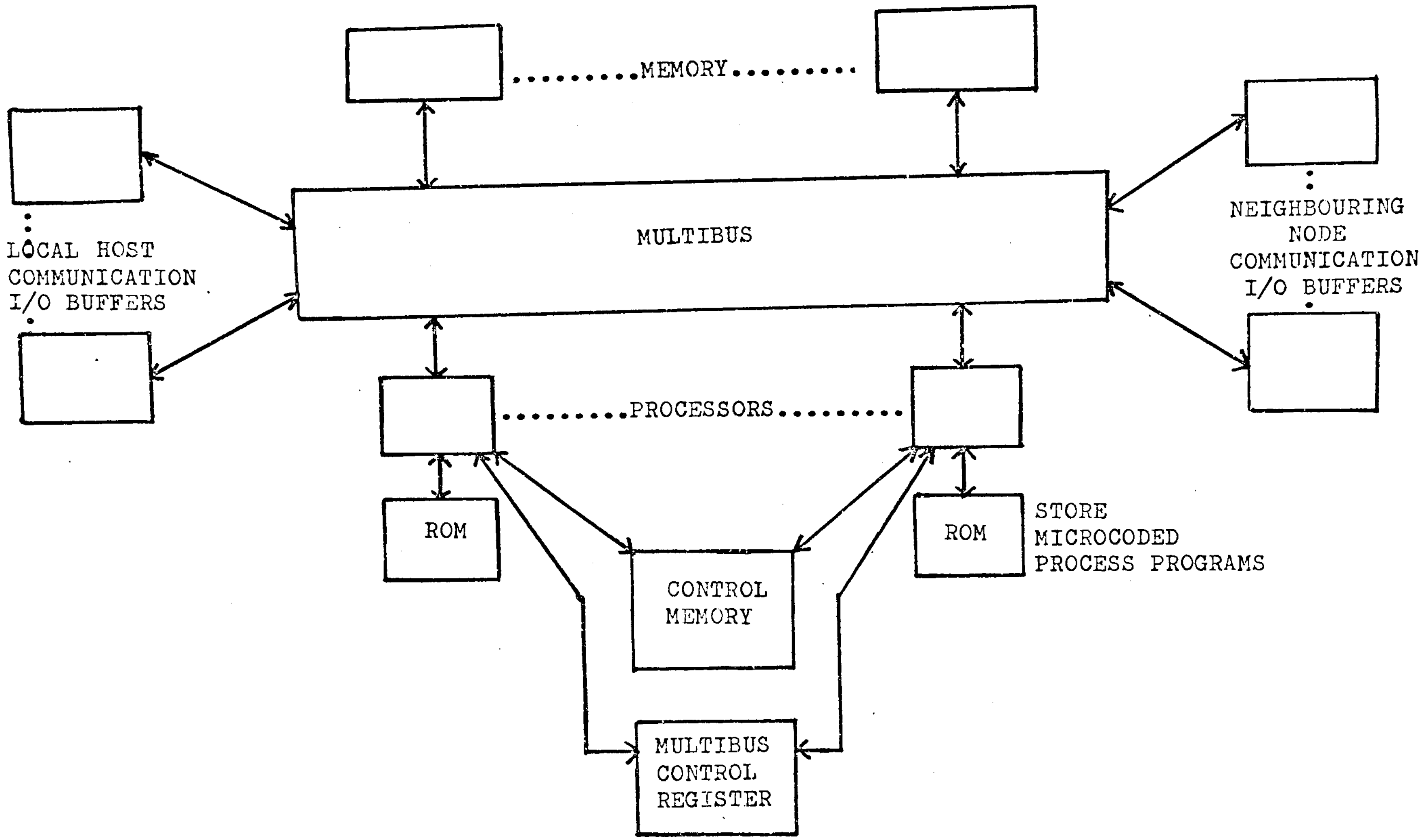


Figure 3.10 Basic Node Architecture

dialogue, which is initiated by an interrupt from the slave processor. Upon completion of a routine the slave will interrupt the command processor which then has to allocate appropriate tasks. The provision of a set of unique locations in the control memory for each processor will enable the command processor to specify the entry point of the new routine to the slave processor. No other information needs to be passed to the slave since the memory module and buffers to be used are determined by the multibus control registers and the slave requires no knowledge of these.

To indicate completion of a routine the slave inserts a unique bit pattern in its control memory location and interrupts the command processor, which determinesthe source of the interrupt by examining the control memory. The slave would periodically check its control memory location for the start address of a new task. If the command processor makes no attempt to service the slave and change the deposited bit pattern the slave assumes that a malfunction has occurred possibly of the command processor and will attempt to carry out a diagnostic check.

The slave attempts to gain control of the command process by resetting the status register. Since more than one processor may be in contention for access to the status register, the 'logically nearest' processor is given control.

3.5 Node functions

In order to satisfy the basic design aim of simplicity and low cost the node functions must be kept to the bare minimum. This is further necessitated by the limited amount of ROM that will be available due to cost. The node's operational software will have to be micro-coded into approximately 256-512 words.

It was decided to restrict the number of packets in the network by permitting each host to have one packet on the network at any time. Kleinrock [12] observed that the vast majority of messages on ARPA were single-packet messages and questioned the wisdom of providing the sophisticated mechanisms for handling multi-packet messages.

The hosts are given the responsibility for breaking up messages into packets, formatting the packets, sequencing and reconstructing messages - although the last two functions are automatically taken care of by only permitting one packet per host on the system at any time. This approach avoids reassembly lockup, avoids reserving memory for message reconstruction and simplifies routing. Routing simplification arises by virtue of the fact that the network is fully connected. However, should a node, line or buffer develop a fault and a direct link is not available, a simple random routing algorithm would suffice to transmit the packet to its destination via an alternative route. This approach removes a lot of traffic from the network that would have to be generated in the case of providing a sophisticated routing algorithm needing frequent queue/status information from other nodes.

The node operation can be split into several distinct processes which provide the basic unit of work for the slave processors. The four processes are:

1) Scanning the input buffers from local hosts.

2) Scanning the input buffers from neighbouring nodes.

3) Servicing the node/host output buffers.

4) Servicing the node/node output buffers.

Processes (1) and (2) are basically similar as are processes (3) and (4). Carrying out the first task requires checking the destination address of the packet to ensure that the specified host is in fact connected to the network. Once a packet has entered the network the malfunction of the destination host will be detected either by a control packet broadcasting the fact or by the failure of the source to receive a 'send next packet' after a pre-set time.

Incoming packets will be one of three types: information packets needing an acknowledgement, acknowledgements which are not retransmitted, and control packets which require acknowledgement and some additional action on the part of the command process.

After being error checked the incoming packet header is examined to determine the packet type. The packet is then placed into the memory module reserved for the input process and tagged to indicate 'for the attention of the command process' in the case of a control packet, 'message packet', or 'memory free' in the case of an acknowledgement.

The last type of packet requires the output buffer holding the acknowledged packet to be released. To avoid storing the acknowledgement for examination by another process, the input buffer scan must carry out this action itself. This may be achieved explicitly by specifying in the packet header the buffer through which the packet was transmitted. Alternatively, at system start-up the command process can derive a table of buffer pairings by sending appropriate control packets, thus allowing the input buffer scan to determine the buffer to be freed.

Prior to a command processor assigning a task it must establish a data path for the slave e.g. input buffer to memory, or memory to output buffer. The input scan could be programmed to determine that the packet being currently handled was not destined for its node and that it needs to be output. Being the input scan the data path that would be established is the input buffer to memory. In order to carry out the task efficiently, the packet needs to be placed in an output buffer. This requires a data path from the input buffer to the relevant output buffer. This necessitates the slave interrupting the command processor requesting the desired data path. This would greatly increase the microcode although it would be more efficient than transferring the packet to memory for another process to handle. However, since there is no guarantee that the desired data path will be available it seems more logical to bring the packet into memory.

Servicing output buffers requires packets to be transferred from memory to the appropriate output buffer and transmission to be initiated. Since the buffer freeing scheme mentioned earlier requires the buffer number to be included in the header, it is necessary for the output process to generate the packet check bits.

After transferring a packet to the output buffer, the memory module is set free by amending appropriate control structures. An attempt to transmit a packet over a particular line several times without receiving an acknowledgement would result in the packet being brought back into memory or it could be transferred directly to another output buffer and another route tried.

Acknowledgement generation may be regarded as a separate task for the reasons mentioned previously regarding the need for an interrupt requesting the required data path.

The function of the command process is to manage the operation of the other processes. Only the command process may allocate resources, although other processes may release resources allocated to them. There are two methods in which the command process may assign tasks: searching and queueing. By searching for tasks the control process is simplified bearing in mind that the search would be very simple to carry out. It may seem more efficient to provide a queue for tasks but problems may arise in determinig the maximum number of tasks which the queue may accommodate.

Although the primary function of the command processor is to initiate packet handling routines, it must also be able to carry out checks on memory, buffers and lines. Given the restriction imposed on one way command-slave dialogue, a corrupt processor cannot be stopped until it interrupts the command processor. However, the corrupt processor may be isolated by use of the multiuus registers. Should an assigned memory module or buffer cease to function correctly, the slave would be unable to carry out its task and may enter a loop awaiting completion of a memory or buffer transfer. After a corrupt processor has interrupted the command processor, diagnostic tests would be initiated on the processor and all the hardware assigned to it. Alternatively the slave could keep a table of the task hardware resources accessed and pass the information back to the command processor.

Buffers and lines are automatically checked every time a packet is successfully transfered over a particular link. However, a table of last transmission needs to be kept and if necessary dummy packets sent every second, say, as idle traffic to ensure that the line/buffers are in fact still working correctly. After, say, three attempts at sending out a packet, the line/buffers are assumed down. Periodically, the line would be retested and after, say, 25 successful consecutive transmissions the line/ buffers would be assumed to be clear of the fault.

Memory, if not used for a certain period, could be tested with a random pattern. Failure of the test would involve the memory module being only temporarily discarded, since the fault could be intermittent/transient. Further checks would be carried out periodically and if the check is satisfied the memory module would be reinstated.

Both initial setup and recovery are similar operations requiring the creation of a table/list of resources available - processors, memory, buffers and the state of neighbouring nodes and hosts on the network. The latter can be achieved by polling hosts and nodes to check their state. Polling of the nodes and hosts will also determine buffer pairings.

CHAPTER 4

NETWORK MODEL AND SIMULATION PROGRAM

4.1 Introduction

This chapter describes the overall structure of the network model and its implementation as a computer program. The major modelling effort is concentrated on the study of the behaviour of messages queueing for access to the network.

The model accommodates an arbitrary number of nodes and hosts for a symmetric traffic pattern i.e. the traffic destinations are equally distributed. For simplicity, the message arrival rates at all hosts are stationary and uniformly distributed. The overall arrival rate is assumed to be Poisson with a fraction μ of short messages and the remaining $(1 - \mu)$ messages being longer multi-packet messages.

The latter part of the chapter is concerned with the implementation of the model as a computer program. A description is given of the 'Top Hat' method of generating exponential random numbers which is used in the simulation. The chapter finally discusses the output of the simulation program.

4.2 Level of Simulation

A decision has to be made regarding the level of simulation required. It is clear that the more complicated the model the longer it will take to execute the run. Therefore, a compromise has to be made between the complexity of the model and the time that the model will take to run. This involves developing the least complicated model to obtain the required results.

There are a wide range of simulation levels which may be adopted with computer networks ranging from logic elements to nodes. The limiting factors of the network are at the level of line speeds, memory/processor speeds, I/O buffers etc. Several simplifications were carried out regarding the network philosophy discussed in the previous chapter.

It was stated that a control processor controlled all resources and task allocation. In order to cut down the execution time of the model it is assumed that there is no master processor, only slave processors. This can be justified on the basis that the controlling processor task will be fairly small. However, in order to simulate the control process of assigning resources, tasks and data paths would place a great burden on the model. A queue of free processors is maintained from which the first processor is assigned whenever a task needs servicing.

Another simplification was carried out regarding the data paths. In the assignment of tasks the slave processor also required a data path, say, node input buffer → memory module. In the model when a processor takes on a task, it has access to all resources, assuming that all these data paths exist. It is not assumed that memory is free. In this way all input buffers are handled by one routine and the simulation of interrupts to the control processor is avoided. When a processor scans all input buffers and none needs servicing then the processor remains in the free list since no work has been done.

Acknowledgements are not handled as a separate process. Suppose as before that a message packet is transferred from an input buffer to memory, an acknowledgement is generated and placed in the control packet queue by the input buffer scan routine.

Line propogation time delays for ARPA [4,10] are given as approximately 10 μsecs per mile giving a figure of 20 msecs for a 3000 mile channel at 50 kilobits/sec. On campus the distances would be less than $\frac{1}{4}$ mile and line speeds would be operating at speeds up to 10 megabits/sec. Line propogation delays are therefore considered negligable and ignored.

4.3 Time mechanism

The method used in the computer model to move the system being simulated through time is of great importance. There are two general types of methods available to move the model through time: fixed-time increment methods and variable-time increment methods.

Fixed time-increment methods need the computer to simulate a "clock" which records the instant of real time that has been reached in the system to maintain the correct time sequence of events. The time indicated by the "clock" is referred to as "clock time." The clock is updated in uniform discrete intervals of time, e.g. hours, minutes, seconds etc. The system is then scanned or examined every unit of clock time to determine whether there are any events due to occur at that particular clock time.

With variable-time increment the event tables are scanned for the next event and the clock time is then advanced by the amount necessary to cause the next most imminent event to take place. This enables events to occur whenever desired in clock time because time is advanced by variable increments rather than being divided into a sequence of uniform increments. After the execution of all events that occur at the current time indicated by the clock, the next most significant event is determined and the system moves forward again. The intervening time period when no changes occur in the system are skipped over.

Systems where the events can be expected to occur in a regular manner would be computationally more efficient with fixed time increments. Fixed time increments are also useful in the study of systems whose significant events are not well known e.g. large control systems, or the initial phase of system study.

Variable time methods lend themselves more efficiently to systems where events occur unevenly in time. The size of the unit in which clock time is measured does not affect the computational speed of the simulation. Furthermore, variable time increment methods save computer running time when the simulation is static for periods of clock time.

The simulation of the network is likely to be a long one and any saving that could be made would be worthwhile. Variable time increments were therefore chosen.

4.4 The Arrival Process

Since message arrivals cannot be deterministically predicted it is necessary to define the random arrivals by means of some probability distribution. In the network the arrival process is characterised by the way hosts generate messages. Users are independent of one another and so message arrivals are also independently distributed. If the interarrival times are exponentially distributed, it can be shown that the number of arrivals occurring in a fixed period of time can be described by the Poisson distribution. The Poisson distribution has been used by many researchers including Kleinrock [30,31,32] to describe network message arrivals.

To obtain a Poisson distribution with mean λ , advantage is taken of the relationship between the exponential and Poisson distributions. If three conditions are satisfied [44] , namely:

a) The number of arrivals in a given time period are independent of one another.

b) The probability of an arrival in the time interval t to t + Δt is approximately $\lambda \Delta t$ for all values of t.

c) The probability that more than one event takes place in the time interval t to t + $\Delta t \to 0$ as $\Delta t \to 0$

then it can be shown that the density function of the interarrival times is

$$f(t) = \lambda e^{-\lambda t}$$

and the probability of n arrivals occurring in time t is given by

$$P_n(t) = \frac{(\lambda t)^n e^{-\lambda t}}{n!}$$

The number of arrivals occurring in time t may be obtained by generating exponentially the time intervals $t_1, t_2, t_3, \ldots\ldots$ with mean $1/\lambda$. These intervals are summed until the sum exceeds λ (the Poisson Distribution mean). n, the number of arrivals is then defined by

$$\sum_{i=0}^{n} t_i \leqslant \lambda < \sum_{i=0}^{n+1} t_i \qquad (n= 0,1,2,\ldots\ldots)$$

with t_i generated from

$$t_i = - \log r_i$$

r_i being generated from a uniform random distribution.

However, since the model will be moved forward by variable time increments this method is not really convenient. It would be better to know when the next event occurs. Given the mean inter-arrival time $1/\lambda$ this may be obtained from

$$t_{next} = - \frac{1}{\lambda} \log r_i$$

There was no raw data for projected host-host traffic, therefore the network is designed under the assumption of equal traffic between all hosts.

4.5 Hyper-Exponentially Distributed Message Lengths

It was assumed for this investigation that the message lengths are Hyper-exponentially distributed since it allowed better control of the message length and took better account of shorter packets than a straight exponential in much the same manner as Coffman discussed for interarrival times for TSS[45]. The Hyper-exponential distibution is obtained by mixing together two distributions given by:

$$f(t) = \sigma.\mu_1 e^{-\mu_1 t} + (1-\sigma).\mu_2.e^{-\mu_2 t}$$

where

f(t) is the probability density function of the hyper-exponential distribution. σ is the proportion of short packets with a mean of μ_1 and $(1-\sigma)$ is the proportion of long packets with mean μ_2.

The mean value of the resultant distribution is given by

$$\int_0^\infty t.f(t).dt$$

$$= \int_0^\infty t. \left\{ \sigma \mu_1 e^{-\mu_1 t} + (1-\sigma) \mu_2 e^{-\mu_2 t} \right\} . dt$$

$$= \int_0^\infty \left[t.\sigma.\mu_1 .e^{-\mu_1 t} + t.(1-\sigma).\mu_2 e^{-\mu_2 t} \right\} . dt$$

$$= \sigma\mu_1 \left\{ \left[\frac{te^{-\mu_1 t}}{-\mu_1 t} \right]_0^\infty + \int_0^\infty \frac{e^{-\mu_1 t}}{\mu_1} . dt \right]$$

$$+ (1-\sigma)\mu_2 \left\{ \left[\frac{t.e^{-\mu_2 t}}{-\mu_2} \right]_0^\infty + \int_0^\infty \frac{e^{-\mu_2 t}}{\mu_2} dt \right\}$$

$$= \sigma\mu_1 \left[\frac{e^{-\mu_1 t}}{-\mu_1^2} \right]_0^\infty + (1-\sigma)\mu_2 \left[\frac{e^{-\mu_2 t}}{\mu_2^2} \right]_0^\infty$$

$$= \frac{\sigma\mu_1}{\mu_1^2} + \frac{(1-\sigma)\mu_2}{\mu_2^2}$$

$$= \frac{\sigma}{\mu_1} + \frac{(1-\sigma)}{\mu_2}$$

Therefore, the mean of the hyper-exponential distribution is a linear combination of the two respective means μ_1, μ_2 in the ratios σ ,$(1-\sigma)$ respectively.

4.6 Simulation Model

Packet switching communications systems have two fundamental goals in processing data - low delay and high throughput. The major modelling effort is concentrated on the study of the behaviour of messages queueing for access to the network. This seems a reasonable approach since in a packet switching network, messages experience delays as they are transmitted through the network and thus the queue lengths and speed with which messages are throughput are a reasonable performance measurement.

Basically, a set of fixed nodes is assumed located at dispersed locations on campus. Certain nodes are interconnected by transmission lines. Together the nodes and lines constitute a particular network. Connected to each node are hosts which require the use of the network.

Each node and host is assigned a unique identification number, lines being identified by the nodes and hosts that they link together. This arbitrary configuration of an N-node network is represented by several lists.

NET indicates the nodes each node connects to; NETB indicates the input buffer connected at the other end of the line. $NPLR_{ij}$ holds the time required to transmit a message packet from $node_i$ to $node_j$. Similarly NCLR holds the time required to transmit a control packet. HPLR and HCLR are the equivalent times required to transmit packets between nodes and local hosts.

Each node has a set of p processors and p+1 memory units for packet storage. Processor speed and memory access time are matched. Each memory unit is capable of holding as many packets as specified by the initial conditions.

The principle of store-and-forward should make the host-host link over several nodes invisible to the user. Queues may however build up in the network. This condition is accommodated by two queues, NBUFF and CBUFF for message packets and control packets respectively. Figure 4.1 shows their use.

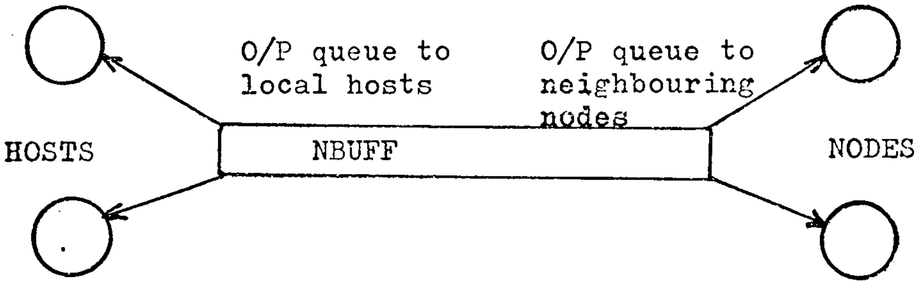


Figure 4.1 Node queue handling

Together with the packet information it is necessary to know which memory the packets are stored in. This is achieved by vector NMEM which has a 1-1 correspondence to NBUFF. Control packets are kept in a queue and not in a memory unit.

Once a packet has been generated the information about it is held in array PKT. The time a message was generated and placed ready for entry to the network is held in vector TPIN.

The information required for each packet is as follows:

1) Source Address

2) Destination Address

3) Type of Packet

   a) 0 - message

   b) 1 - local control packet, Node-Node Acknowledgement

   c) 2 - local control packet, Node-Host Acknowledgement

   d) 3 - 'send next packet'

   e) 4 - local control packet, Host-Node Acknowledgement

4) Packet Sequence Number

5) Number of Packets in a message.

Thus, when packets are moved from buffer to buffer, the only information that needs to be moved is the packet subscript pointing to the packet location in PKT.

From the information in TPIN it is possible to calculate packet and message delays experienced on the network.

4.7 Processor Busy Time

The node computer used in ARPA has a 16-bit word length and a 0.96 μsecond memory cycle time [4]. Kleinrock [10], in his analytical studies of the ARPA network assumed node processing time for all packets to be constant and quoted an operational figure of 0.35 mseconds per packet. McQuillan et al [9] quoted a figure of 550-700 cycle times per packet. It seems reasonable to tie the processor busy time to the memory access time since this will be the limiting factor in transferring a packet to/from memory and from/to a buffer. Given a message packet length of 1024 bits and a 1 µsec memory cycle time, this gives a figure of 128 µsecs to transfer the 128 x 8 bit bytes of a packet. This figure is doubled to take into account other housekeeping operations such as updating tables etc. In normal operation a message packet is transferred from I/P buffer to memory, then from memory to O/P buffer. This would give a figure of 512 µsecs for a node to handle a packet, given that a processor was available to supervise the operation when required. This brings the processing time close to the figures quoted by ARPA. The effects of the multi-processor node being investigated are hidden since the ARPA's IMP is 16 bits therefore a packet may be retrieved from memory twice as fast.

4.8 Model Routines

It can be seen from Figure 4.2 that the routines are handled serially and that four routines have been marked with an '*'. These four routines can only be carried out under processor control. If no processor is available in the node's free list then the task for that node is not carried out. These four tasks are:

1) Servicing Host Output Buffers

2) Servicing Node Output Buffers

3) Updating buffers for transmitting packets to local hosts.

4) Updating buffers for transmitting packets to neighbouring nodes.

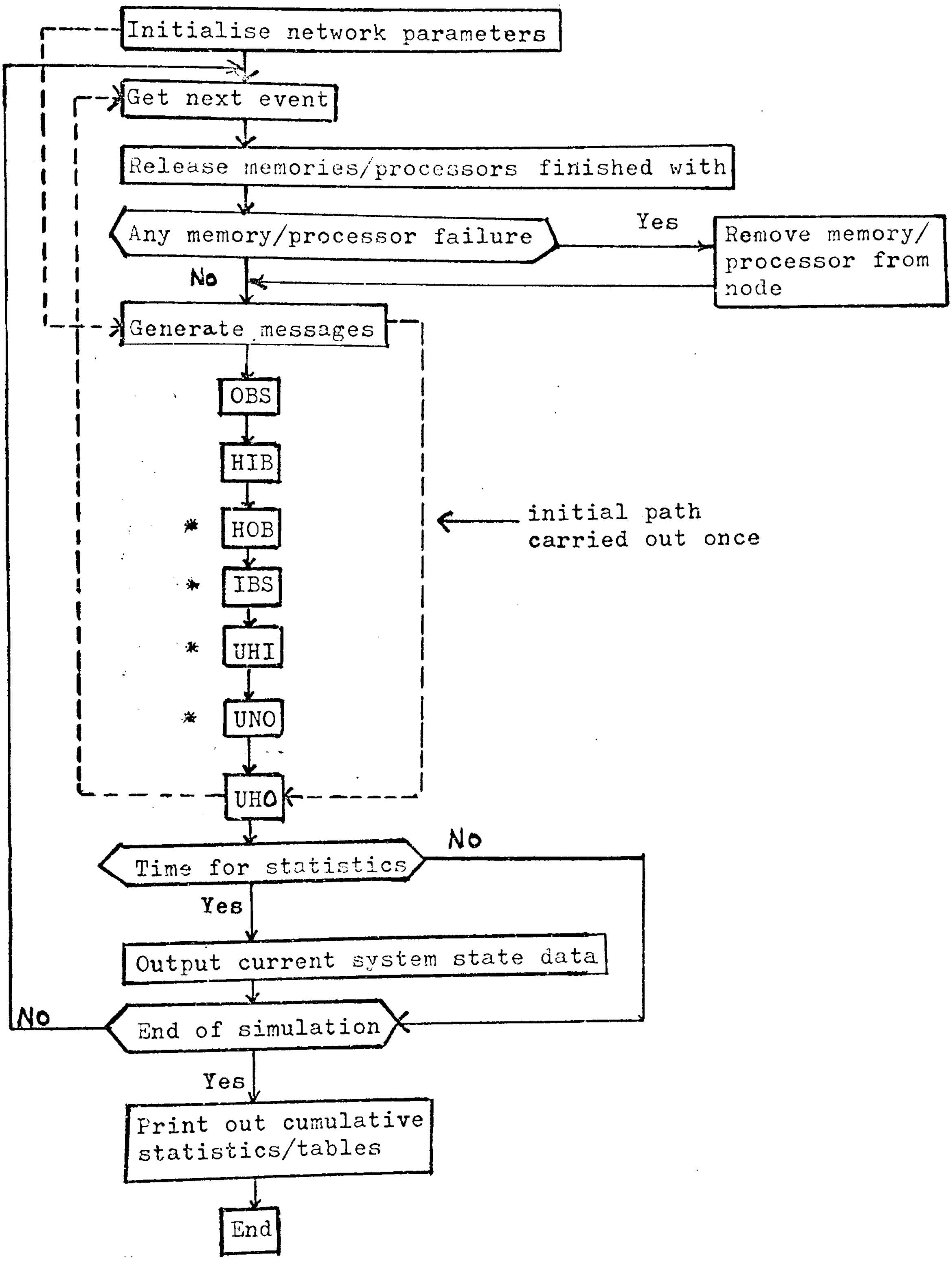

Initialise network parameters
Get next event
Release memories/processors finished with
Any memory/processor failure
Yes
Remove memory/ processor from node
No
Generate messages
OBS
HIB
*
HOB
*
IBS
*
UHI
*
UNO
UHO
initial path carried out once
Time for statistics
No
Yes
Output current system state data
No
End of simulation
Yes
Print out cumulative statistics/tables
End

Figure 4.2 Functional units of Simulation Model

Figure 4.3 illustrates the function of each of these four tasks together with three other tasks that are carried out automatically e.g. once a buffer has been filled by a processor it will empty its data onto the link without further supervision. The three automatic tasks are:

1) Transferring packets from a node output buffer to a node input buffer.

2) Transferring packets from a node host input buffer to the host.

3) Updating the host output buffers to bring packets into the network.

The initialisation of the network is carried out by DINPUT which reads in the network architecture parameters together with the parameters describing the message arrival process. The dotted path shows how the model starts itself. Messages are generated and routine Update Host Output buffers is called to place packets in the host output buffer ready to enter the network.

The simulation is table driven and a set of tables has to be scanned to determine when the next event will occur. Each of the seven tasks is carried out for each node in turn. Prior to a decision being made as to whether an event has taken place, SMALL is subtracted from non-zero event times. Buffers, memory etc. will be moved forward to the next event and at the appropriate time will be freed. Consider the routine that services the node input

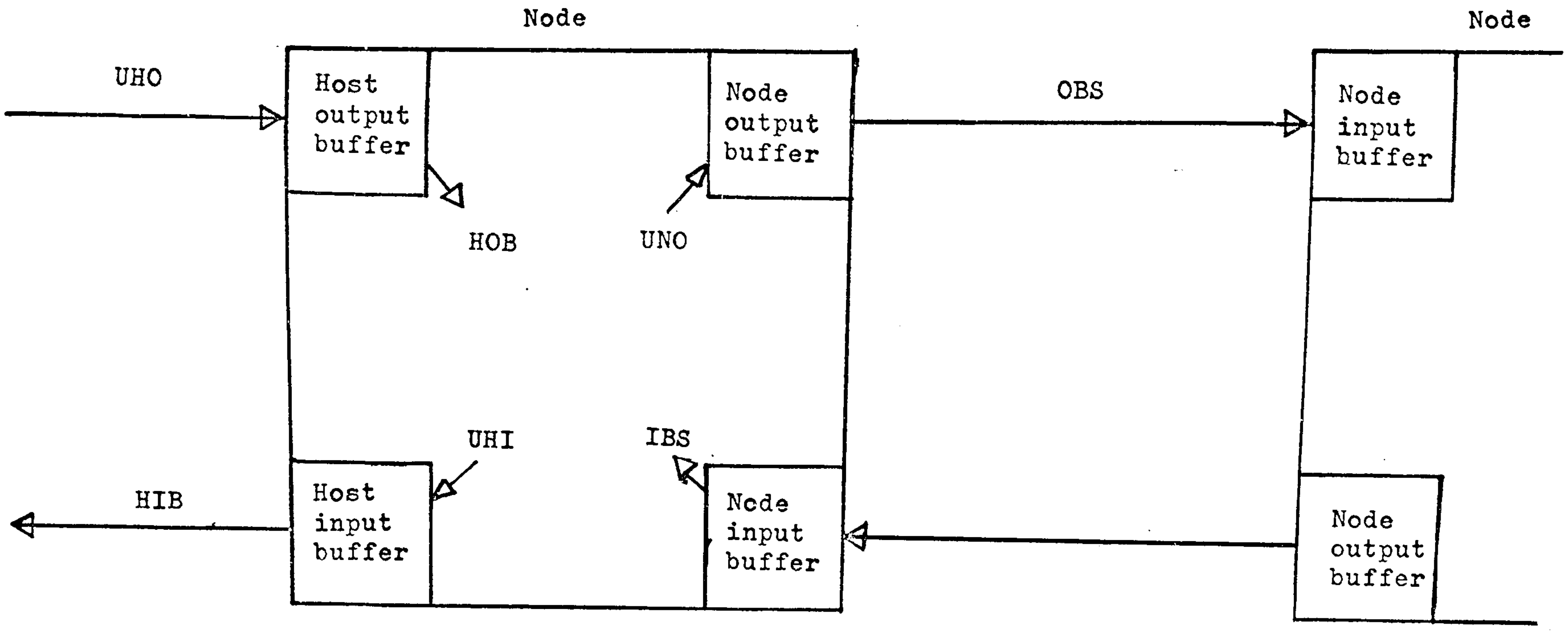


Figure 4.3 Functional Units of the Simulation Model

buffers; after servicing a buffer, the routine remains in node$_i$ and services the remaining input buffers before moving to the next task.

For each message packet or 'send next packet' the task must first determine whether sufficient buffer space exists in the node. If there is a shortage the packet remains in the buffer until space becomes available. Control packets always take priority over message packets for output. In this way freeing of buffers is achieved with the minimum time delay. The destination of a packet is always inspected to determine whether the packet is at the destination node in which case it goes into the queue for output to the local host; or whether it goes into the queue for further output to another node. Control packets are handled in a similar manner but at the destination node action is taken immediately on receipt.

Most of the routines are functionally described in the text to a sufficient degree to allow fairly easy reading of the flowcharts. Figure 4.4 shows the process of generating messages. When MESS becomes zero a message arrival is assumed to have taken place. At this moment in time however, more than one host may be generating a message. This is taken into account by sampling from an exponential distribution to give NN hosts (the mean having been specified in the initialisation of the run). The time of the next message is also exponentially generated. All the unique host identification numbers are placed in a vector which is then sampled using a uniform

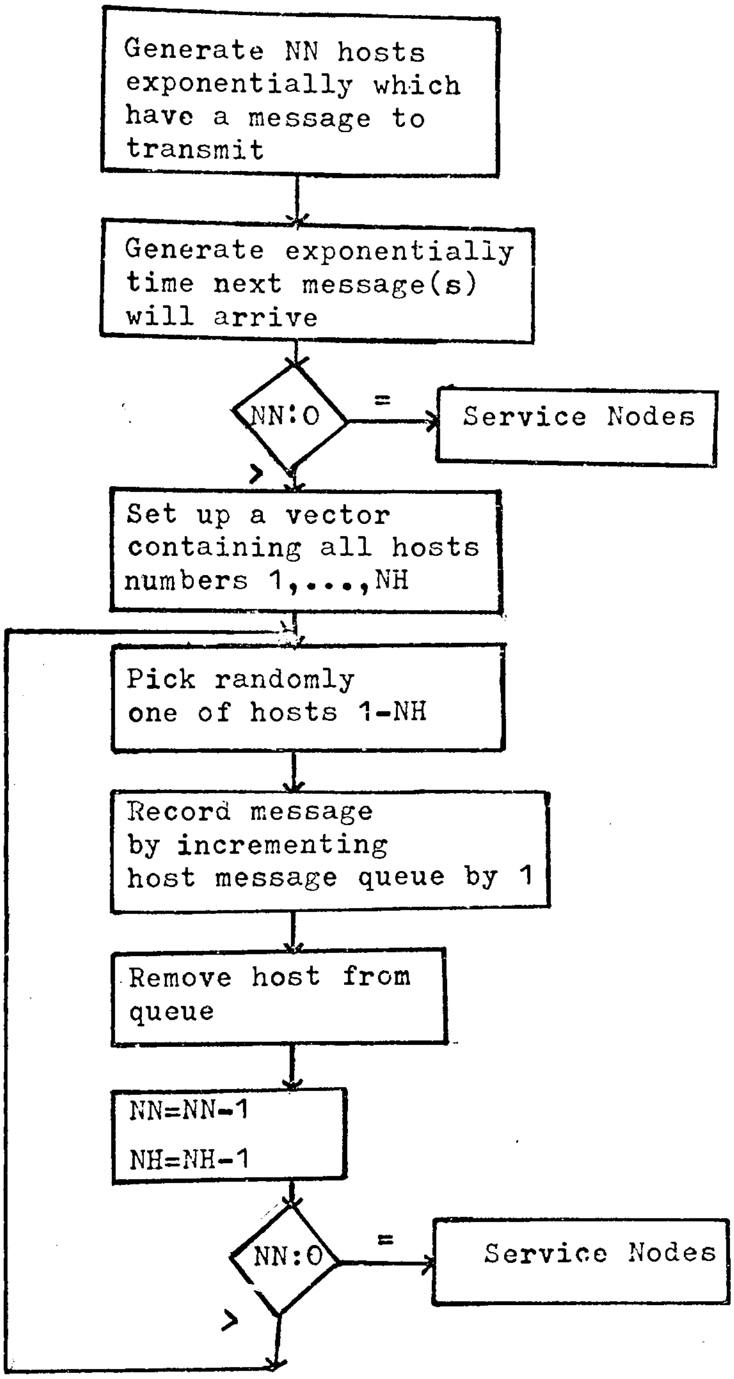

Generate NN hosts exponentially which have a message to transmit
Generate exponentially time next message(s) will arrive
NN:0
=
Service Nodes
>
Set up a vector containing all hosts numbers 1,...,NH
Pick randomly one of hosts 1-NH
Record message by incrementing host message queue by 1
Remove host from queue
NN=NN-1
NH=NH-1
NN:0
=
Service Nodes
>

Figure 4.4 Generation of Messages

distribution. A message is added to the specified host's queue and that host removed from the generation queue. The remaining hosts are sampled again in the same fashion until NN messages are generated.

The host output buffer update routine scans all the hosts to bring packets into the host buffers to enter the network. As before retransmission packets are handled first, followed by control packets then message packets. When dealing with message packets, if there are none in the queue ready for output, the message queue is checked. Should there be any new messages then packet statistics are generated after which the first packet in the sequence is placed on the queue and the event table set for the time required to transmit the packet. Figure 4.5 shows the flowchart of the routine.

The host output buffer routine is concerned with transferring packets from the host output buffer to node memory in the case of message packets or processing control packets. However, this routine may only be carried out if there is a free processor in node_i, the current node. For an acknowledgement to a previous node → host transfer the acknowledgement details are recorded, the output buffer cleared, the node host input retransmission buffer cleared of the message packet being acknowledged and the retransmission clock reset. The processor is set busy and the next buffer handled. A 'send next packet' has to be decoded to determine whether it is destined for another node or a local host. On this basis it is placed either in the CBUFF queue for

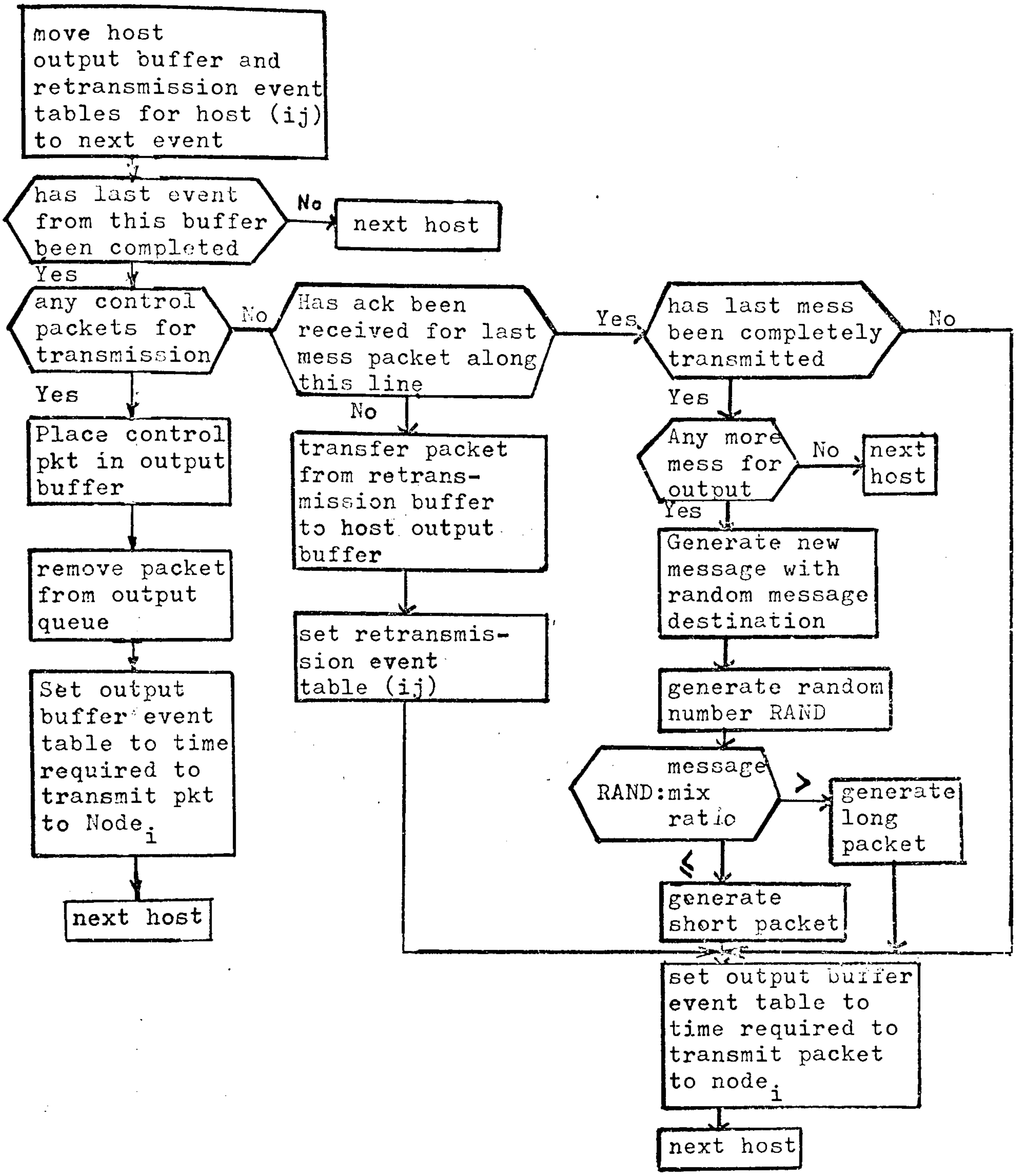

move host output buffer and retransmission event tables for host (ij) to next event
has last event from this buffer been completed
No
next host
Yes
any control packets for transmission
No
Has ack been received for last mess packet along this line
Yes
has last mess been completely transmitted
No
Yes
Place control pkt in output buffer
No
transfer packet from retrans-mission buffer to host output buffer
Yes
Any more mess for output
No
next host
Yes
remove packet from output queue
Generate new message with random message destination
set retransmis-sion event table (ij)
generate random number RAND
Set output buffer event table to time required to transmit pkt to Node$_i$
RAND:message mix ratio
>
generate long packet
<
generate short packet
next host
set output buffer event table to time required to transmit packet to node$_i$
next host

Figure 4.5 Update host output buffer - node(i), host (j)

output to another node or placed in the queue for output to a local host. The processor is set busy for a further period and the next buffer handled. As before the output buffer is cleared. If a message packet is received, a free memory unit has to be found in which to store it. If no memory unit is free the buffer retains the packet until a memory unit becomes free. Should a memory unit be free, the node decides from the header information whether the packet is at the destination node or not. The packet is placed in the NBUFF queue either to be sent to another node or for transmission to a local host. The memory unit is set busy and the processor set for a further period. The next buffer is serviced after an acknowledgement has been generated. Figure 4.6 shows this process.

Updating the node output buffers involves bringing packets into the node output buffers for transmission to other nodes and requires processor control. For each retransmission buffer containing a message packet, the retransmission clock is checked. If retransmission is required the output buffer event table and clock are set. Should any buffers not be in the process of transmission the output queues are searched for control packets which require a particular buffer for output. Finally, should any buffers be empty and there be no control packets to transmit, message packets are searched for in the node output queue (NBUFF). When a packet is found a check is carried out on the memory module storing it. If the memory is free, the packet is transferred to the buffer ; then the memory is set busy as is the processor. Should there be no packets requiring transmission directly to

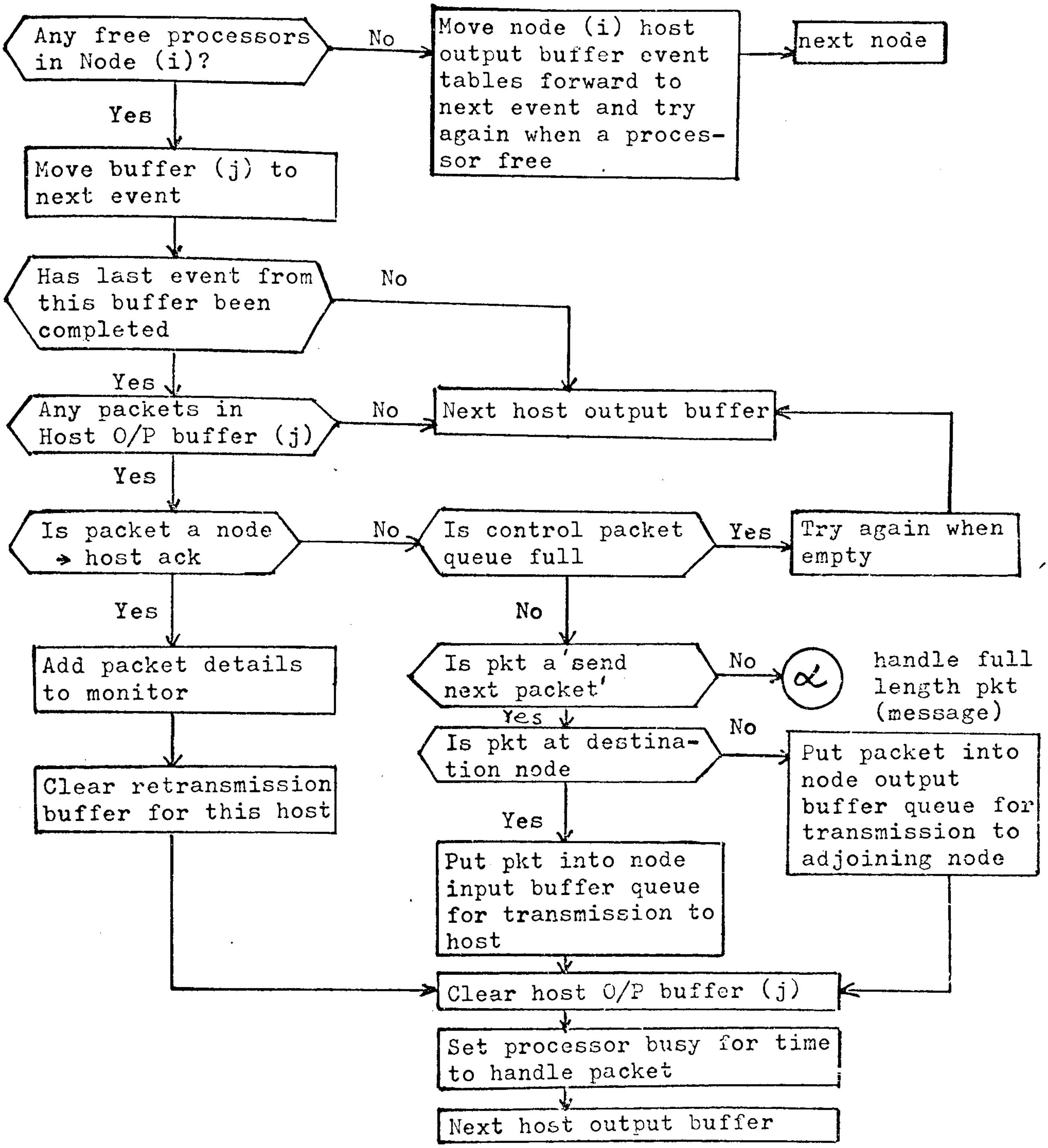

Any free processors in Node (i)?
No
Move node (i) host output buffer event tables forward to next event and try again when a processor free
next node
Yes
Move buffer (j) to next event
Has last event from this buffer been completed
No
Yes
Any packets in Host O/P buffer (j)
No
Next host output buffer
Yes
Is packet a node → host ack
No
Is control packet queue full
Yes
Try again when empty
Yes
No
Add packet details to monitor
Is pkt a 'send next packet'
No
α
handle full length pkt (message)
Yes
Is pkt at destination node
No
Put packet into node output buffer queue for transmission to adjoining node
Clear retransmission buffer for this host
Yes
Put pkt into node input buffer queue for transmission to host
Clear host O/P buffer (j)
Set processor busy for time to handle packet
Next host output buffer

Figure 4.6(i) Input packet from host (j) to node (i)

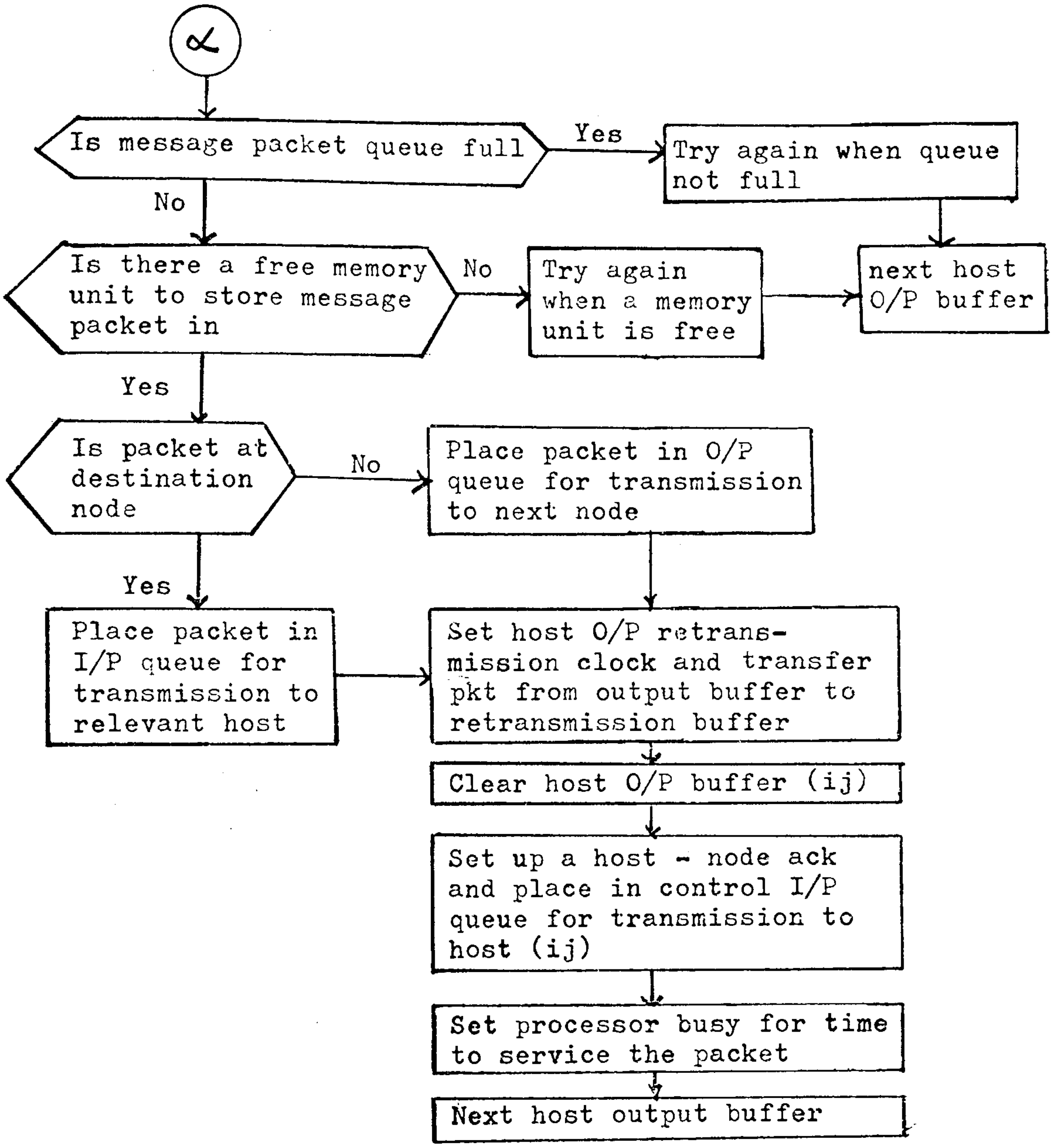

α
Is message packet queue full
Yes
Try again when queue not full
No
Is there a free memory unit to store message packet in
No
Try again when a memory unit is free
next host O/P buffer
Yes
Is packet at destination node
No
Place packet in O/P queue for transmission to next node
Yes
Place packet in I/P queue for transmission to relevant host
Set host O/P retransmission clock and transfer pkt from output buffer to retransmission buffer
Clear host O/P buffer (ij)
Set up a host - node ack and place in control I/P queue for transmission to host (ij)
Set processor busy for time to service the packet
Next host output buffer

Figure 4.6(ii) Input packet from host (j) to node (i) neighbouring nodes and there be free buffers remaining then remaining packets are transmitted. The packets are placed in random buffers to be sent to a random destination. Figure 4.7 shows the routine in more detail.

The node output buffer routine is concerned with the actual transmission of packets from one node to another over a transmission link. When the node output event element for buffer$_{ij}$ becomes zero the packet has been transmitted over the line. A copy of the packet subscript is taken from the output buffer and placed into the input buffer of the receiving node. In order for the input buffer to be serviced a delay of 1 µsecs is put into the node input buffer event table. The sending node sets its retransmission clock to 0.1 seconds and places a copy of the packet into the retransmission buffer. Figure 4.8 shows this routine.

Servicing the node input buffers requires processor control and the allocation of a memory unit for message packets. Node-Node acknowledgements initiate clearing of the input buffer together with the output buffer being acknowledged. The retransmission clock and buffer are reset, and the acknowledgement statistics are recorded. A 'send next packet' is stored in CBUFF in the queue for further retransmission or transmission to a local host. The input buffer is cleared. Processing memory packets requires a free memory unit to be found. If one is not found the packet remains in the buffer until a memory unit becomes free. The packet is put into NBUFF for either

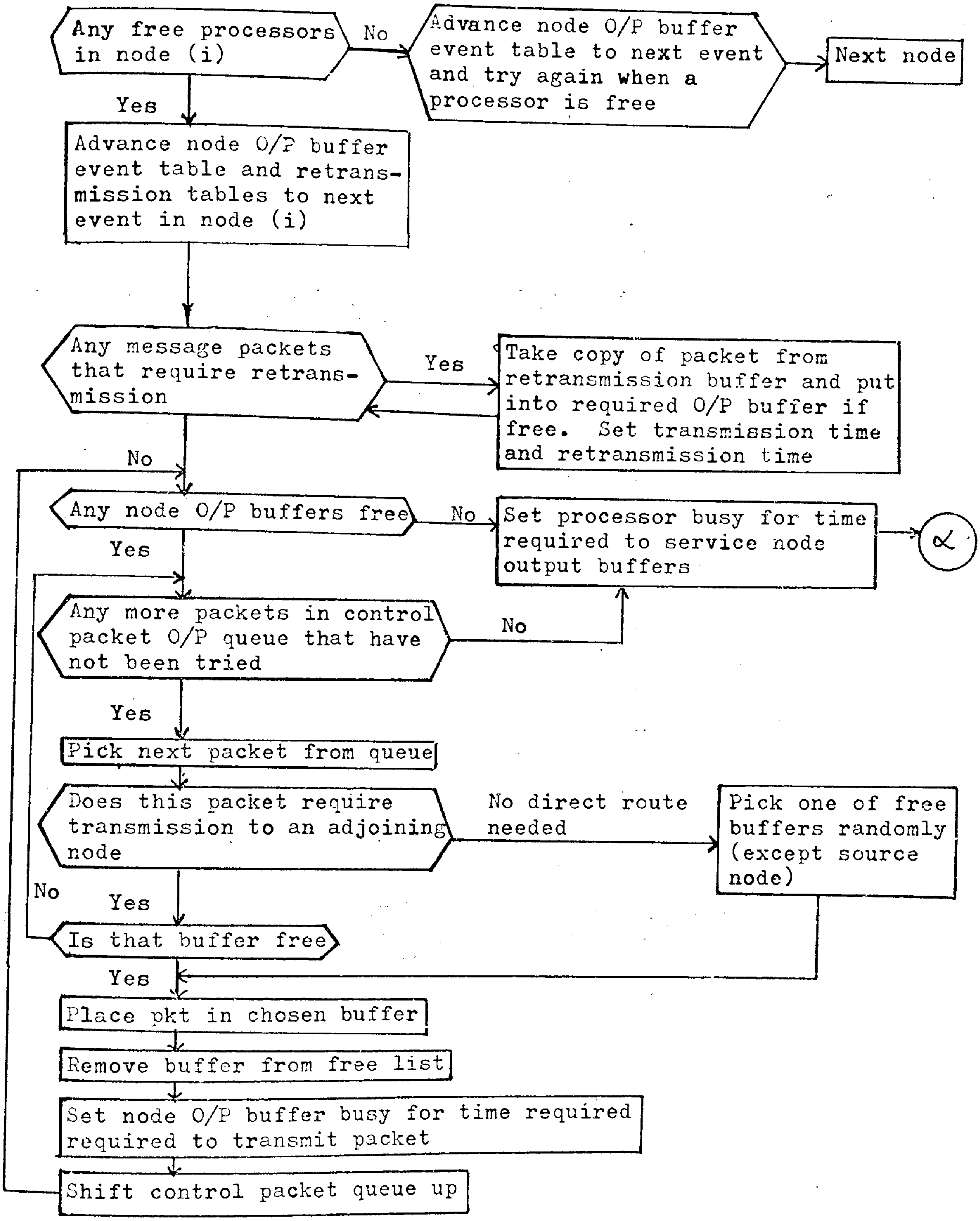

Any free processors in node (i)
No
Advance node O/P buffer event table to next event and try again when a processor is free
Next node
Yes
Advance node O/P buffer event table and retransmission tables to next event in node (i)
Any message packets that require retransmission
Yes
Take copy of packet from retransmission buffer and put into required O/P buffer if free. Set transmission time and retransmission time
No
Any node O/P buffers free
No
Set processor busy for time required to service node output buffers
α
Yes
Any more packets in control packet O/P queue that have not been tried
No
Yes
Pick next packet from queue
Does this packet require transmission to an adjoining node
No direct route needed
Pick one of free buffers randomly (except source node)
No
Yes
Is that buffer free
Yes
Place pkt in chosen buffer
Remove buffer from free list
Set node O/P buffer busy for time required required to transmit packet
Shift control packet queue up

Figure 4.7(i) Update node output buffers

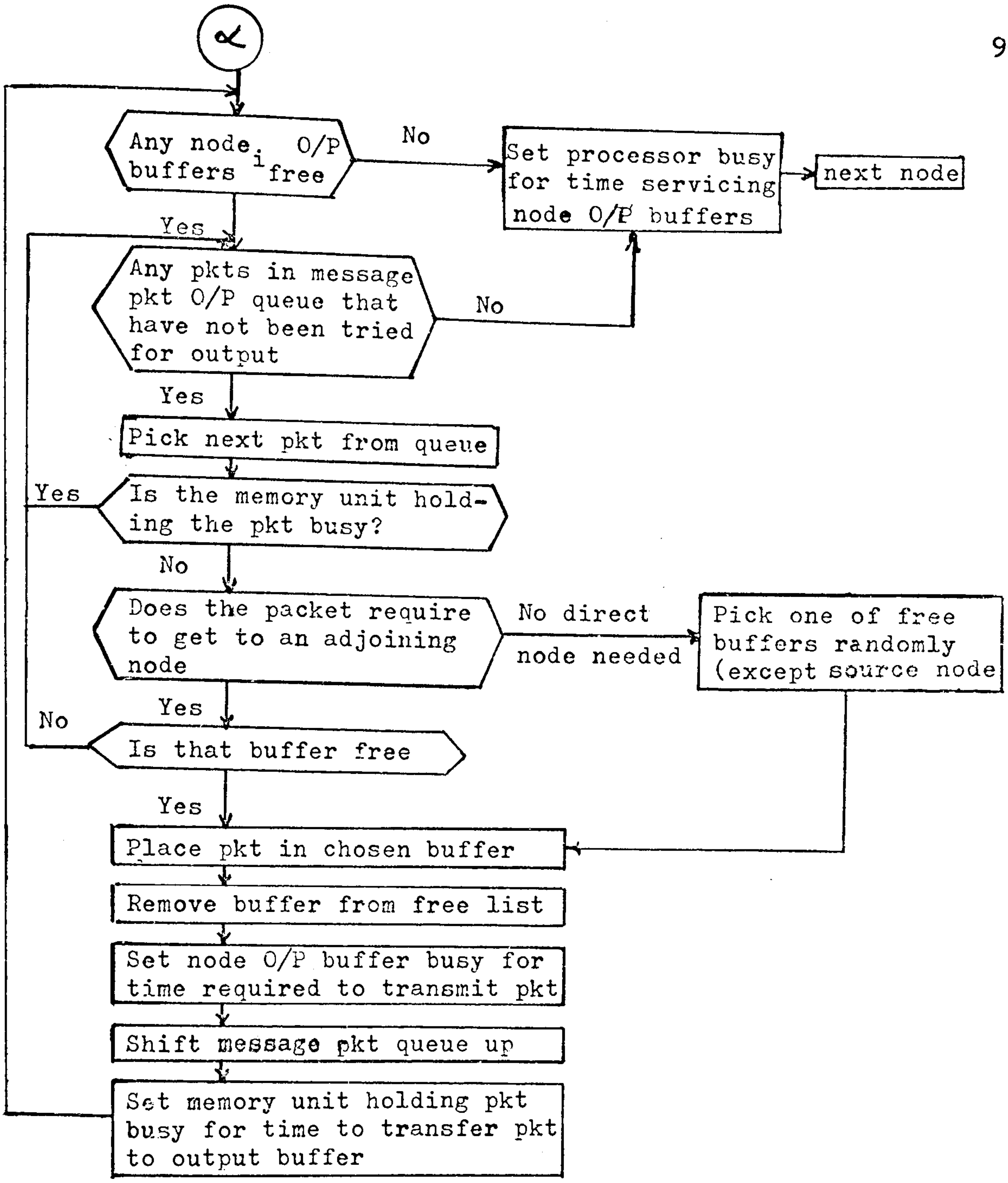

α
Any node$_i$ O/P buffers free
No
Set processor busy for time servicing node O/P buffers
next node
Yes
Any pkts in message pkt O/P queue that have not been tried for output
No
Yes
Pick next pkt from queue
Yes
Is the memory unit holding the pkt busy?
No
Does the packet require to get to an adjoining node
No direct node needed
Pick one of free buffers randomly (except source node
Yes
No
Is that buffer free
Yes
Place pkt in chosen buffer
Remove buffer from free list
Set node O/P buffer busy for time required to transmit pkt
Shift message pkt queue up
Set memory unit holding pkt busy for time to transfer pkt to output buffer

Figure 4.7(ii) Update node output buffers

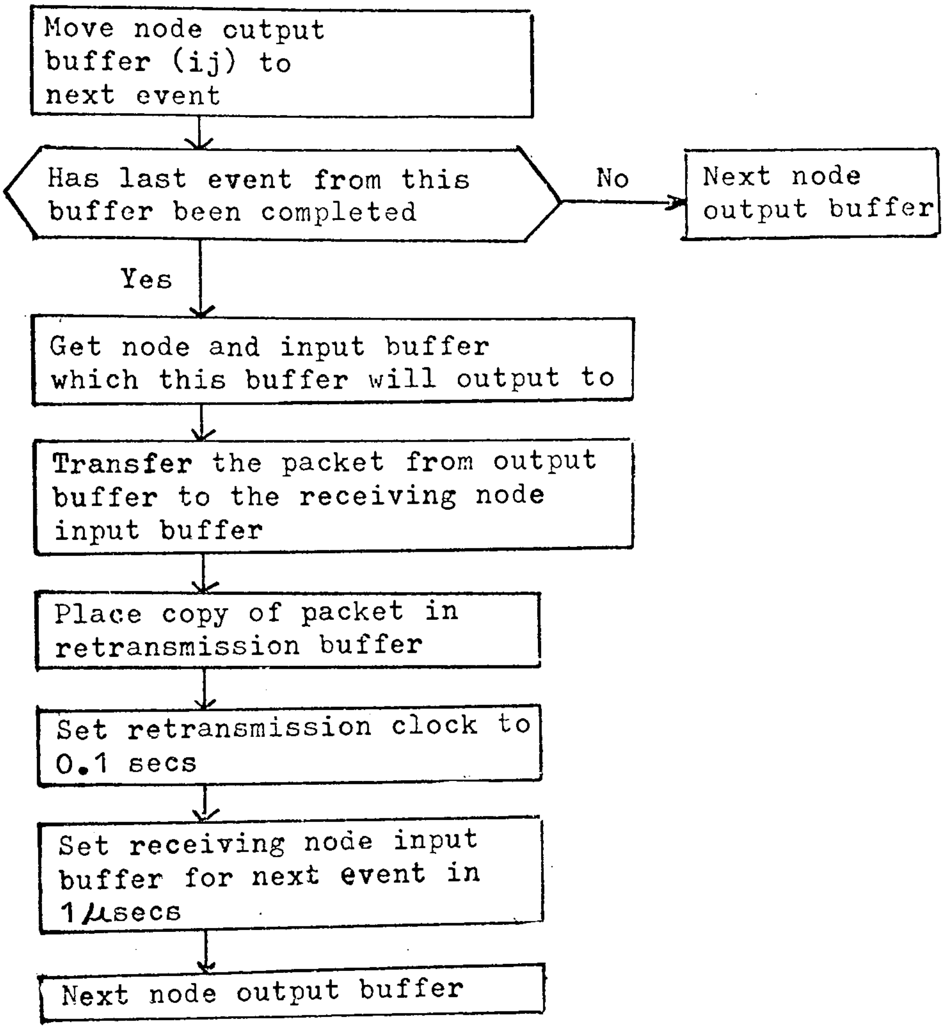

Move node output buffer (ij) to next event
Has last event from this buffer been completed
No
Next node output buffer
Yes
Get node and input buffer which this buffer will output to
Transfer the packet from output buffer to the receiving node input buffer
Place copy of packet in retransmission buffer
Set retransmission clock to 0.1 secs
Set receiving node input buffer for next event in 1 μsecs
Next node output buffer

Figure 4.8 Service node output buffer - node(i), host (j)

transmitting to another node or for output to a local host. An acknowledgement is generated and queued in CBUFF. The processor and memory are set busy for the time used. Figure 4.9 shows the flowchart of the routine.

Host input buffer update is another routine which requires processor control and is concerned with transferring packets into the host input buffers. For each free buffer a packet is searched for initially from the control packet queue (CBUFF) - since control packets have a higher priority. Should there be no control packets for this buffer, the retransmission clock is checked. If the message packet in the retransmission buffer has not been acknowledged for 0.1 seconds then the clock is set and transmission attempted again. After a copy of the packet is transferred to the output buffer. If no control packet has been allocated to the buffer and there is no message packet awaiting acknowledgement then NBUFF is scanned for a message packet destined for the local host. However, if a packet is found it can only be serviced if the memory module in which it is stored is free. Should that be the case the packet is transferred from memory to the buffer. The memory and processor are set busy. Figure 4.10 shows the flowchart of the routine.

The host input buffer scan routine is similar to the node output buffer routine, but this time packets are moved from nodes to their local hosts. When the host receives the packet it handles it according to its type. For an acknowledgement to a previous

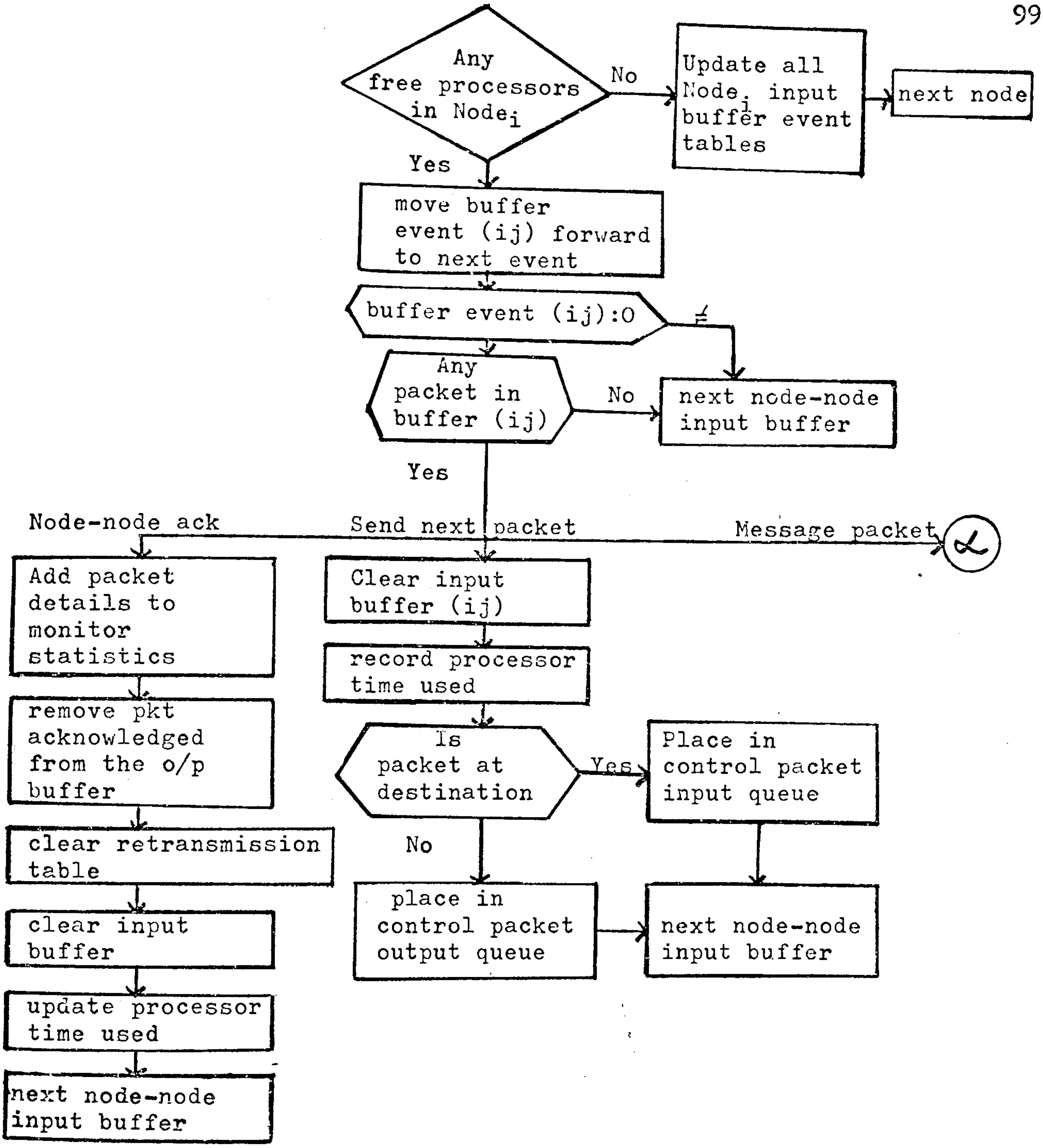
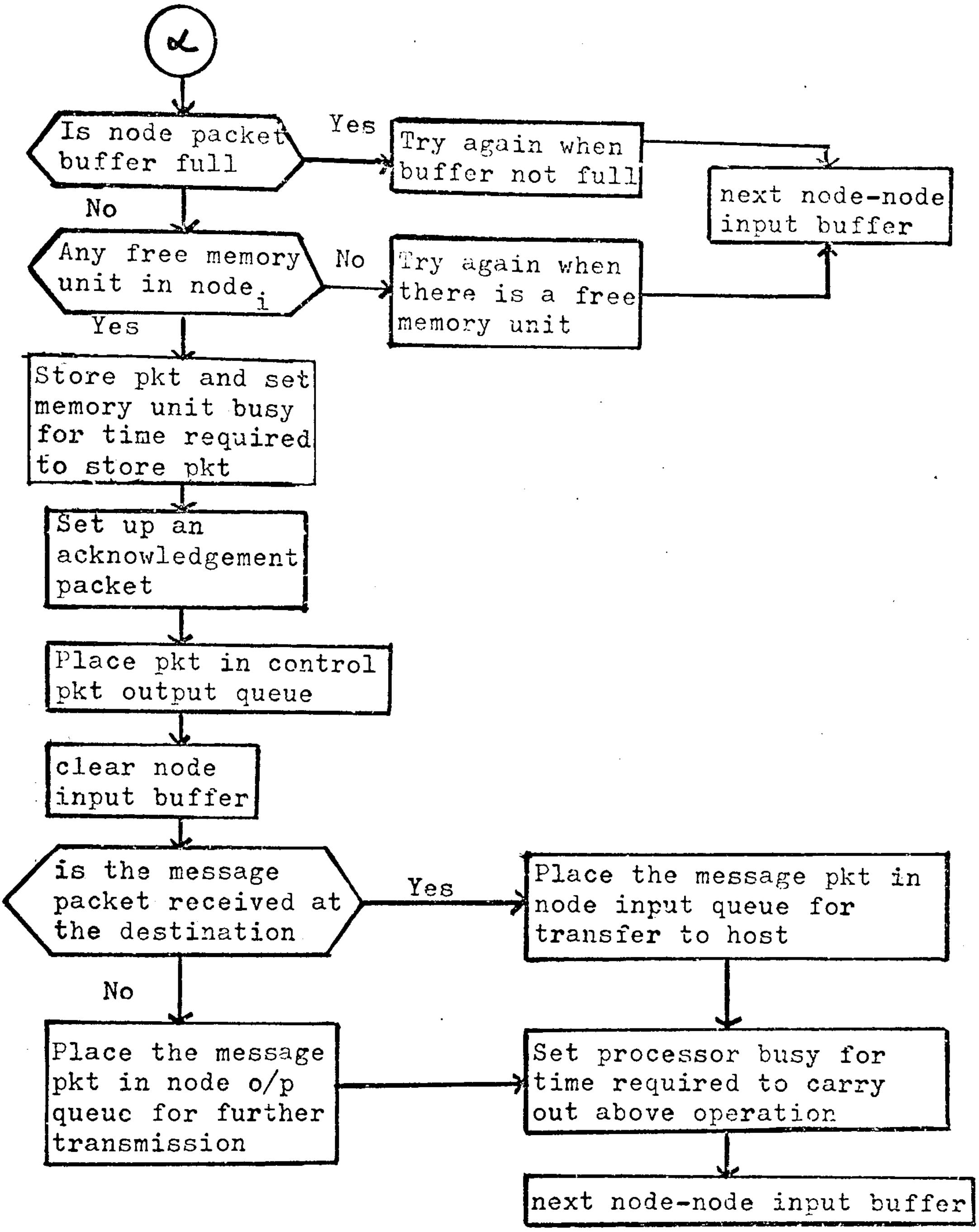

Any free processors in Node$_i$
No
Update all Node$_i$ input buffer event tables
next node
Yes
move buffer event (ij) forward to next event
buffer event (ij):0
≠
Any packet in buffer (ij)
No
next node-node input buffer
Yes
Node-node ack
Send next packet
Message packet
α
Add packet details to monitor statistics
remove pkt acknowledged from the o/p buffer
clear retransmission table
clear input buffer
update processor time used
next node-node input buffer
Clear input buffer (ij)
record processor time used
Is packet at destination
Yes
Place in control packet input queue
No
place in control packet output queue
next node-node input buffer

Figure 4.9(i) Node-node input buffer service - node(i), host (j)



α
Is node packet buffer full
Yes
Try again when buffer not full
next node-node input buffer
No
Any free memory unit in node$_i$
No
Try again when there is a free memory unit
Yes
Store pkt and set memory unit busy for time required to store pkt
Set up an acknowledgement packet
Place pkt in control pkt output queue
clear node input buffer
is the message packet received at the destination
Yes
Place the message pkt in node input queue for transfer to host
No
Place the message pkt in node o/p queuc for further transmission
Set processor busy for time required to carry out above operation
next node-node input buffer

Figure 4.9(ii) Node-node input buffer service - node(i), host (j)

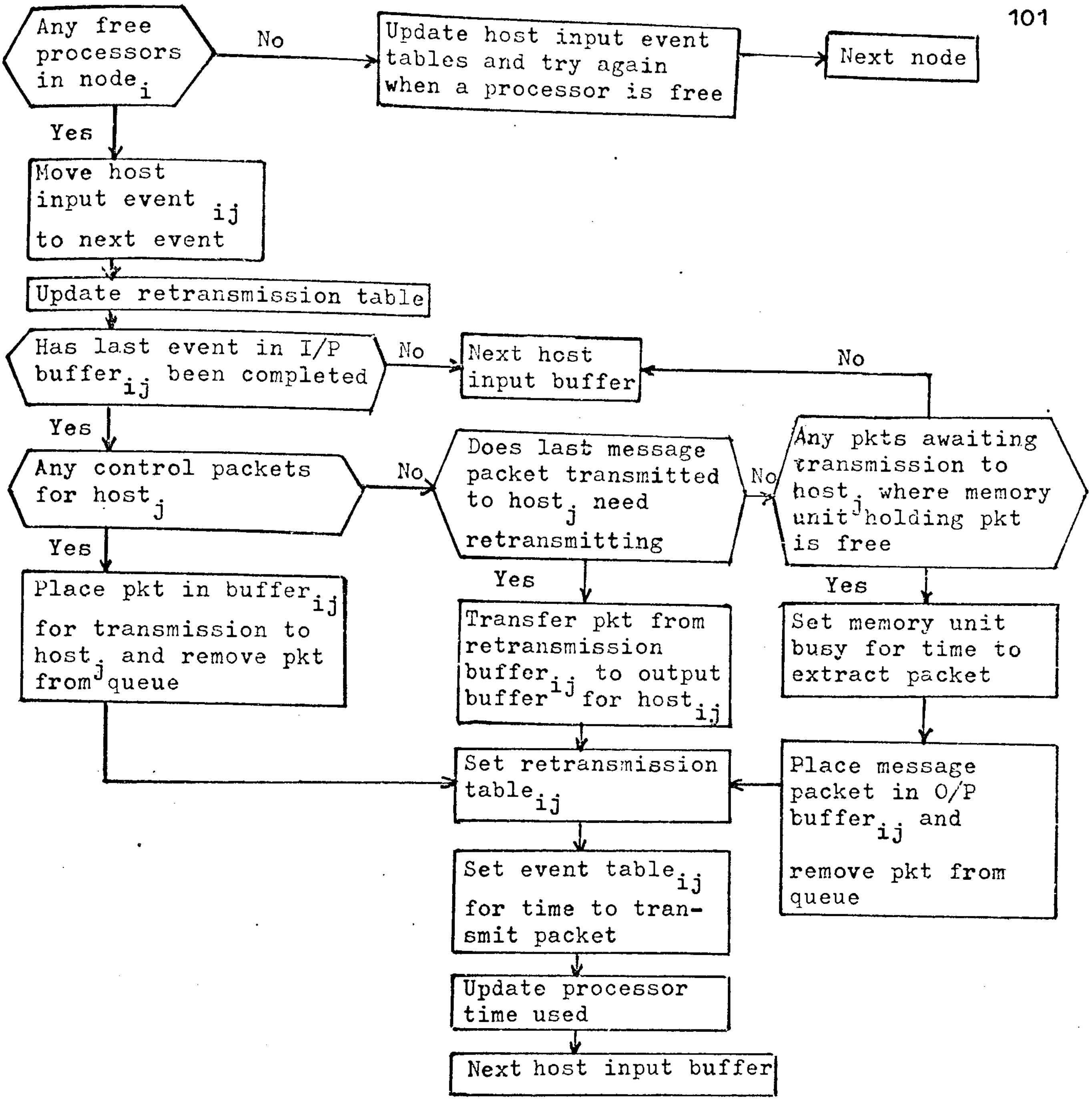

Any free processors in node$_i$
No
Update host input event tables and try again when a processor is free
Next node
Yes
Move host input event$_{ij}$ to next event
Update retransmission table
Has last event in I/P buffer$_{ij}$ been completed
No
Next host input buffer
No
Yes
Any control packets for host$_j$
No
Does last message packet transmitted to host$_j$ need retransmitting
No
Any pkts awaiting transmission to host$_j$ where memory unit holding pkt is free
Yes
Place pkt in buffer$_{ij}$ for transmission to host$_j$ and remove pkt from queue
Yes
Transfer pkt from retransmission buffer$_{ij}$ to output buffer$_{ij}$ for host$_{ij}$
Yes
Set memory unit busy for time to extract packet
Set retransmission table$_{ij}$
Place message packet in O/P buffer$_{ij}$ and remove pkt from queue
Set event table$_{ij}$ for time to transmit packet
Update processor time used
Next host input buffer

Figure 4.10 Update host input buffer - node(i), host (j)

host → node message packet transfer, the control packet statistics are recorded and the packet deleted. The host input buffer in the node is cleared and the next buffer serviced. For a 'send next packet', the nodes host input buffer is cleared and the host output buffer and retransmission clock corresponding to the sent packet cleared. The control packet is recorded. The host then has to decide from the header whether all the packets of the message have been transmitted. If not the next packet in the sequence is placed in the queue for output. If the entire message has been output and there are no more messages the next buffer is handled. Otherwise a new message is generated and the first packet in the sequence placed in the host output queue. Finally, a message packet may be received. This requires an acknowledgement packet and a 'send next packet' to be generated and be placed in the host output queue. The next buffer is then handled. Figure 4.11 shows the flowchart of this routine.

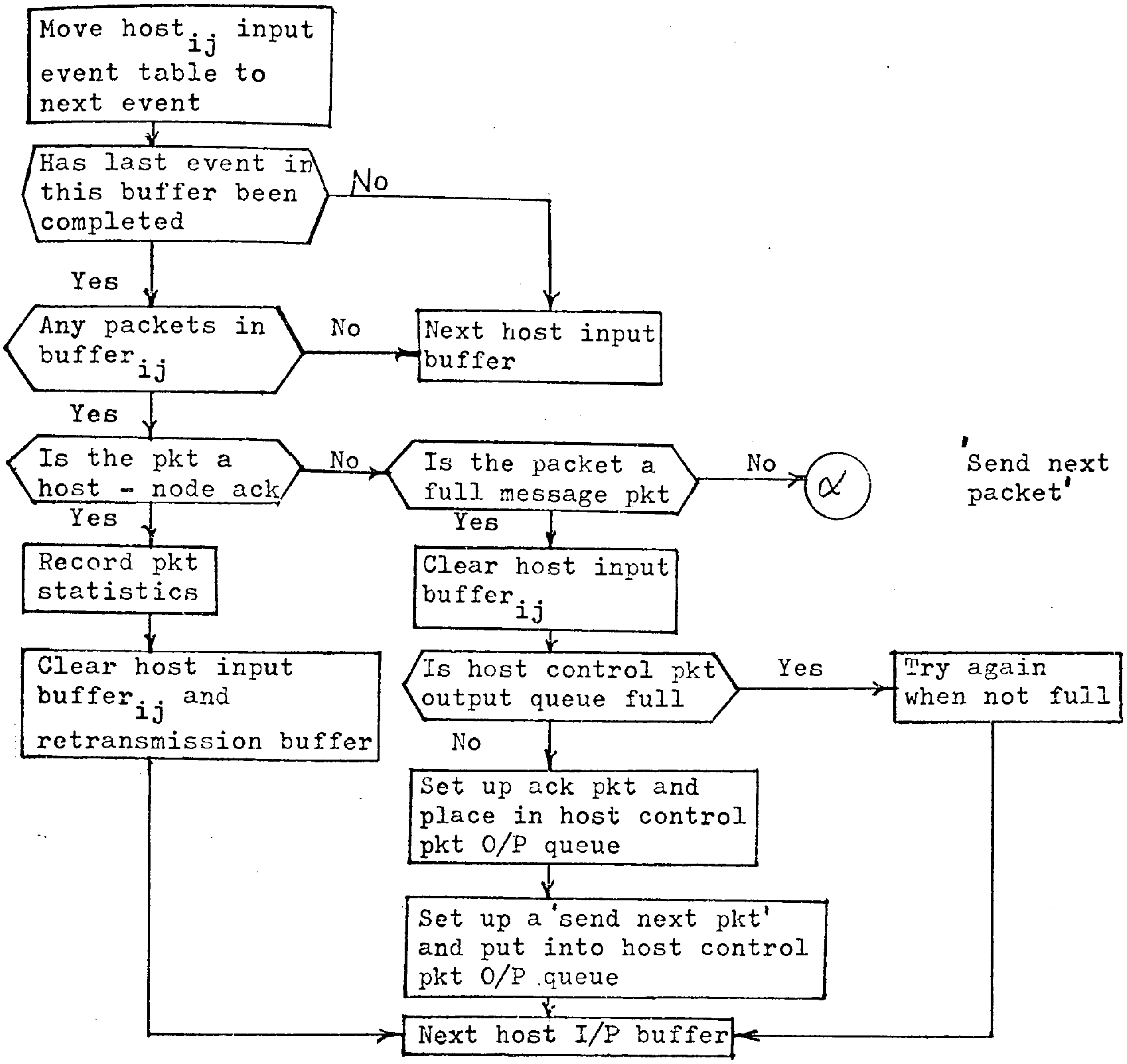

Move host$_{ij}$ input event table to next event
Has last event in this buffer been completed
No
Yes
Any packets in buffer$_{ij}$
No
Next host input buffer
Yes
Is the pkt a host - node ack
No
Is the packet a full message pkt
No
α
'Send next packet'
Yes
Record pkt statistics
Yes
Clear host input buffer$_{ij}$
Clear host input buffer$_{ij}$ and retransmission buffer
Is host control pkt output queue full
Yes
Try again when not full
No
Set up ack pkt and place in host control pkt O/P queue
Set up a 'send next pkt' and put into host control pkt O/P queue
Next host I/P buffer

Figure 4.11(i) Host input buffer service - node(i), host(j)

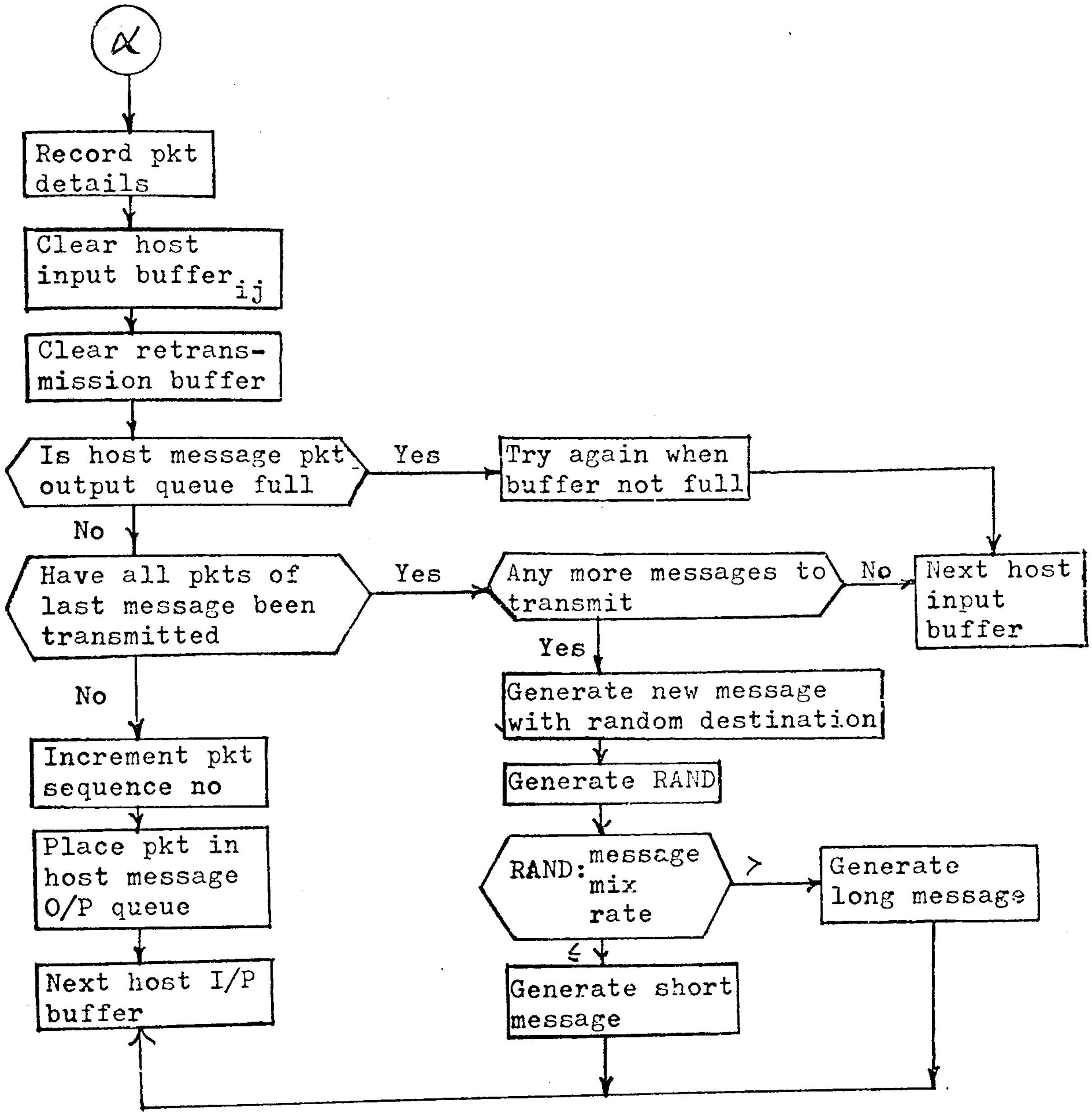

α
Record pkt details
Clear host input buffer$_{ij}$
Clear retrans-mission buffer
Is host message pkt output queue full
Yes
Try again when buffer not full
No
Have all pkts of last message been transmitted
Yes
Any more messages to transmit
No
Next host input buffer
Yes
Generate new message with random destination
No
Increment pkt sequence no
Generate RAND
Place pkt in host message O/P queue
RAND: message mix rate
>
Generate long message
≤
Next host I/P buffer
Generate short message

Figure 4.11(ii) Host input buffer service - node(i), host(j)

4.9 Simulation Program

The simulation program is written in Fortran. There was no alternative to Fortran since the original simulation was written for the departmental Prime 300 which only offered Fortran as a compilable high level language. The program was transferred to an ICL 1904S when the reliability of the Prime proved unsatisfactory. At the time of transfer the simulation program had been tested out and time was short to complete production runs. A simulation language was therefore not considered.

The complete version of the simulation program appears in Appendix II. The rest of the chapter will be concerned with a discussion of the simulation program.

4.10 Generating starting conditions

Some comments have to be made regarding the starting conditions of the simulation run. The values chosen must reflect the typical state of the network if it were inspected at random. If the network closed down at frequent intervals then the starting conditions would be easy to determine. With a network that ran for a long time the initial conditions would be difficult to estimate.

A technique that is commonly used is to invent starting conditions and run the simulation for some time. The final state of the system is then taken as the starting conditions of the genuine run. This in turn raises the question of how long the preliminary run should be. Generally, the longest cycle in the network should be executed 3 or 4 times to enable abnormal behavior of the network induced by non-sensible starting conditions to die away.

The system could also be started from an empty network by introducing a very high traffic intensity (>1) at the start in order to allow queues to build up very rapidly at the start. Reducing the traffic intensity and allowing a few cycles to be executed should bring the network to a normal state.

A decision also has to be made regarding whether successive runs are to be independent (as described above) or that the final calculations of one run be used as the starting conditions of the next run.

In practice, the labour of making valid fresh starts on each run weighs heavily in favour of continued runs. It is very difficult to predict any instabilities that may arise from continued running of the same network and comparisons between runs with different parameters become difficult to compare.

From several trial runs it was found that the network reached stable conditions very quickly (<0.05 seconds). In view of this fact it was decided to start each run afresh with an empty network. Using the same chaincode generators for each run will also help to make comparisons between different network parameters easier to make.

4.11 Generation of Pseudo-Random Numbers

During the initial development of the simulation program the Prime 300 was a new machine and the available software did not include a random number generator. The congruence method was considered because of its simplicity but the cycle length of the generator was limited. The congruence method is based on an equation of the form shown in equation 4.2

$$r_{i+1} = (\alpha r_i + \beta) \pmod{M} \qquad \ldots\ldots\ldots\ldots\ldots(4.2)$$

where α and β are defined constants and M is determined by the number of bits in the computer word. On the Prime the MOD function was limited to 512 due to a software fault.

Chaincodes, which are fully described in Appendix 1, are machine independent and permit very long sequences before repetition. Chaincodes were therefore chosen although the cost of computation was much higher.

The generation of exponentially distributed random numbers is based on an exponential cumulative density function of the form:

$$E(t) = 1 - e^{t/m}$$

where m is the mean of the distribution and t is time.

A vector is filled using this function with integers beginning with '1' in such a way that the number of integers in each group decreases exponentially as the value of the integer increases. The distribution of integers so generated is shown in Figure 4.12.

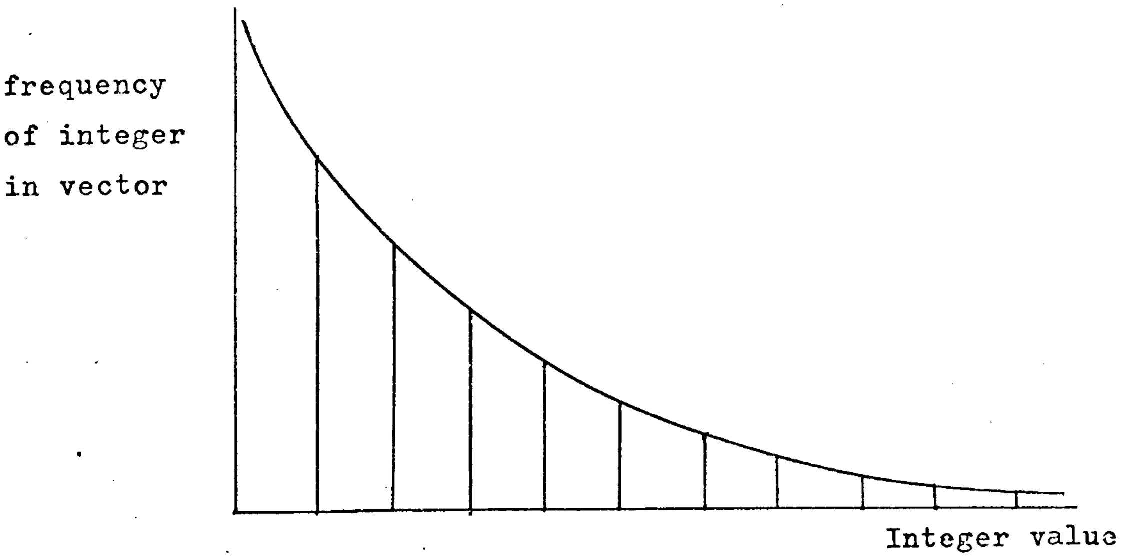


Figure 4.12 Distribution of integers in sampling vector

Exponentially distributed random numbers may now be obtained by sampling from the vector using a uniformly distributed random generator giving variates in the range '1' to 'size of the vector'. The larger the size of the vector, the more accurately the distribution will be represented. Exponentially distributed variates are required for three purposes:

1) Message Interarrival times (real value)
2) No of generating hosts (integer value)
3) Length of a message (integer value)

To obtain a reasonable level of accuracy would require a moderately large vector of around 32k. For the above this would cause an unjustifiable overhead in that 128k words of memory would be required to store three vectors (2 integer vectors and one real.)

However, use of the fact that variates generated from an exponential distribution of mean 1 may be scaled by the desired exponential mean and the variate will show the characteristics of having been generated from the required exponential distribution.

The vector was filled by evaluating for each t the expression

$$\text{INT}\left\{(1-e^{-t_n/m}) - (1-e^{-t_{n-1}/m})\right\}\text{. array size.}$$

t was chosen to be 0.01 to give a reasonable spread of integers. Scaling t by 100 gave the integer stored in the vector. When the vector was sampled the integer obtained was first scaled down by 100. The vector size was taken to be 32k and cut off at 32383 since the above expression was yielding very small values which resulted in zeros after the INT operation. Using the 32383 cut off point ensured that when the vector was sampled a non-zero value was always yielded. Since 99% of the distribution is represented this seemed reasonable. For each value of t, the expression yielded the number of vector elements to be labelled with that value of t * 100.

A pseudo-random generator employing four chaincode generators was used to generate uniformly distributed numbers in the range 1 - 2^{15}, values greater than 32383 resulted in a new number being generated. Separate sets of chaincodes were kept for each of three exponential distributions required to help make comparisons between runs more easy.

On transferring the program to the ICL 1904S, 32k words of memory being devoted to random number generation proved a heavy overhead resulting in runs being given a very low execution priority. This was overcome by compressing the 32k vector into a cumulative frequency table of 600 elements (limiting the vector size to 32383 produced 600 variates distributed between 0.01 and 6.0 with a mean of 1).

Figure 4.13 shows part of the distribution of E(t) and Figure 4.14 shows part of the cumulative distribution of E(t).

t	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
freq	326	322	319	316	313	310	307	303	300	297

Figure 4.13 Part of E(t) distribution

t	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
cum freq	326	648	967	1283	1596	1906	2213	2516	2816	3113

Figure 4.14 Part of Cum freq Dist of E(t)

Figure 4.15 shows the distribution of integers in the cumulative distribution frequency table.

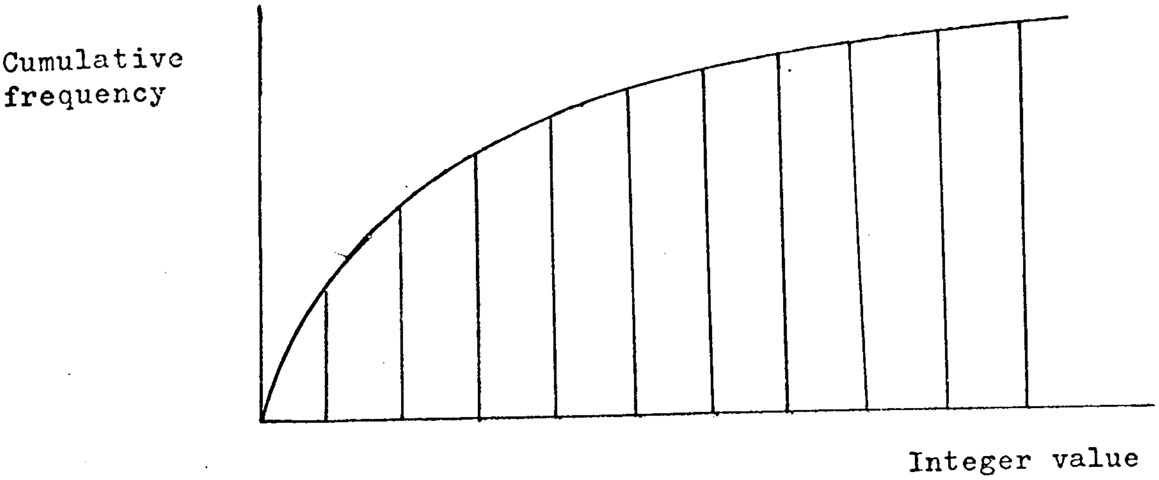


Figure 4.15 Distribution of integers in Cumulative Frequency Table

When a uniformly distributed random number is generated, say 1600, a binary search is carried out on the 600 locations holding the cumulative frequencies. Using Figure 4.14, 1600 lies between t = 0.05 and t = 0.06 and hence the required exponential variate is 0.05. Although a binary search requires $\log_2 600$ comparisons as compared to the direct extraction, it will not cause as great an overhead as running the job with an extra 32k of memory. Tocher [46] describes this process in more detail and calls it the 'Top Hat'. method.

4.12 Simulation Program Printouts

Output from the simulation program takes two forms. Firstly, there is a periodic output the frequency of which is specified by the user, then at the completion of the run cumulative statistics are given. The outputs begin by giving full information concerning the network topology which has been input by the user together with details concerning message arrivals.

Periodic output takes the following form:

1) Absolute simulation time
2) No of packets in the network
3) No of packets handled by the network since the last printout
4) Total number of packets handled to current printout
5) Each host's queue length of messages awaiting input to network
6) Total number of messages awaiting input to network
7) % of network processor time used since last printout
8) % of network memory time used since last printout

During these periodic printouts any absence of memory when required was also noted. At the completion of the simulation run the following statistics are given:

1) Number of each type of packet handled
2) Frequency of each message length handled
3) Average time to handle each message length
4) Mean message length
5) Total % processor time used by each processor.

CHAPTER FIVE

EFFECTS OF PARAMETER CHANGES IN THE NODE

5.1 Introduction

This chapter begins with methods of smoothing raw data, after which the Autocorrelation method is described and then used to determine the sampling frequency of data obtained from the simulation.

A standard network is then presented to give a base against which parameter changes may be compared. The parameter changes in this section are concerned with the node itself and include reducing the number of processors in the node, using slower processors/memory and reducing each memory module to contain only one packet.

5.2 Data Smoothing [47]

Some method is required to enable the desired features to be obtained from the sampled data system i.e. the removal of wild fluctuations. One simple way to do this is to use some form of cumulative average as given in (5.1)

$$\hat{a}_T = \frac{1}{N} \sum_{i=0}^{T} x_i \qquad \text{.........(5.1)}$$

where N is the number of samples up to time T. As each new sample is considered it is added to the running sum of all previous samples and divided by the number of samples considered up to and including the present one.

It can be seen that as T and therefore N become large $\hat{a} \rightarrow \bar{a}$ (the mean). In other words all peaks and troughs are smoothed out and transient conditions are not revealed.

Moving averages enable the drawbacks of cumulative averaging to be overcome. Only the previous n samples are taken to estimate $\hat{a}$, other samples than the nth being discarded. Thus dividing the sum of the previous n samples by n will give the new estimate as shown in (5.2).

$$\hat{a}_T = \frac{1}{n} \sum_{i=o}^{n-1} x_{T-i} \qquad \text{.........(5.2)}$$

The choice of n will determine the degree of smoothing i.e. for n=1 the raw data is repeated and as n increases so it will tend to the estimate given by cumulative averaging. The computational price has also increased. To the running total, the new sample is added and the nth previous sample must be subtracted.

Cox [48] suggested a technique which retains the flexibility of the moving averages and the computational simplicity of the cumulative average. A more elaborate weighting scheme is introduced which decreases the contribution of the sample with respect to time. This type of smoothing is called Exponentially Weighted Moving Average (EWMA) and can be implemented by an equation of the form given in (5.3)

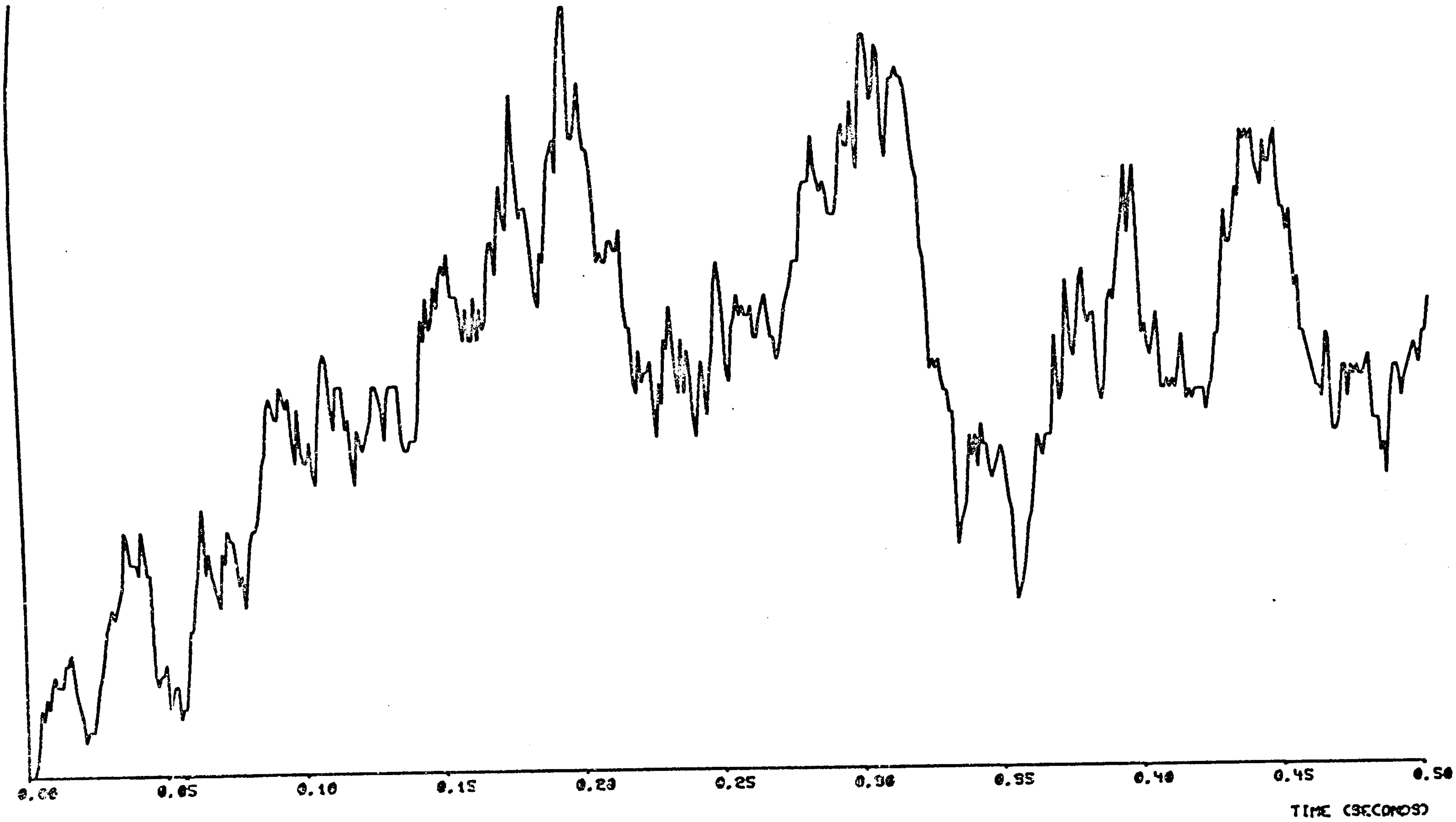


Figure 5.1 Raw message queue for input into network.

$$\hat{a}_T = \alpha . x_T + (1-\alpha) . \hat{a}_{T-1} \qquad \ldots\ldots\ldots(5.3)$$

Where α is the smoothing constant $0 \leqslant \alpha \leqslant 1$. The estimate may be recursively formed from a weighted version of the present sample and the previous estimate. Expanding equation (5.3) gives

$$\begin{aligned}\hat{a}_T &= \alpha . x_T + (1-\alpha)\left[\alpha . x_{T-1} + (1-\alpha) . \hat{a}_{T-2}\right] \\ &= \alpha . x_T + \alpha . (1-\alpha) . x_{T-1} + (1-\alpha)^2 . \left[\alpha . x_{T-2} + (1-\alpha) . x_{T-3}\right] \\ &= \alpha . x_T + \alpha . (1-\alpha) . x_{T-1} + \alpha . (1-\alpha)^2 . x_{T-2} + \\ &\qquad \ldots . + \alpha . (1-\alpha)^K . x_{T-K} + \ldots . (1-\alpha)^T . x_0\end{aligned}$$

The weight given to a previous sample decreases with age as a geometric series in general given by (5.4)

$$\hat{a}_T = \alpha . \sum_{K=0}^{T-1} (1-\alpha)^K . x_{T-K} + (1-\alpha)^T . \hat{a}_0 \qquad \ldots(5.4)$$

It can be seen that the degree of smoothing is entirely dependent upon α. Figure 5.1 shows messages arriving with a traffic intensity of 0.8, and queueing for entry to the network. Figure 5.2 shows the effects of smoothing the raw data with α = 0.01, 0.05, 0.1, 0.2.

With α = 0.01 troughs and peaks have disappeared as would have happened with cumulative averaging.

α was chosen as 0.1 since it masked fluctuations yet retained most peaks and troughs of importance.

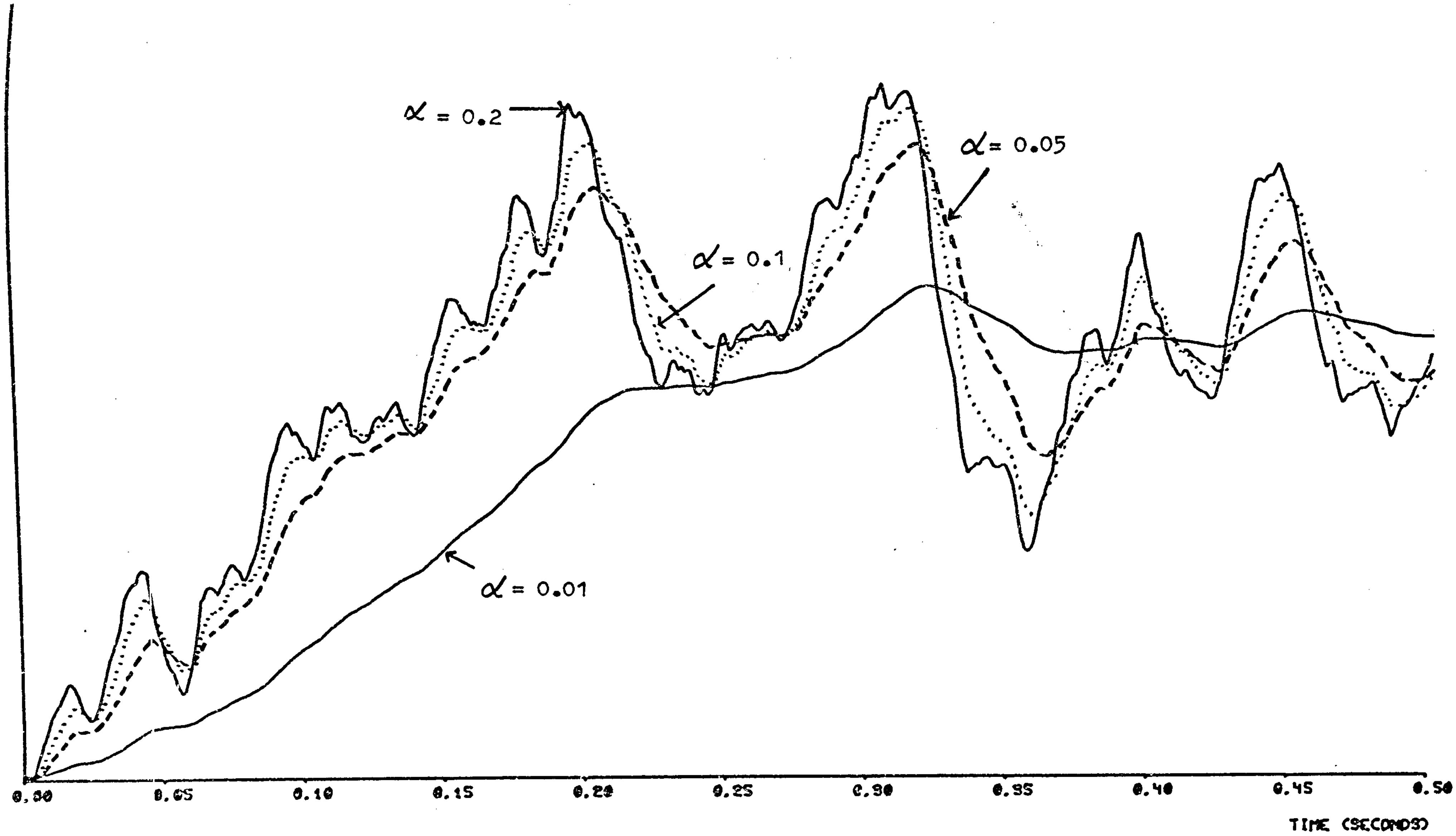


Figure 5.2 Exponentially Weighted Moving Average of message queue for input into network, with α = 0.01, 0.05, 0.1, 0.2.

5.3 Selection of Sampling Frequency

In any simulation, an important decision has to be made regarding the rate at which the system will be sampled. Blackman and Tukey [49] developed a powerful method of analysing processes for their frequency component distribution or power spectrum. The method will now be discussed but will be limited to equi-spaced samples which form a stationary time series i.e. fluctuations about a constant mean.

The correlation coefficient r represents the goodness of fit of an equation of the form y=bx+a to the sets of variables x and y. The correlation coefficient can be derived from normal equations [50] and is of the form given in (5.5).

$$r_{xy} = \frac{N\sum x_i \cdot y_i - (\sum x_i)(\sum y_i)}{\sqrt{\left\{N\sum(x_i)^2 - (\sum x_i)^2\right\}.\left\{N\sum(y_i)^2 - (\sum y_i)^2\right\}}} \qquad i=1,\ldots\ldots, N$$

$$\ldots\ldots\ldots\ldots\ldots\ldots(5.5)$$

This result may be applied to a set of time series data to calculate the autocorrelation coefficient between x_i and x_{i+p}, where p is a constant interval of time or lag of time. Equation (5.5) can be modified to give the autocorrelation coefficient for a particular lag p as given by equation (5.6).

$$r_p = \frac{(N-p)\sum x_i \circ x_{i+p} - (\sum x_i)(\sum x_{i+p})}{\sqrt{\left\{(N-p)\sum(x_i)^2 - (\sum x_i)^2\right\}.\left\{(N-p)\sum x^2_{i+p} - (\sum x_{i+p})^2\right\}}}$$

$$\ldots\ldots\ldots\ldots\ldots\ldots(5.6)$$

If the data is first normalised by

$$x_n = \frac{x_n - \bar{x}}{s} \qquad \ldots\ldots\ldots\ldots\ldots\ldots(5.7)$$

where $\bar{x}$ is the mean, and s is the standard deviation, then r_p becomes

$$r_p = \frac{\sum x_i . x_{i+p}}{\sum x^2_i} \qquad i = 1, \ldots, N-p \qquad \ldots\ldots\ldots\ldots\ldots\ldots (5.8)$$

as a function of p.

r_p will show those lags over which the data seems to be correlated.

If a time series is considered with a zero mean, autoco-variance (or autocorrelation) function Ø (p) may be defined as

$$\text{Ø}(p) = \frac{1}{N-p} \sum_{i=1}^{N-p} x_i . x_{i+p} \qquad \ldots\ldots\ldots\ldots\ldots\ldots (5.9)$$

from which it follows that

$$r_p = \frac{\text{Ø}(p)}{s^2_x} \qquad \ldots\ldots\ldots\ldots\ldots\ldots (5.10)$$

For a continuous stationary time function x(t) the autocorrelation function is given by

$$\text{Ø}(p) = \lim_{T \to \infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t).x(t+p).dt \qquad \ldots\ldots\ldots (5.11)$$

The power spectral density function P(f) of the same process is defined as

$$P(f) = \lim_{T \to \infty} \left\{ \int_{-T/2}^{T/2} x(t).\, e^{-j2\pi ft} .dt \right\}^2 \qquad \ldots\ldots\ldots\ldots\ldots\ldots (5.12)$$

which represents the contribution to the variance of x(t) with frequencies between f and f+df.

Blackman and Tukey [49] showed that these two functions could be written as a fourier transform pair

$$\emptyset(p) = \int_{-\infty}^{\infty} P(f).\ \cos 2\pi fp.df \qquad \ldots\ldots\ldots\ldots(5.13)$$

$$P(f) = \int_{-\infty}^{\infty} \emptyset(p).\ \cos 2\pi fp.dp \qquad \ldots\ldots\ldots\ldots(5.14)$$

which could be simplified to

$$\emptyset(p) = 2\int_{0}^{\infty} P(f).\ \cos 2\pi fp.df \qquad \ldots\ldots\ldots\ldots(5.15)$$

$$P(f) = 2\int_{0}^{\infty} \emptyset(p).\ \cos 2\pi fp.dp \qquad \ldots\ldots\ldots\ldots(5.16)$$

When considering a discrete time series x(t) it is necessary to introduce a finite fourier series transformation instead of the infinite continuous summation.

Equation (5.9) shows the summation for the autocorrelation function at time lag p.

Given that the significant power contributions are below π rad/lag time, L(p), which gives the raw estimates of the power spectral density function can be obtained using (5.10) by

$$L(p) = \emptyset(0) + 2\sum_{q=1}^{M-1} \emptyset(q)\cos\frac{qp\pi}{M} + \emptyset(M)\ \cos p\pi$$

$$\ldots\ldots\ldots\ldots\ldots\ldots(5.17)$$

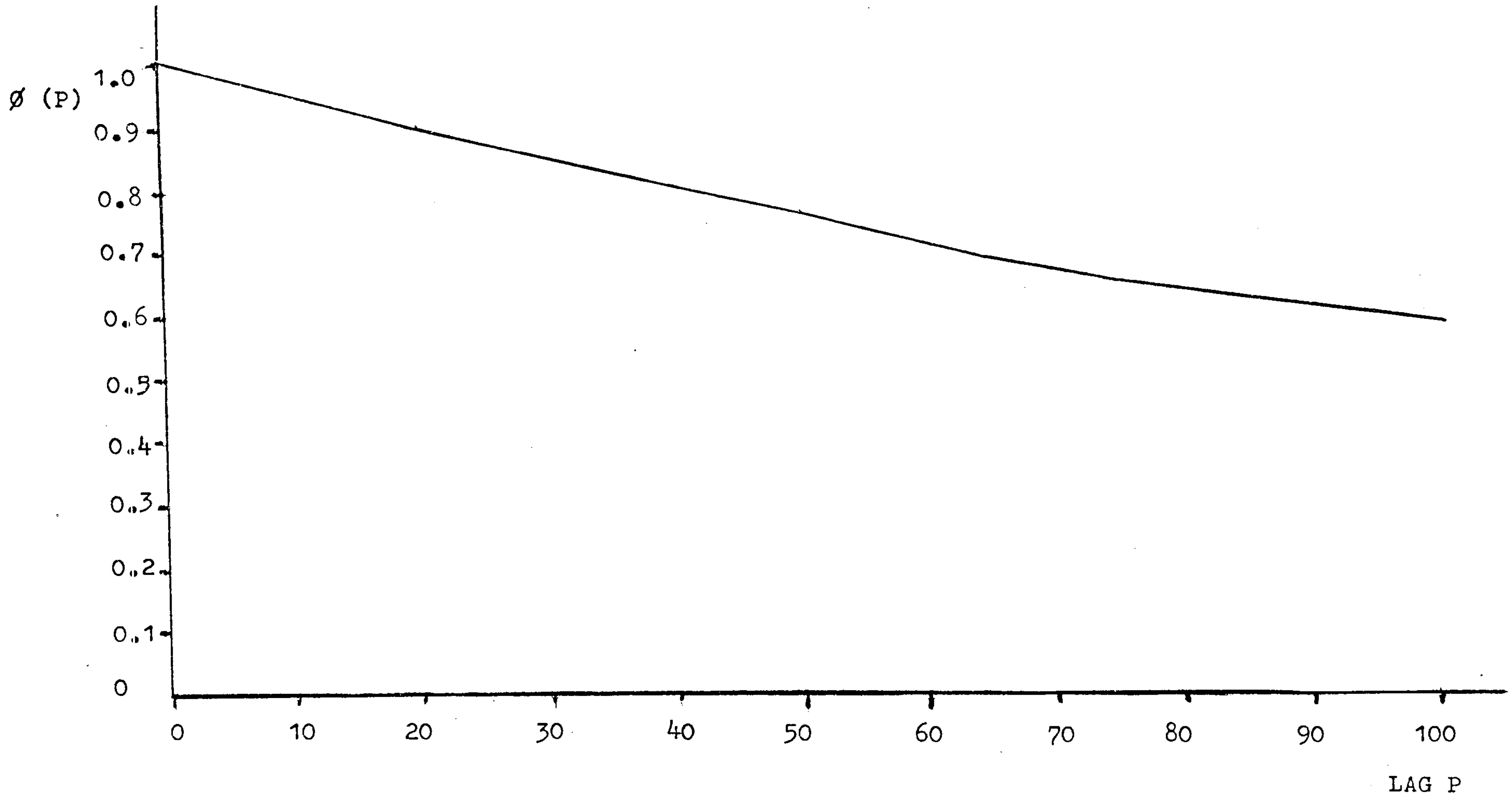


Figure 5.3 System Autocorrelation Function

where M is the maximum lag value of p,[51].

These raw estimates may be smoothed in various ways. Southworth[51] recommends

$$U(p) = 0.23\ L(p-1) + 0.54\ L(p) + 0.23\ L(p+1) \qquad \ldots\ldots(5.18)$$

where $L(-1) = L(1)$ and $L(M+1) = L(M)$.

$U(p)$ represents the corrected estimates of the smoothed power density which gives the power contribution in the frequency interval $\frac{\pi p}{M} - \frac{\pi}{2M}, \frac{\pi p}{M} + \frac{\pi}{2M}$.

5.4 Obtaining the Sampling Frequency

A simulation run was made with a three node network and three hosts per node. This configuration was chosen together with fast line speed and fast memory/processor since it would give the heaviest traffic load that the network would have to sustain. In order to get a power spectrum the system had to be in a steady state i.e. where the service time was less than the arrival rate. A traffic intensity of 80% was chosen with a mean interarrival time of 350μseconds. (Several runs were made for the particular network and the level of saturation determined). Southworth[51] recommends that the ratio of M(maximum lag) to N(no of samples) does not exceed 10%. In order to get M=100, 5000 samples of the message queue for service are taken at a rate of 100μ seconds interval. The ratio is therefore as low as 2%. The simulation was given 0.05 seconds simulation time to settle down.

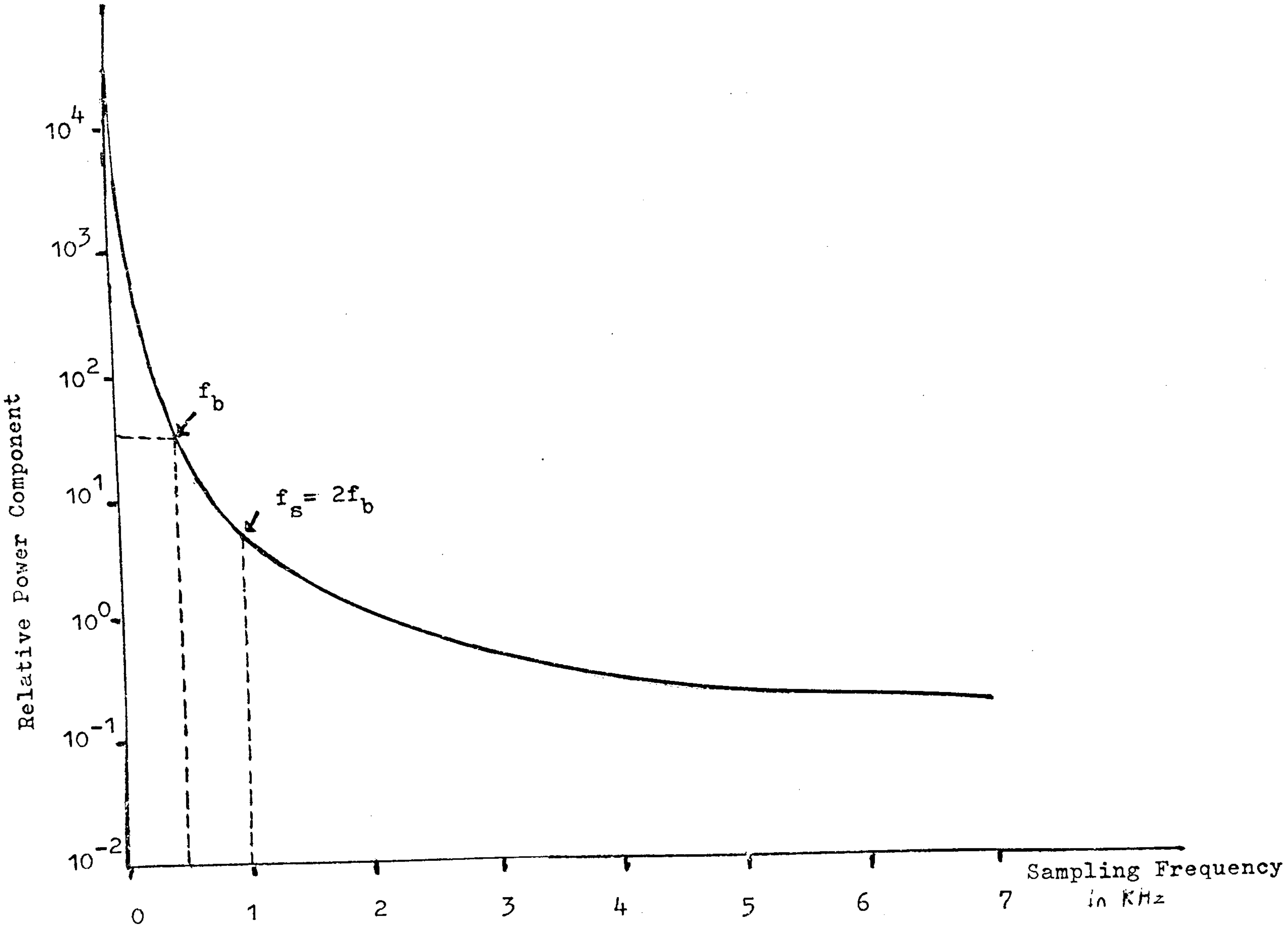


Figure 5.4 Sample Frequency Selection Using the Power Spectrum

These 5000 samples were than used to compute the system Autocorrelation function shown in Figure 5.3. The smoothed Power Spectrum is shown in Figure 5.4.

The significant bandwidth is contained within 0 → 500Hz. A frequency greater than two times this upper frequency is chosen. F_s, the sampling frequency, was chosen conveniently to be 1KHz, i.e. at a sampling interval of 1 msec.

5.5 Standard Network

In order to be able to evaluate parameter changes in the network, a standard network was chosen against which comparisons could be made. Figure 5.5 shows the basic network. The network consists of three fully connected nodes to each of which are

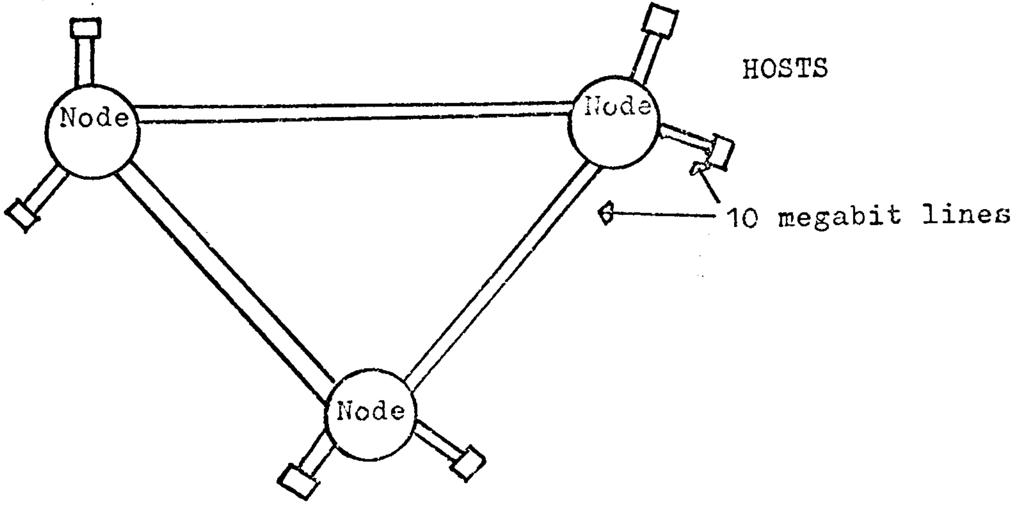


Figure 5.5 Standard Network

connected 2 hosts. All lines, nodes and local hosts are interconnected by 10 megabit lines.

Figure 5.6 is a block diagram of the structure of the node. Each node is architecturally identical containing 2 processors and 3 memory units. Memory cycle time and processor speed are matched at 100 nseconds. Packet sizes are set at 1024 bits. Each memory module of 1024 words (8 bit word) is thus able to hold 8 packets each. The node is connected to the other nodes via 2 sets of I/O buffers (1 packet length) and similarly connected to the two local hosts. Control packets are 8 words long. The mean number of generating hosts at each message event time is one host. It is assumed that the mean length of short messages is 1 packet and of long messages 10 packets, there being a ratio of 3:1 between short and long messages. Using the equation derived in the last chapter for the mean of a hyper-exponential distribution:

$$\frac{6}{\mu 1} + \frac{1-6}{\mu 2} \qquad \text{.........(5.19)}$$

for $\frac{1}{\mu 1} = 1$ and $\frac{1}{\mu 2} = 10$

this gives a mean message length of

$$0.75 \times 1 + (1- 0.75) \times 10 = \underline{3.25 \text{ packets.}}$$

The simulations is run for 0.5 seconds simulated time and sampled every 1 millisecond as discussed in section 5.4.

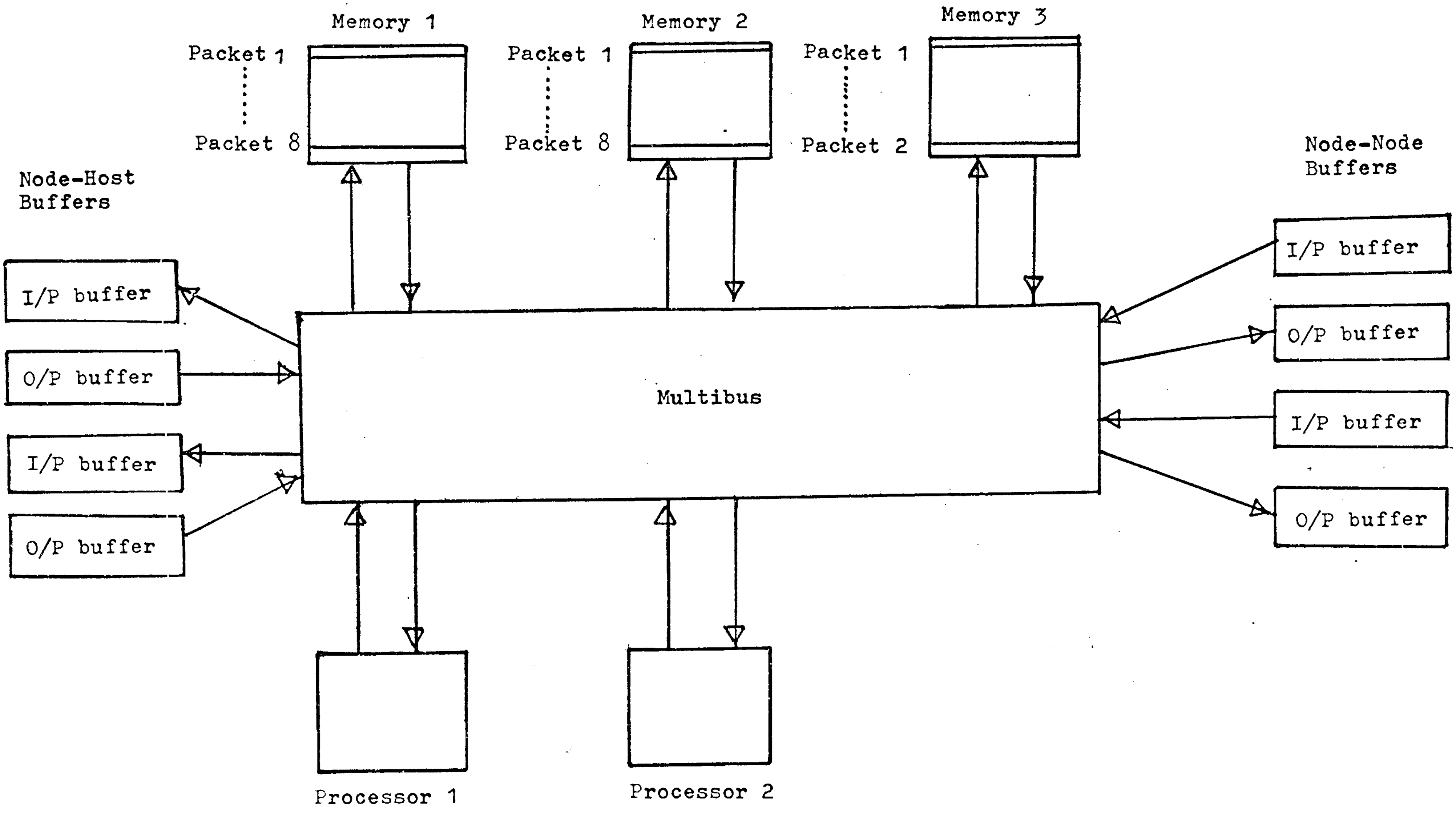


Figure 5.6 Standard Network: Node Architecture

The effects of all parameter changes are presented in the same way as for the standard network (SN) as shown in Figures 5.7 - 5.13. Figures 5.8 - 5.12 give the data at saturation level. When the average demand for service is less than the capacity of the system, the system is said to be in a steady state. The information which these graphs give is shown below.

1) Effect of varying message mean interarrival rate on message queue for input into network, (Figure 5.7).

2) Number of packets processed by network per millisecond sample interval, (Figure 5.8).

3) Percentage of network processor time used per millisecond sample interval, (Figure 5.9).

4) Percentage of network memory time used per millisecond sample interval, (Figure 5.10).

5) Distribution of message lengths sent through network, (Figure 5.11).

6) Effect of message length on message throughput time, (Figure 5.12).

7) Effect of varying message mean interarrival rate on network traffic, (Figure 5.13).

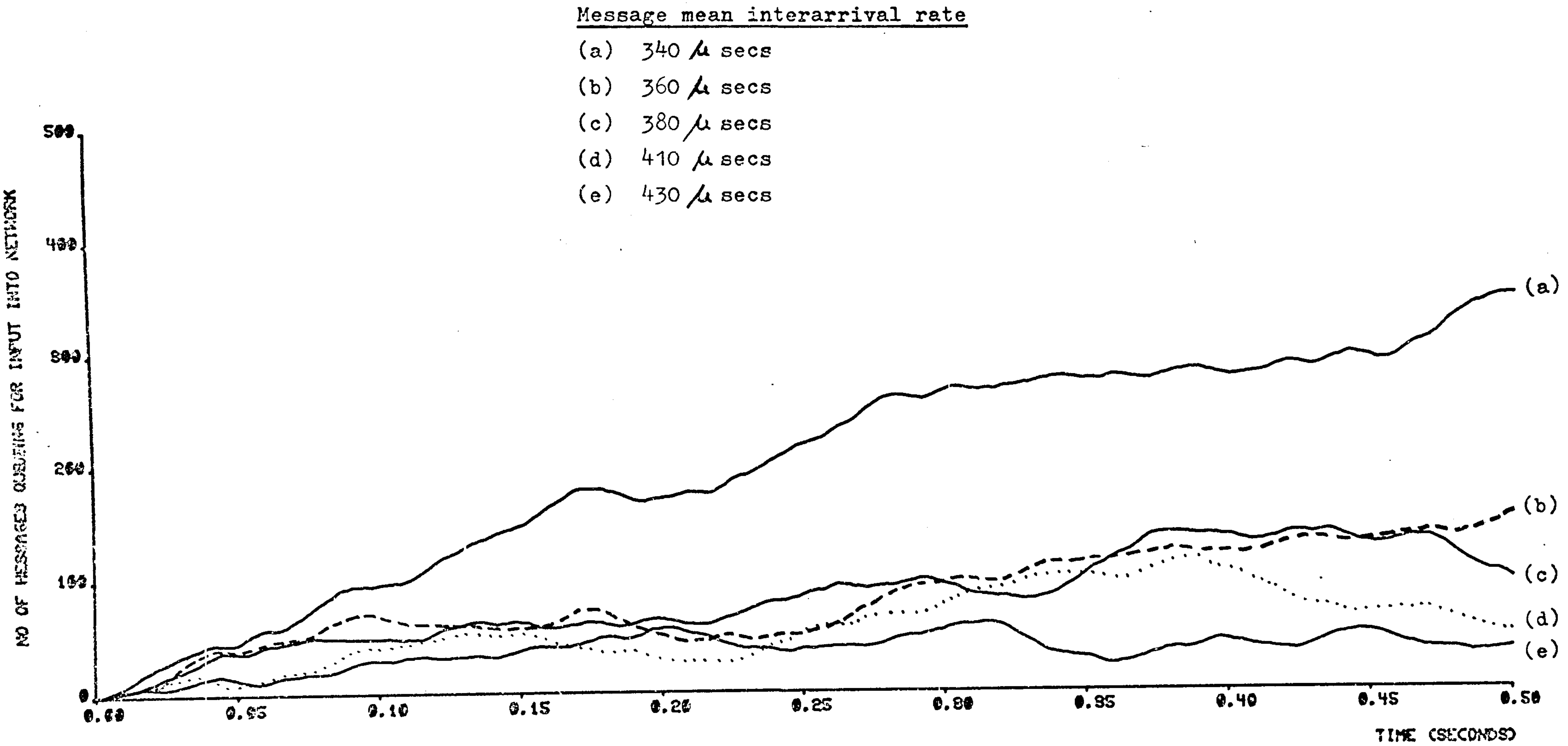


Figure 5.7 EWMA Standard Network.
Effect of varying message mean interarrival
rate on message queue for input into network.

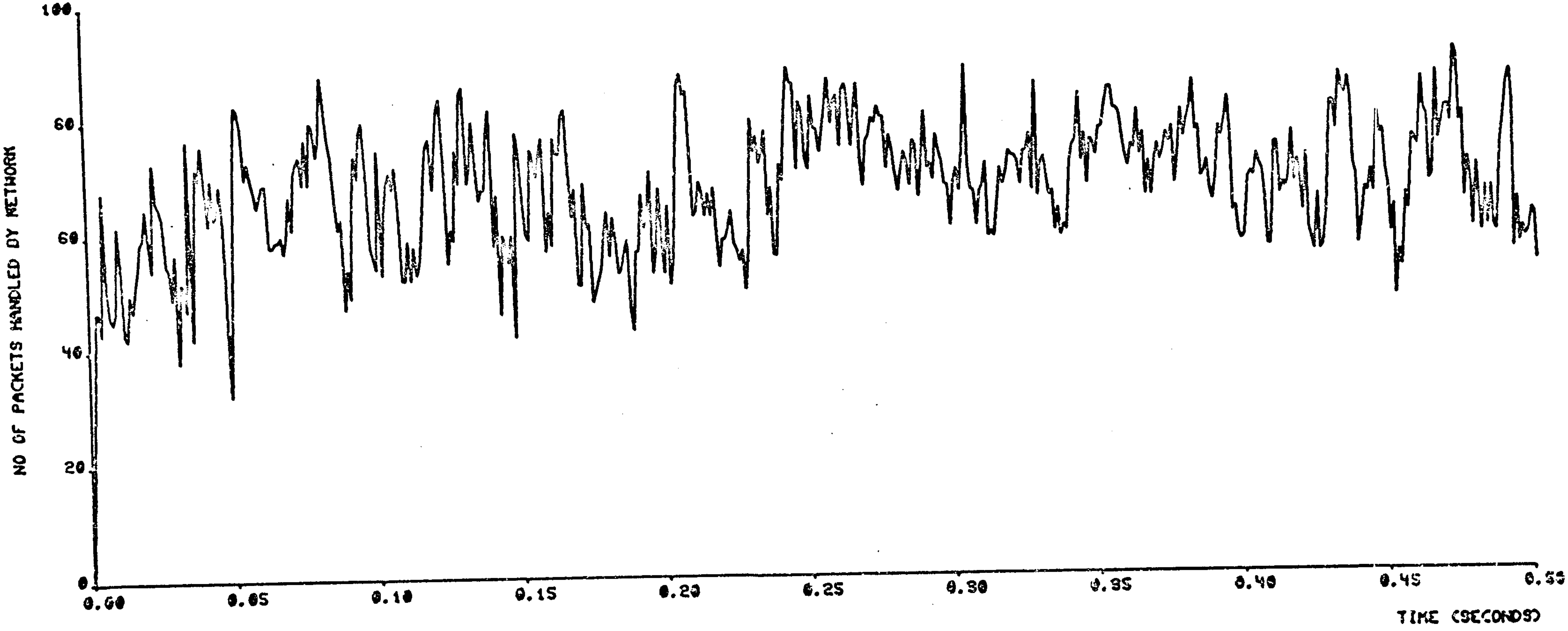


Figure 5.8 Standard Network.
Number of packets processed by network
per millisecond sample interval.

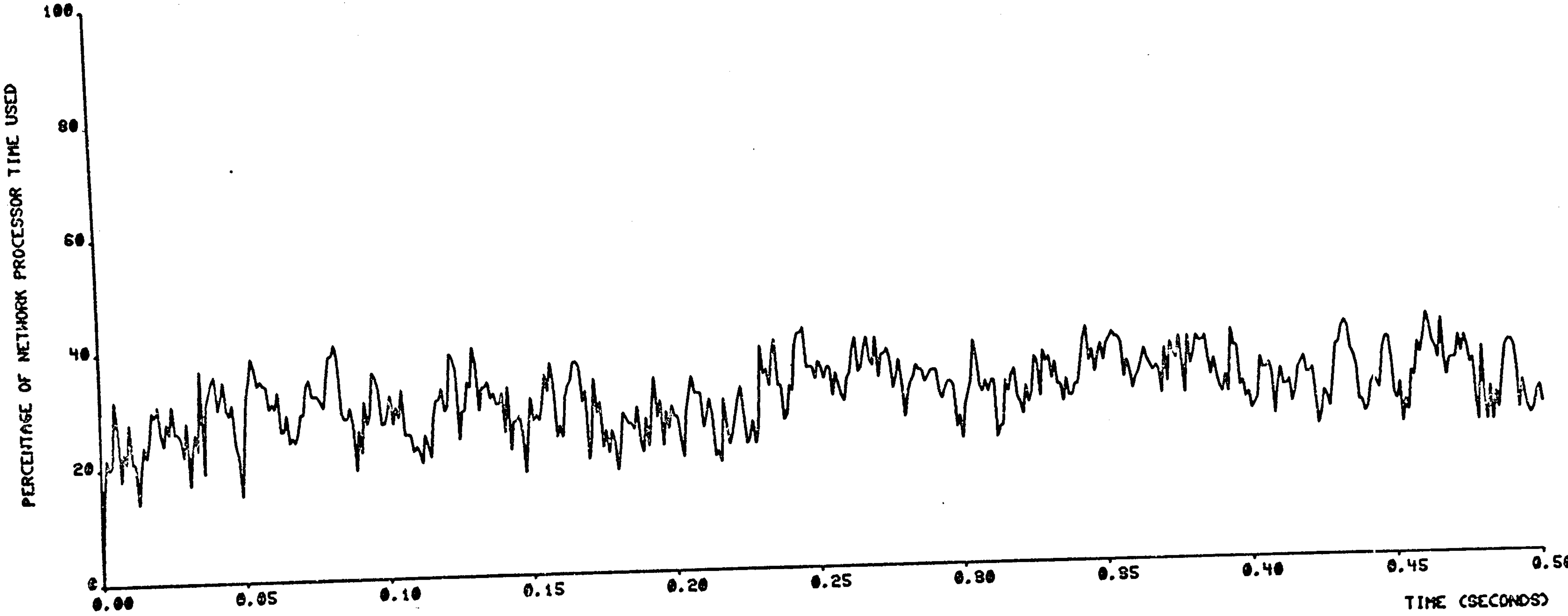


Figure 5.9 Standard Network.
Percentage of network processor time used
per millisecond sample interval.

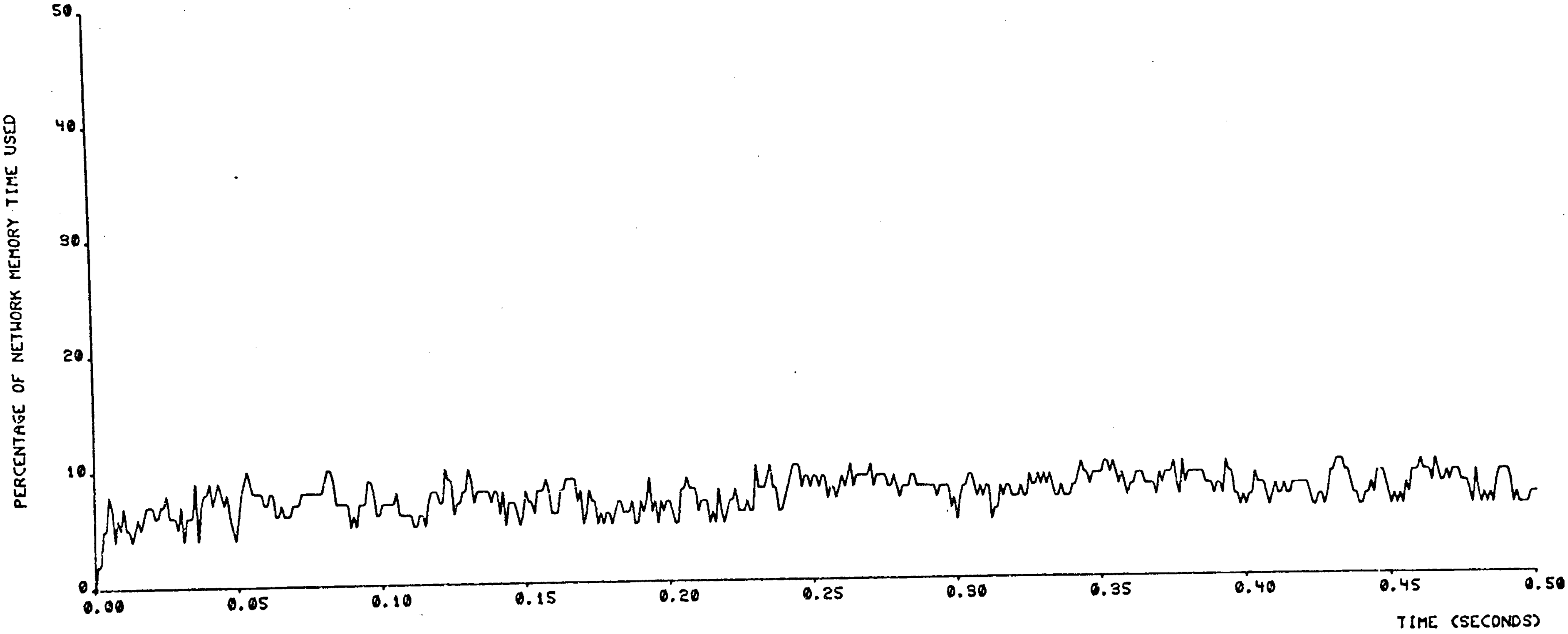
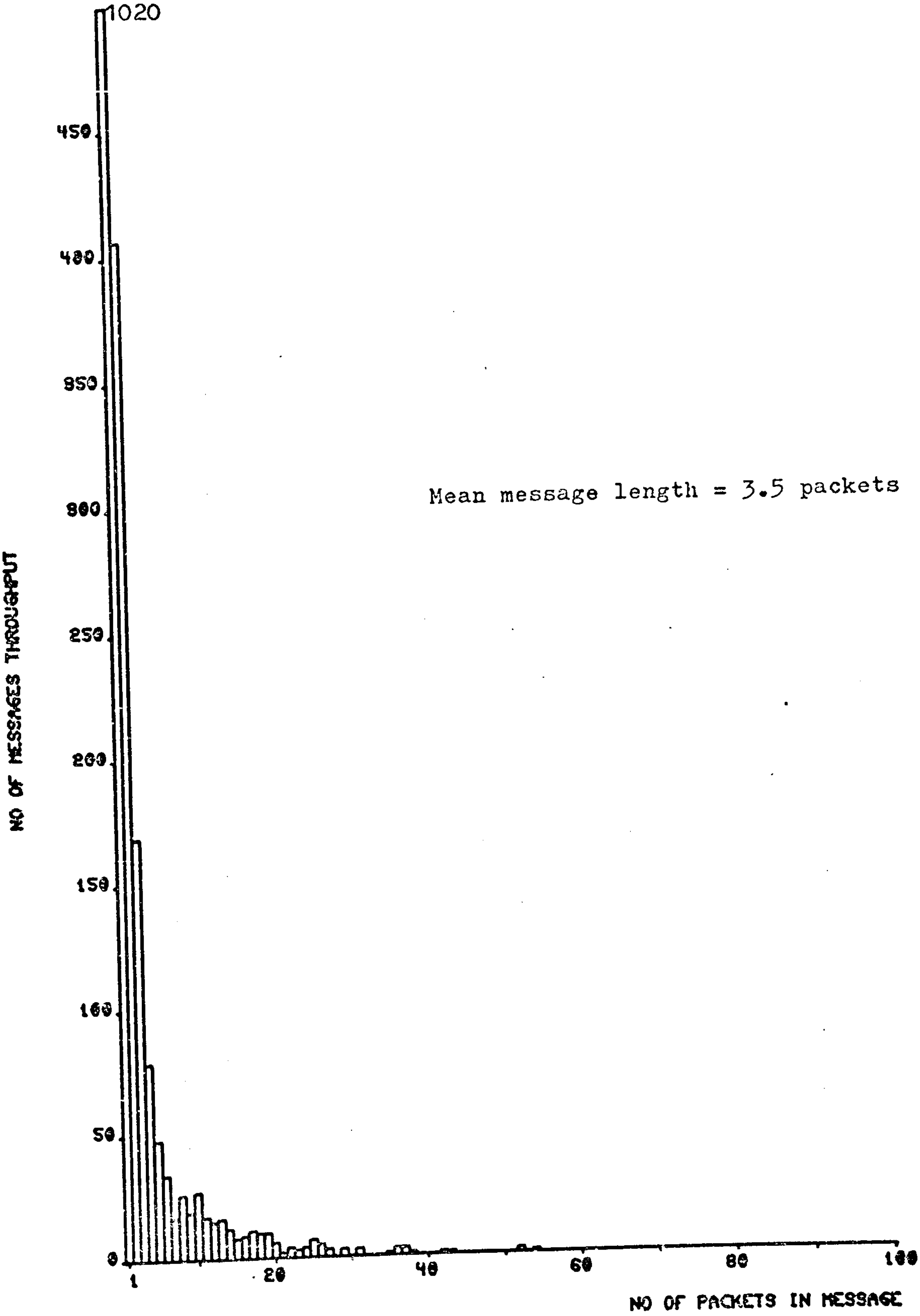


Figure 5.10 Standard Network.
Percentage of Network Memory time used
per millisecond sample interval.



Mean message length = 3.5 packets

Figure 5.11 Standard Network.
Distribution of message lengths
sent through network.

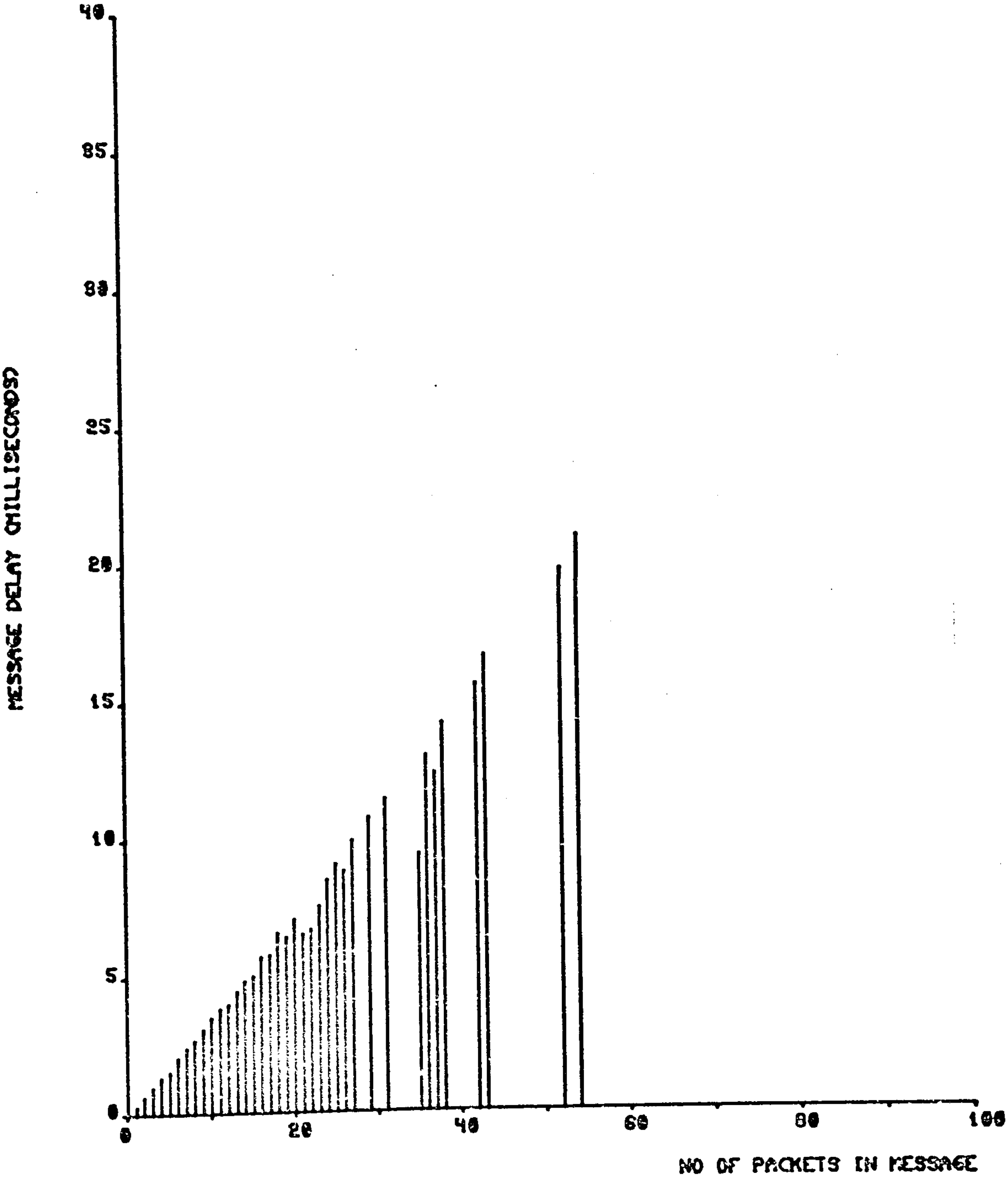


Figure 5.12 Standard Network.
Effect of message length on message throughput time.

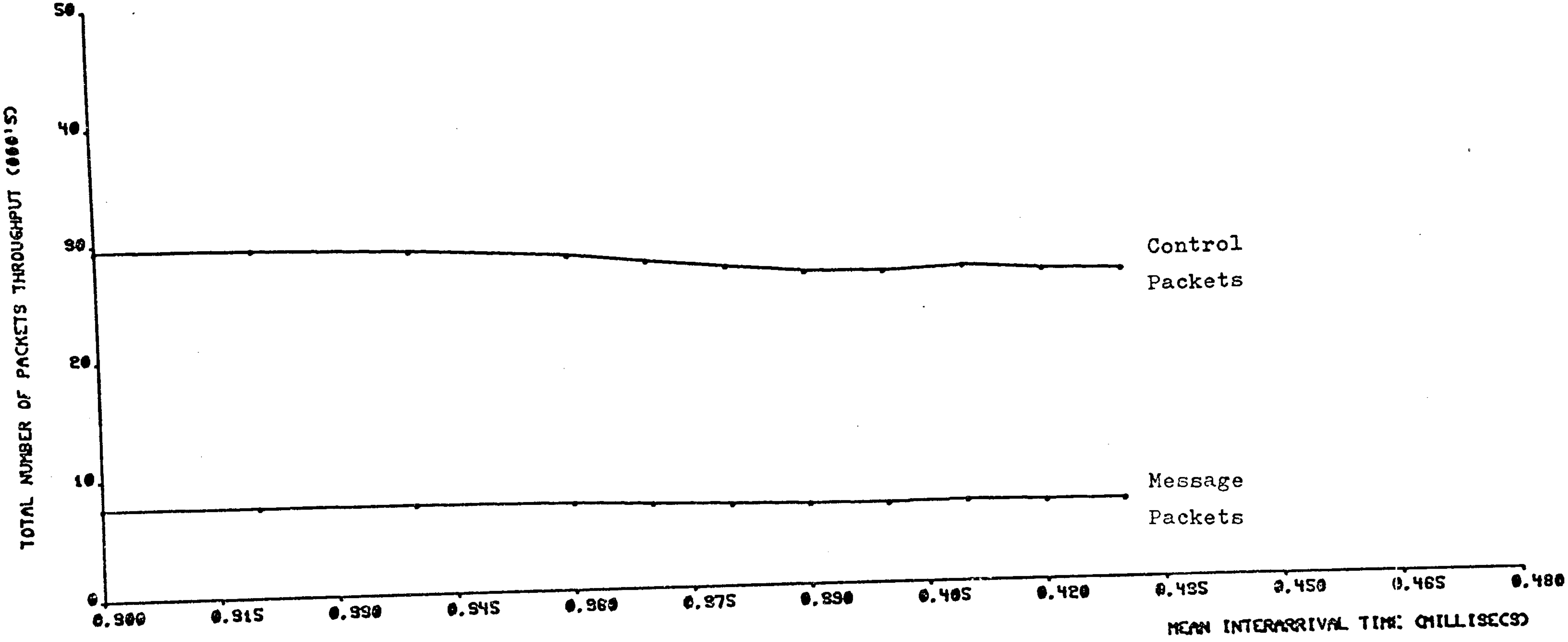


Figure 5.13 Standard Network.
Effect of varying message mean
interarrival rate on network traffic.

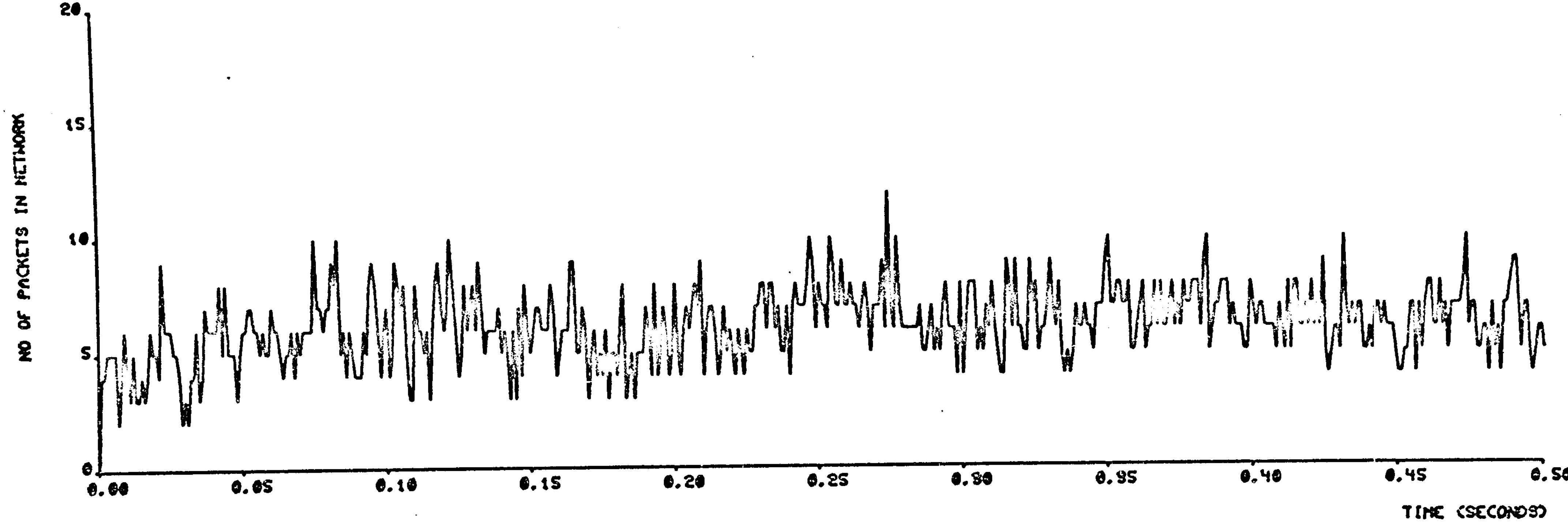


Figure 5.14 Standard Network - Number of packets in network.

For the standard network saturation occurred at a mean message interarrival time of 380μsecs shown in figure 5.7. This resulted in 7 X 10^3 message packets being throughput which required 27 X 10^3 control packets being generated. This represents a throughput rate of approximately 14 megabits/sec. The throughput time for a single packet was 300 μsecs while for a 54 packet message the time was 21 msecs. It can be seen from figure 5.12 that the throughput times are linear. This is explained by the fact that the standard network is fully connected and so the throughput procedure for each packet is identical.

Figure 5.9 shows the total processor time used on the network to be approximately 31% while figure 5.10. shows memory to be used at 7% of the total available. So there is a lot of spare capacity. Although the figure of 31% processor time used is spread over all the processors in the system, the figures given below indicate that the workload is fairly evenly distributed amongst all three nodes.

	node 1	node 2	node 3
proc 1	31%	35%	30%
proc 2	29%	32%	28%

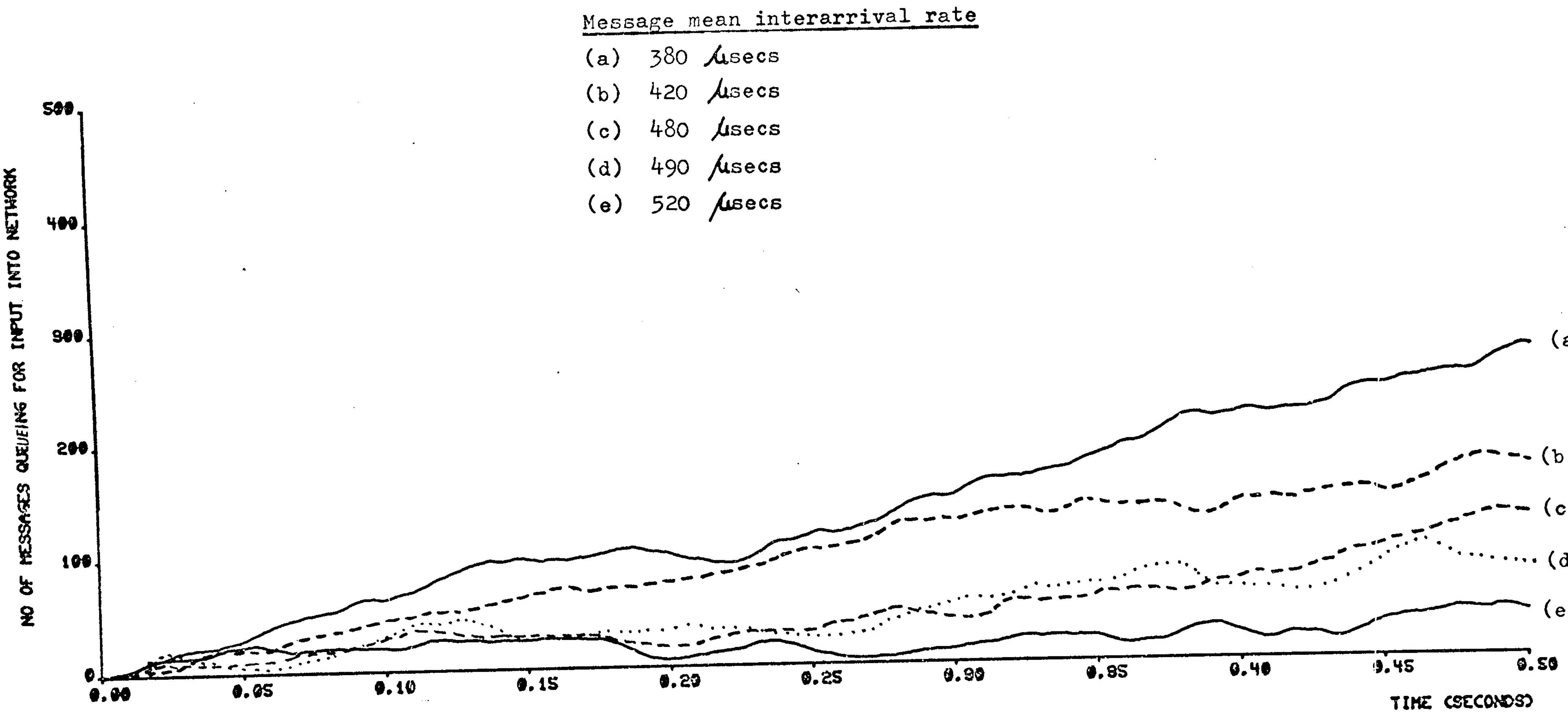


Figure 5.15 EWMA Standard Network - 1 processor, 2 memory units per node.
Effect of varying message mean interarrival rate on message queue for input into network.

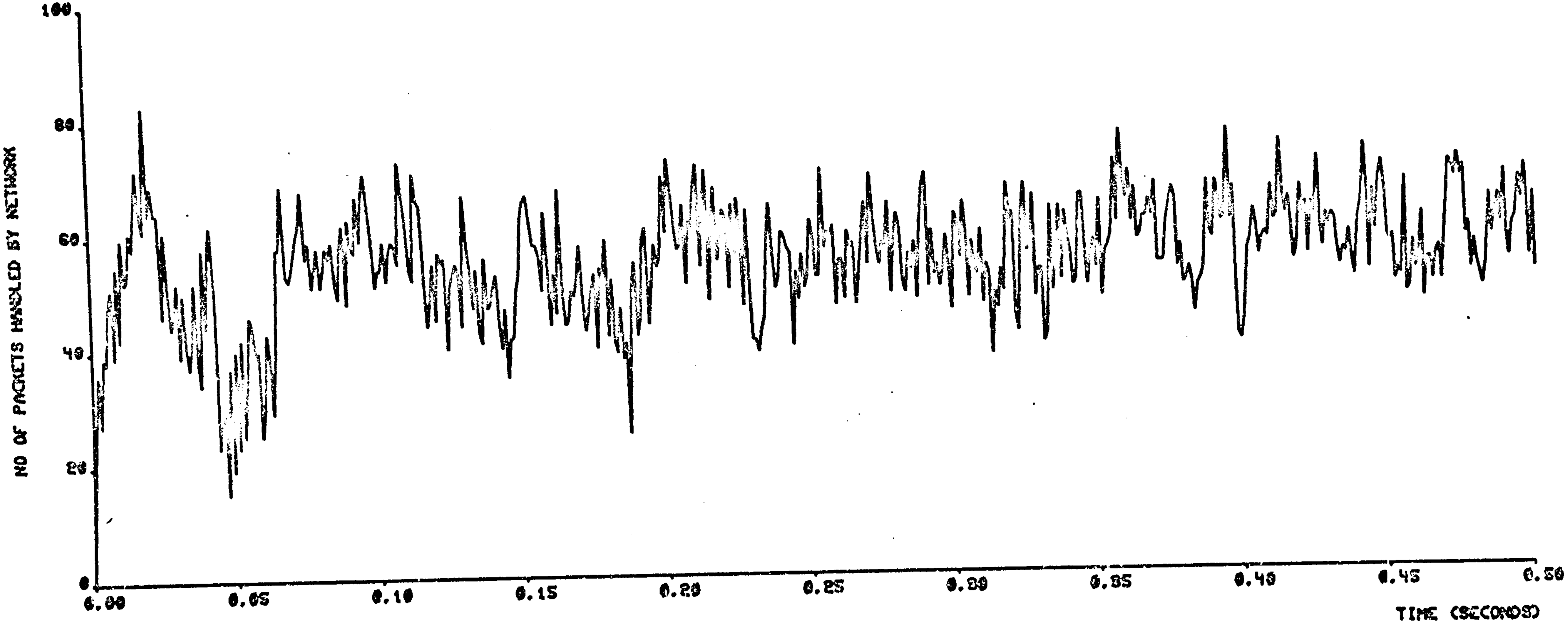


Figure 5.16 Standard Network - 1 processor, 2 memory units per node.
Number of packets processed by network per millisecond sample interval.

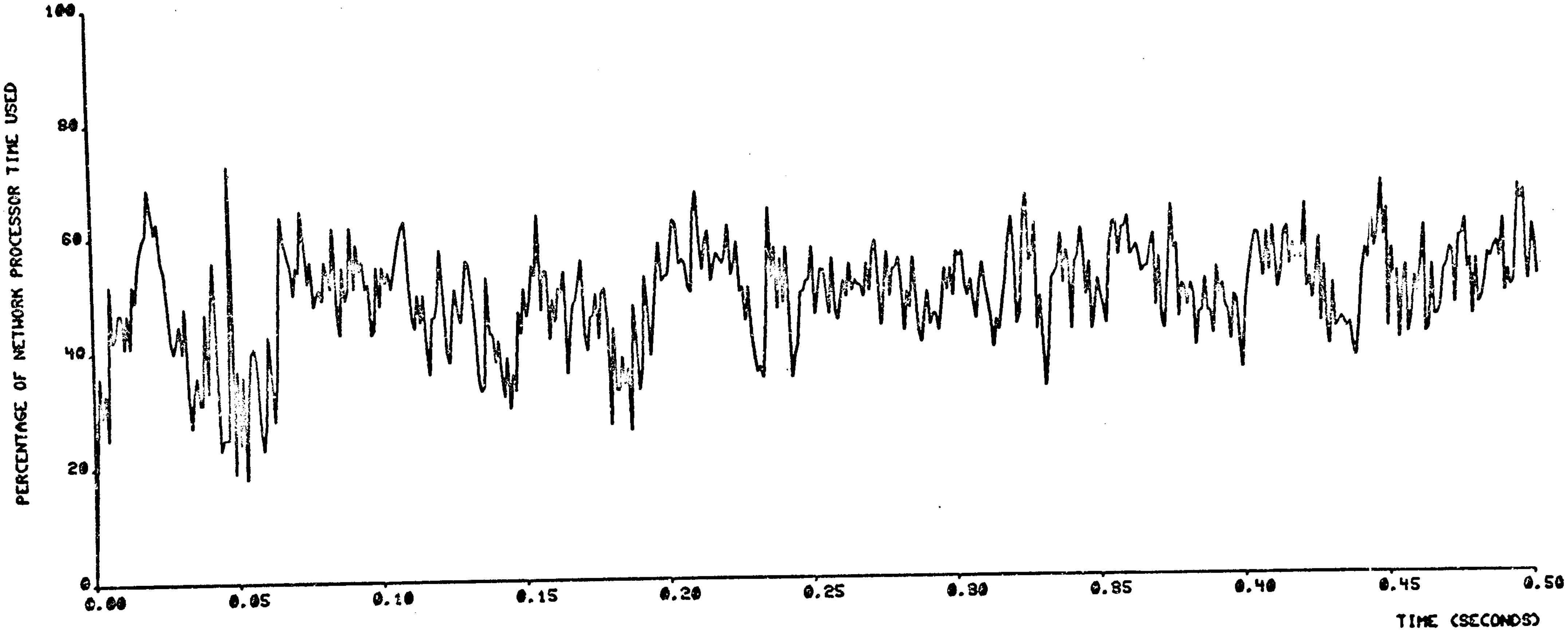
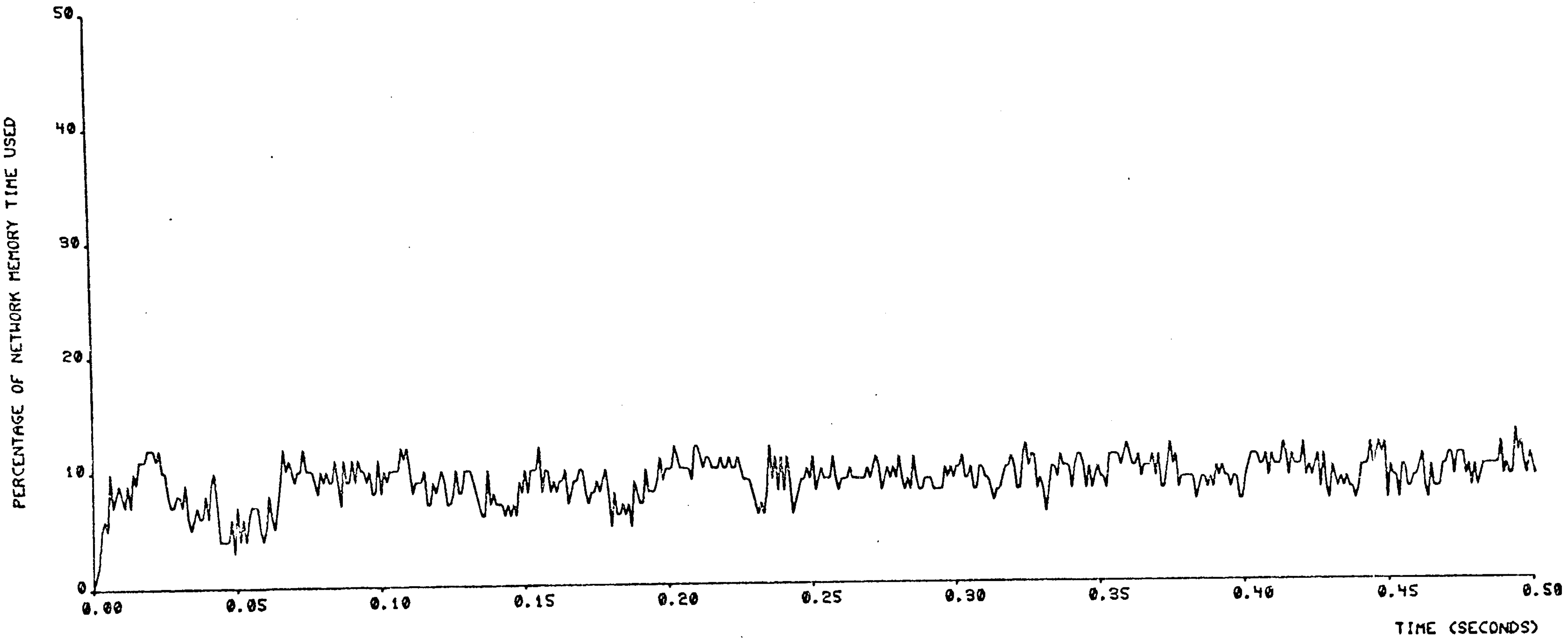


Figure 5.17 Standard Network - 1 processor, 2 memory units per node.

Percentage of network processor time used per millisecond sample interval.



<u>Figure 5.18</u> Standard Network - 1 processor, 2 memory units per node.
Percentage of network memory time used per millisecond sample interval.

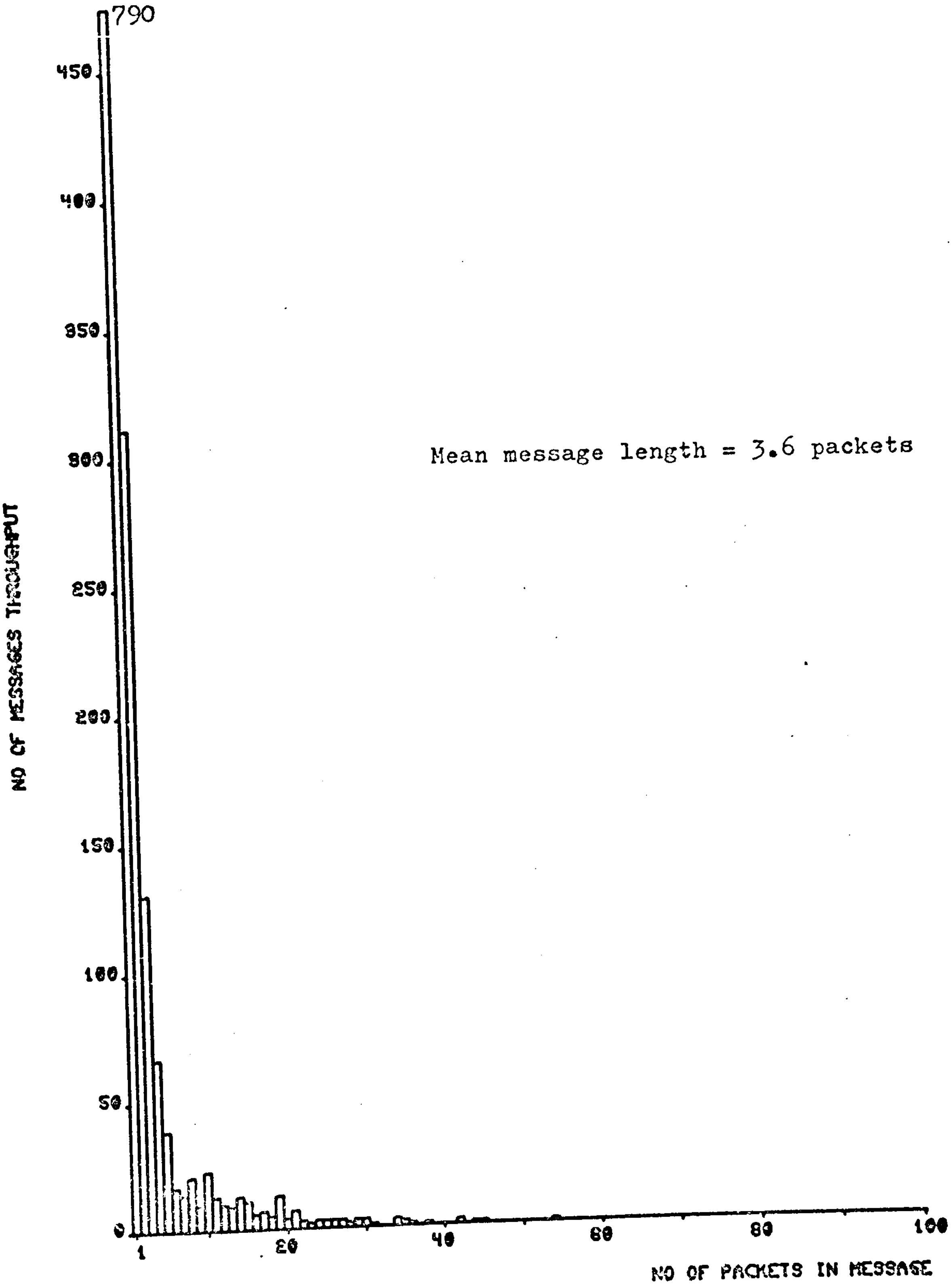


Figure 5.19 Standard Network - 1 processor, 2 memory units per node.

Distribution of message lengths sent through network.

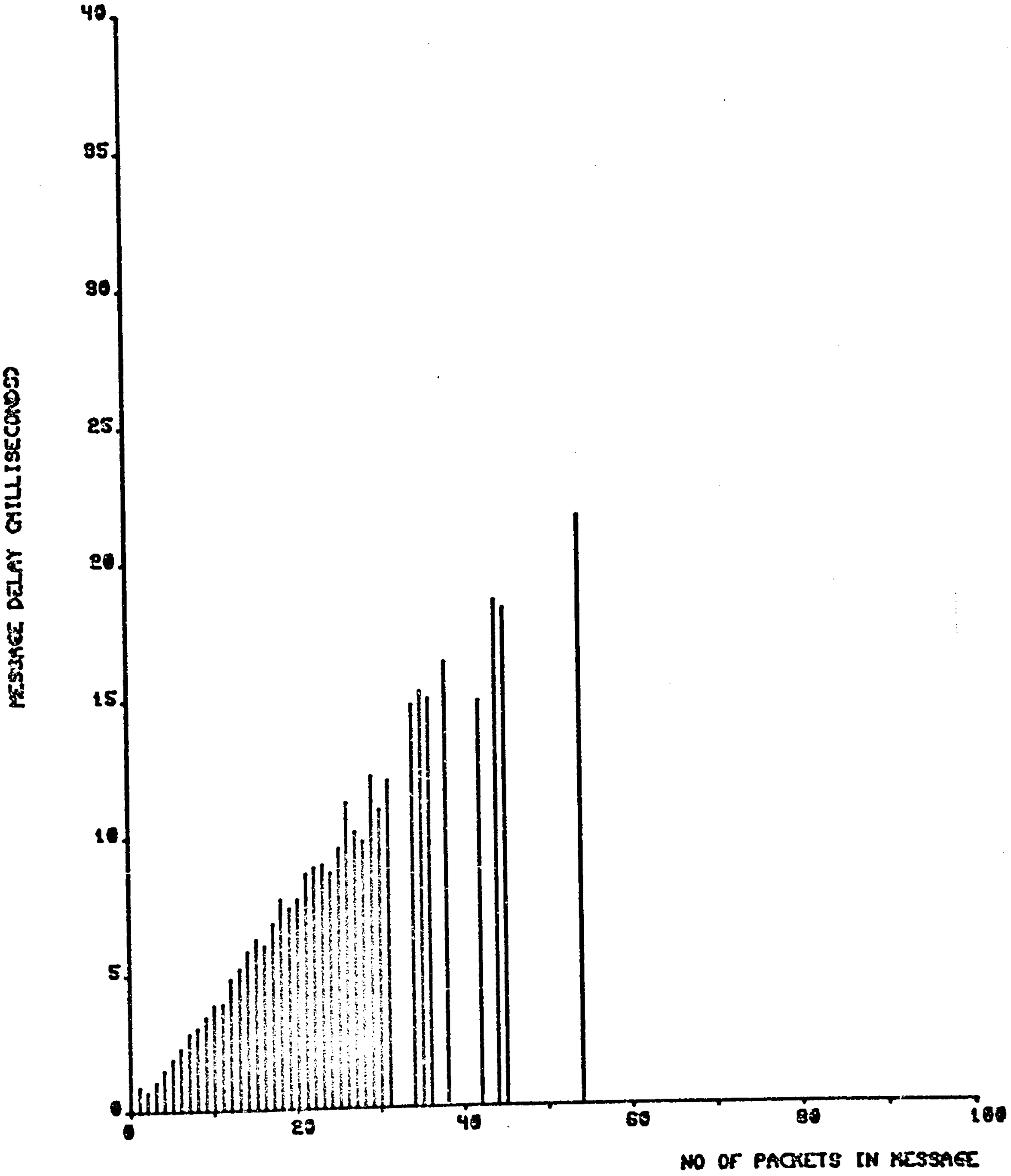


Figure 5.20 Standard Network - 1 processor, 2 memory units per node.

Effect of message length on message throughput time.

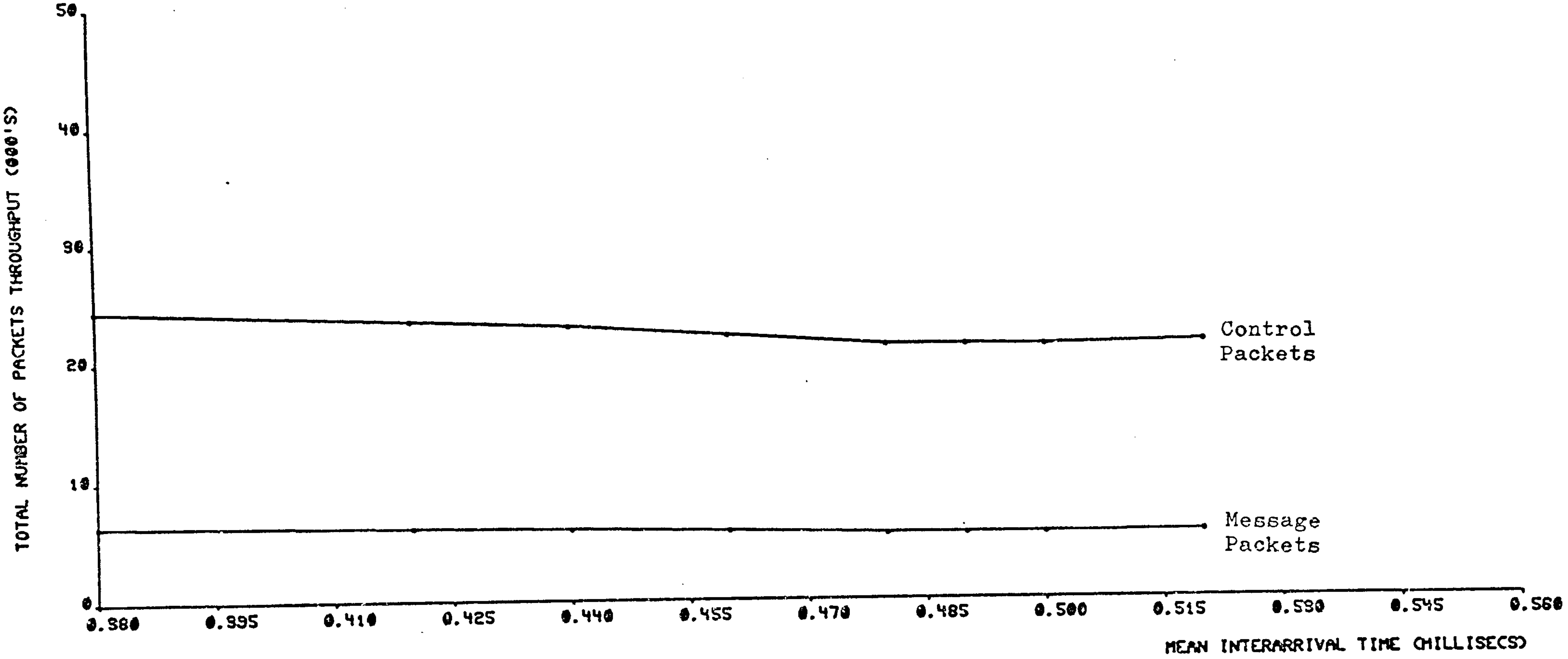


Figure 5.21 Standard Network - 1 processor, 2 memory units per node.

Effect of varying message mean interarrival rate on network traffic.

From figure 5.11 it can be shown that the mean message length is 3.5 packets compared to the theoretical mean of 3.25 packets derived in equation 5.20. Not a single occasion arose where memory was not available when required. The queue of messages in node 1 was also monitored. For the greater part the queue was empty, when it was not there was only one packet in the queue.

Figure 5.13 shows the effects of decreasing message arrivals from a mean of one every 300 μsecs to 430 μsecs. This 43% decrease in traffic resulted in 7.6 X 10^3 packets throughput dropping to 6.8 X 10^3 packets which is an 11% drop in traffic. In other words queues were cleared more quickly and the system was working near full capacity. This suggests that although the system is saturated at an interarrival time of 380 μsecs there is still capacity. This will be explained in the next chapter.

Figure 5.14 shows the number of packets in the system at the time of being sampled awaiting processing by the network. This indicates that there is no need for the facility of queueing within the network.

5.6 Effects of the number of processors/memory modules

Processing time is made up of several factors: waiting for processor/memory; direct memory usage under processor control - this quantity being proportional to the packet length;

executing I/O - this quantity includes the time needed for both waiting on an I/O queue and for actual execution of I/O; waiting on the ready queue.

Figures 5.15 to 5.21 show the effect of reducing the number of processors in the standard network from two to one, and the memory modules from three to two. This resulted in a throughput drop from 7 X 10^3 packets to 5.6 X 10^3 packets. This 22% drop was not greater since the processors on the standard network were under-utilised. In the case of the node with one processor the processor usage figures were as follows:

	node 1	node 2	node 3
proc 1	46%	59%	43%

When the standard network processor/memory speeds were dropped from 100 nsecs to 1 μsec the effects were more noticeable as shown in figures 5.22 to 5.28. Packet throughput dropped from 7 X 10^3 packets to 1.6 X 10^3 packets. Individual processor usage was as follows:

	node 1	node 2	node 3
proc 1	74%	83%	63%
proc 2	72%	81%	61%

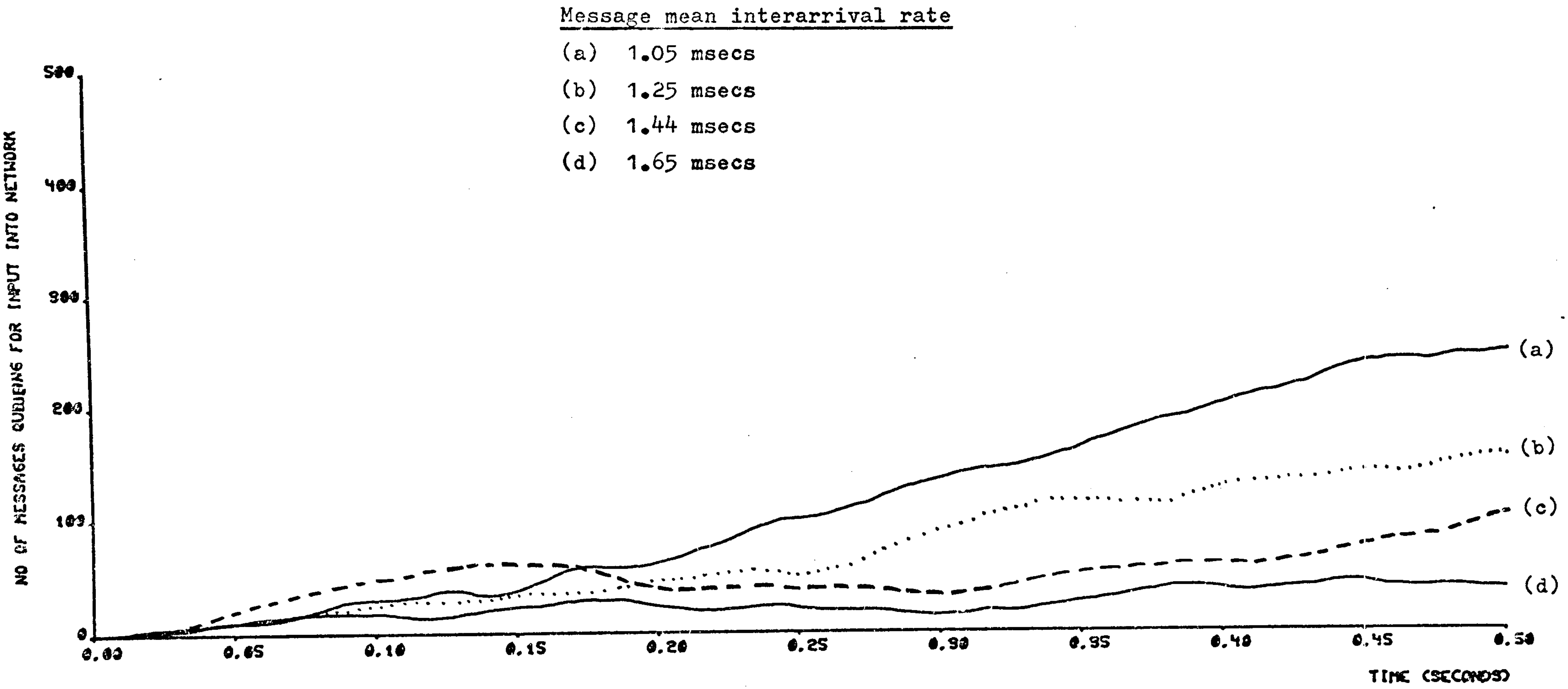


Figure 5.22 EWMA Standard Network - 1 μsecond memory/processor. Effect of varying message mean interarrival rate on message queue for input into network.

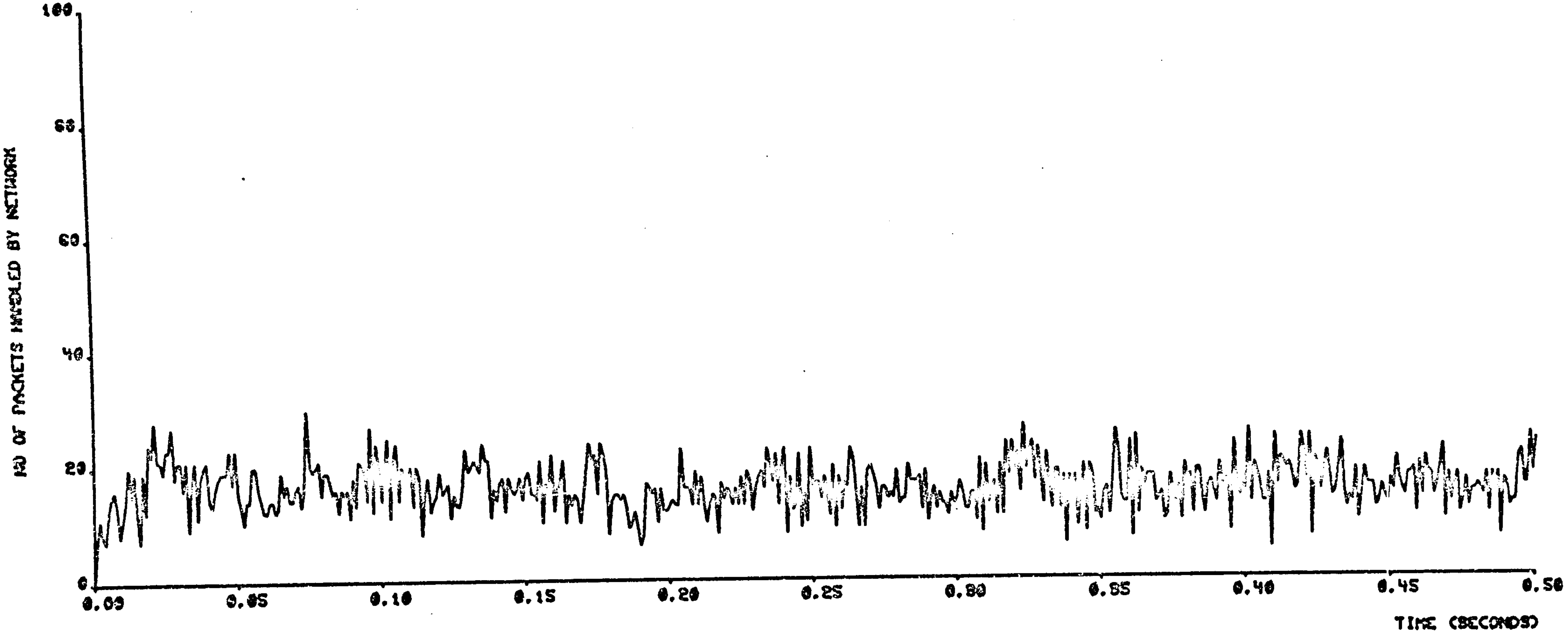


Figure 5.23 Standard Network - 1 /usecond memory/processor. Number of packets processed by network per millisecond sample interval.

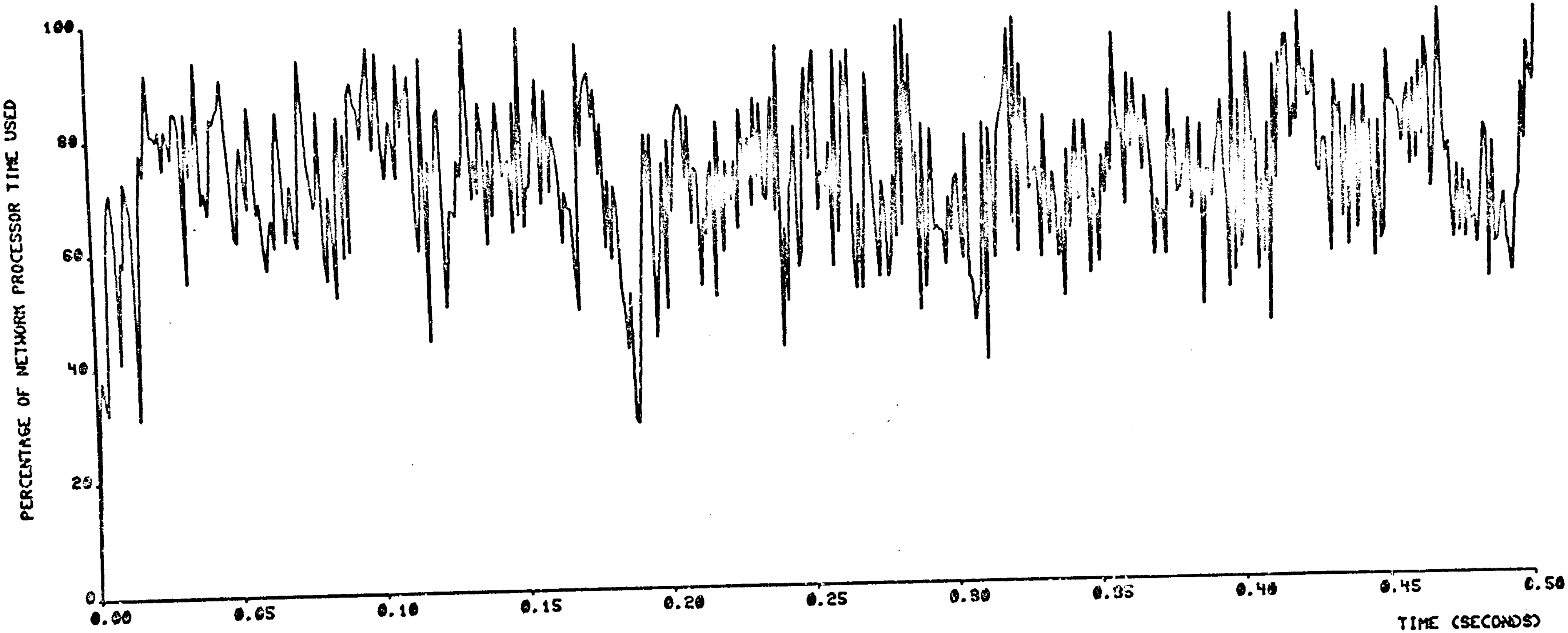
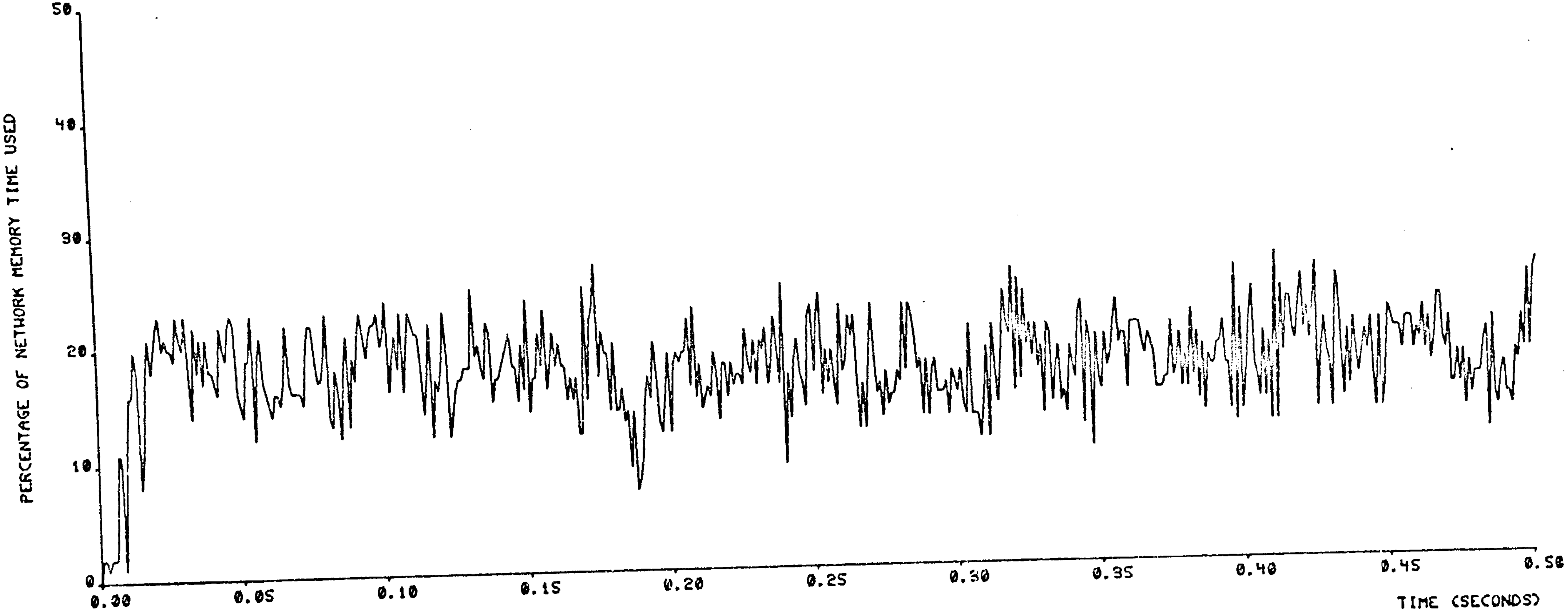
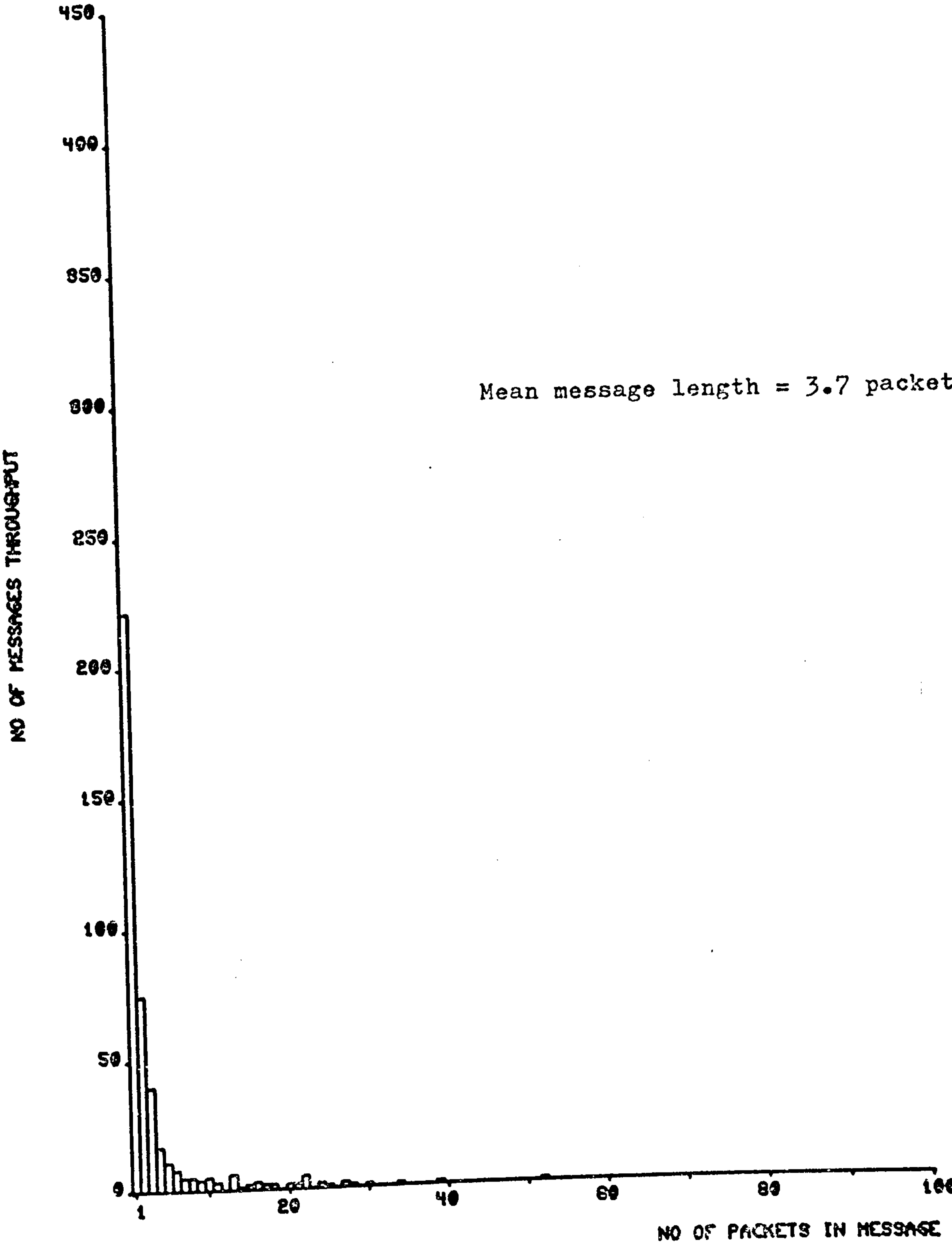


Figure 5.24 Standard Network - 1 μsecond memory/processor. Percentage of network processor time used per millisecond sample interval.



<u>Figure 5.25</u> Standard Network - 1 µsecond memory/processor. Percentage of network memory time used per millisecond sample interval.



Mean message length = 3.7 packets

Figure 5.26 Standard Network - 1μsecond memory/processor
Distribution of message lengths
sent through network.

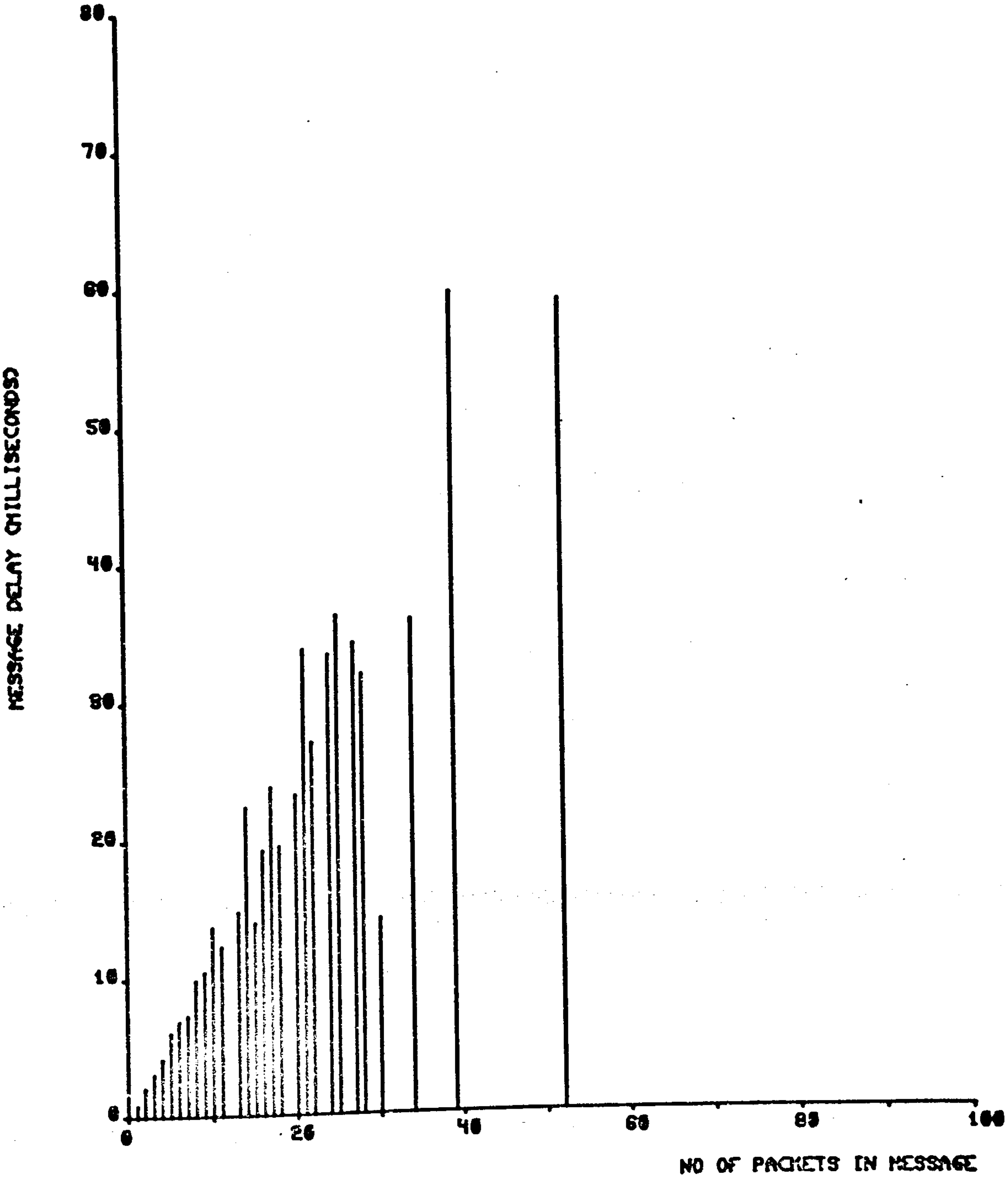


Figure 5.27 Standard Network - 1 μsecond memory/processor. Effect of message length on message throughput time.

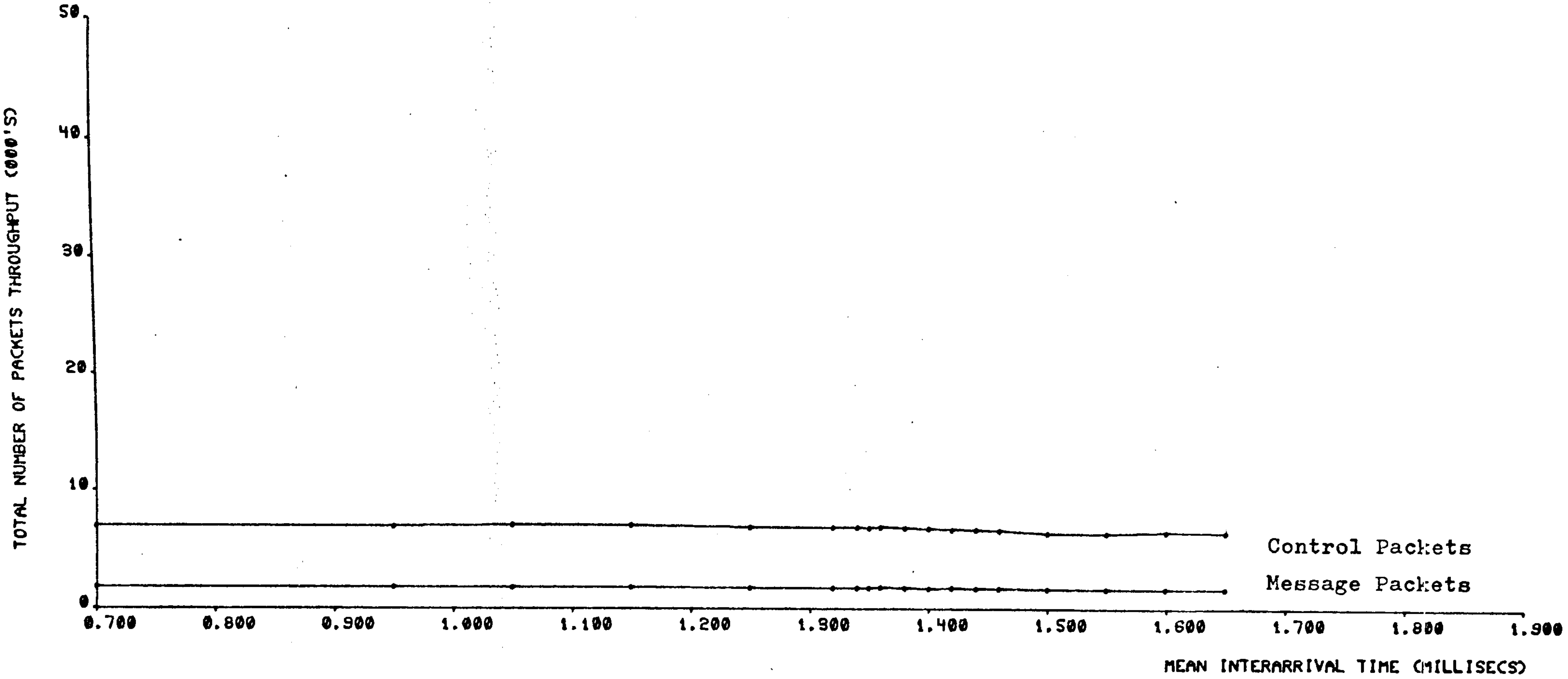


Figure 5.28 Standard Network - 1 µsecond memory/processor. Effect of varying message mean interarrival rate on network traffic.

5.7 Memory module size

Figures 5.29 to 5.35 show the effect of reducing the memory module size from 1K words to 128 words i.e. from storing eight packets to storing one packet. Comparing figure 5.7 and figure 5.28 it can be seen that saturation takes place at about the same inter-message arrival rate. When 1K words memory modules were used memory was always available when needed. Not a single occasion arose when it was not available. However, when 128 words memory modules were used memory was not available on one or two occasions. Given that over 7 X 10^3 packets were handled in 0.5 secs and two memory accesses were required at both the source and the destination nodes, this figure is negligible.

The simulation did not take into account the extra accesses that would have been required to extract a packet from a particular memory location in the 1K words memory module. So it can be seen that the 128 words memory module would not only be easier to implement, but would also be faster since access is easier and the probability of memory contention non-existant i.e. where two processors wanted to access different packets in the same memory module.

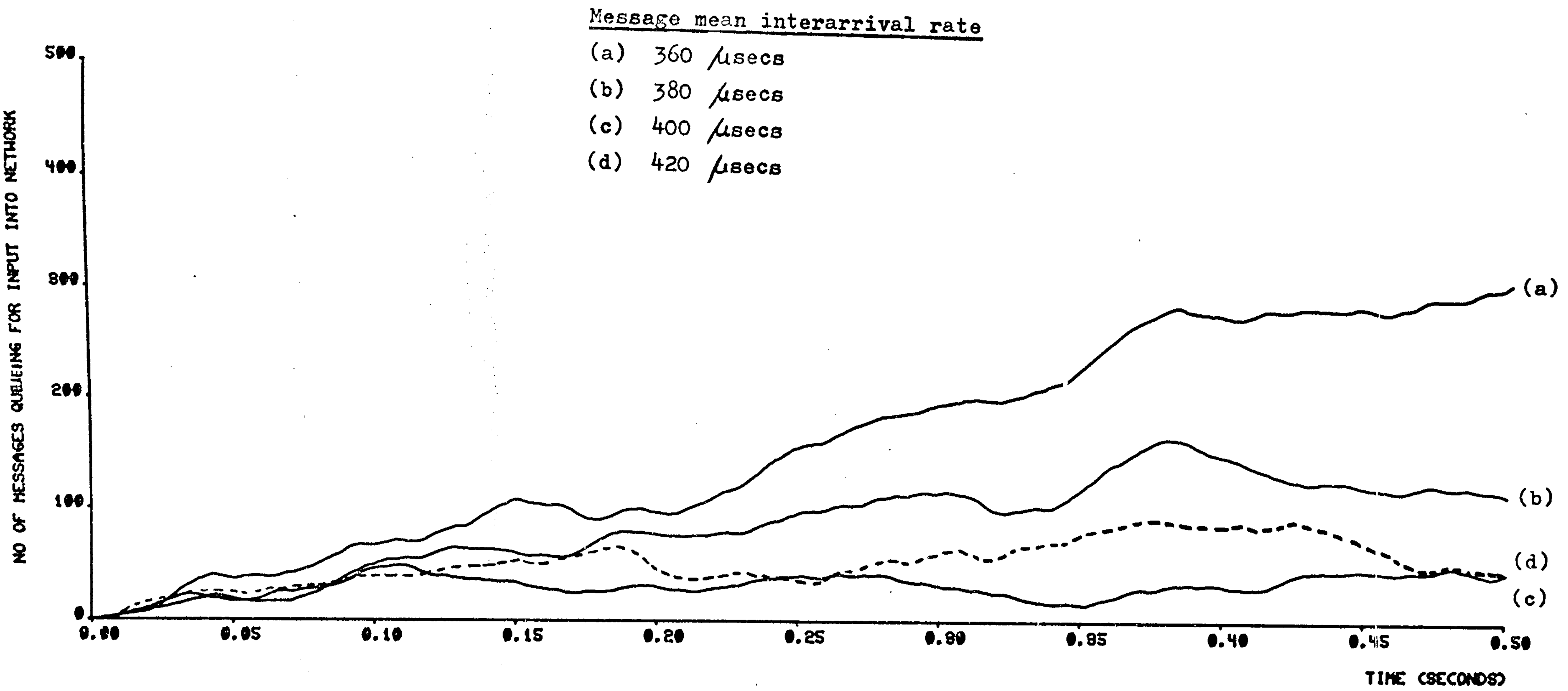


Figure 5.29 EWMA Standard Network - 128 word memory module. Effect of varying message mean interarrival rate on message queue for input into network.

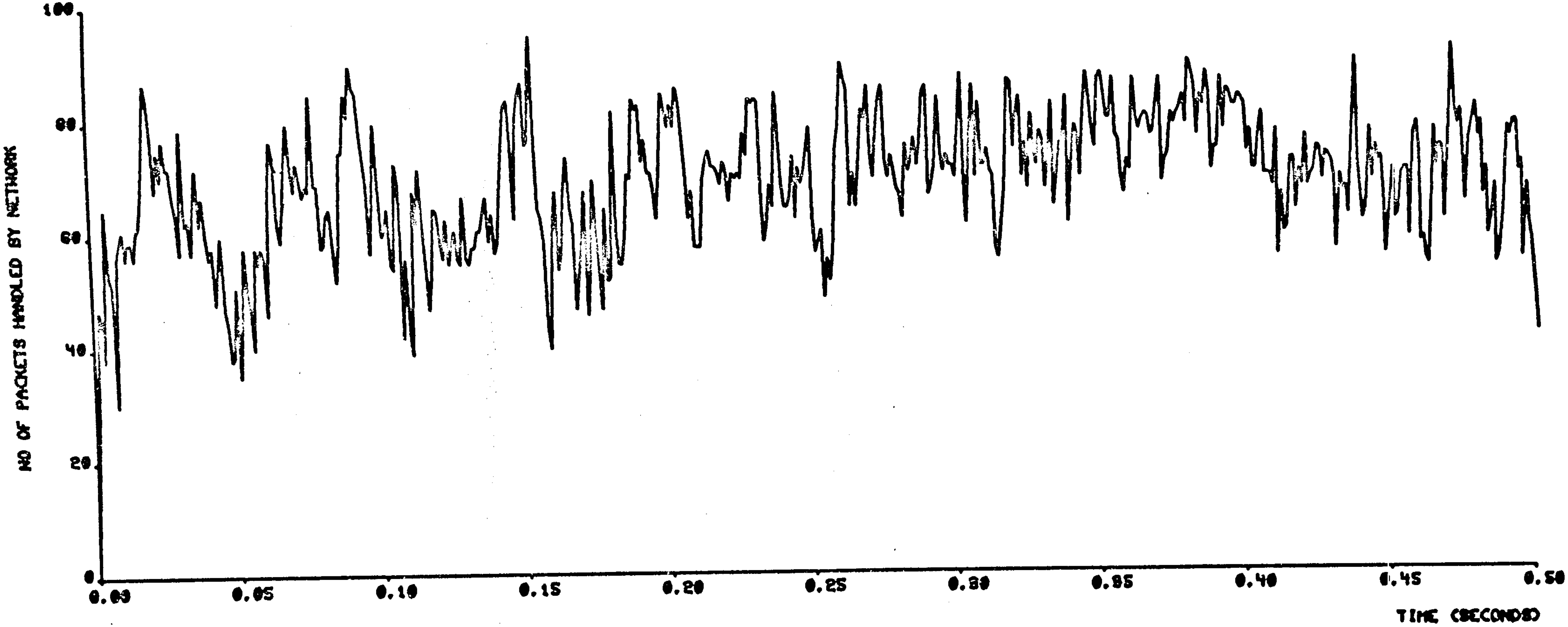


Figure 5.30 Standard Network - 128 word memory module. Number of packets processed by network per millisecond sample interval.

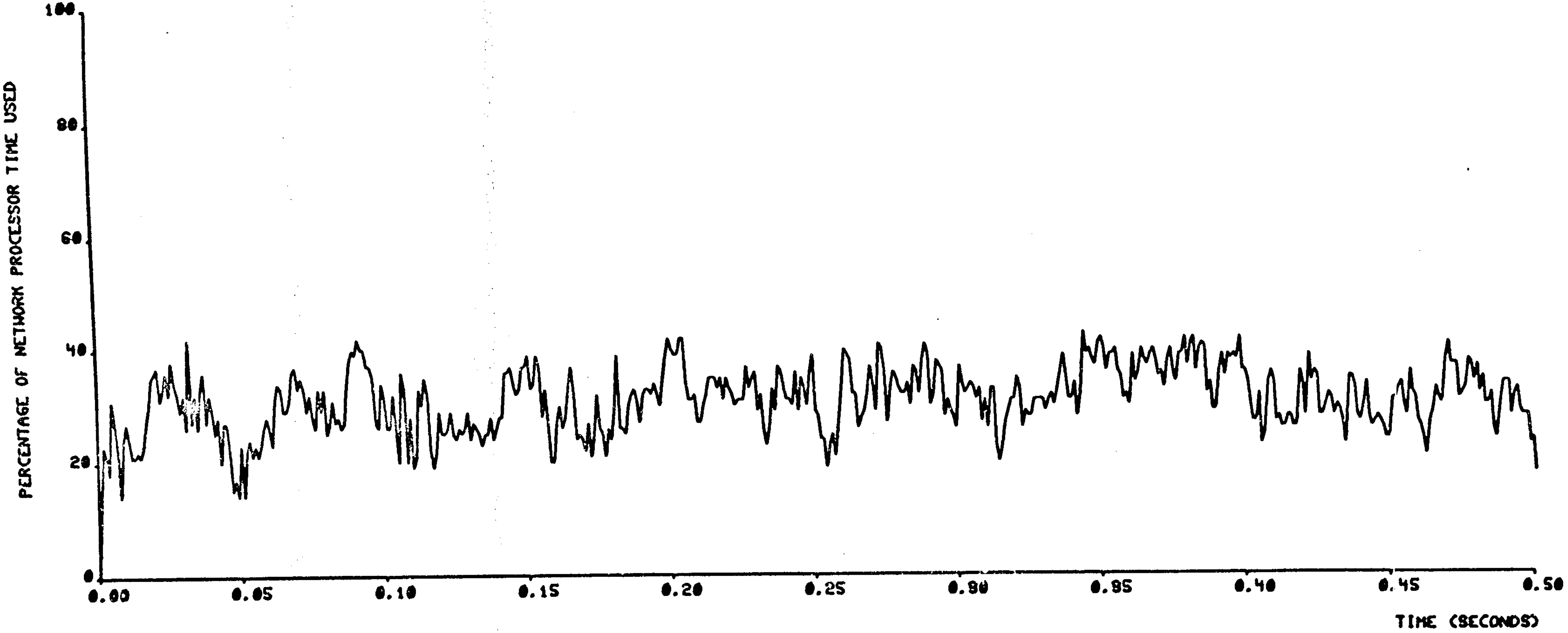
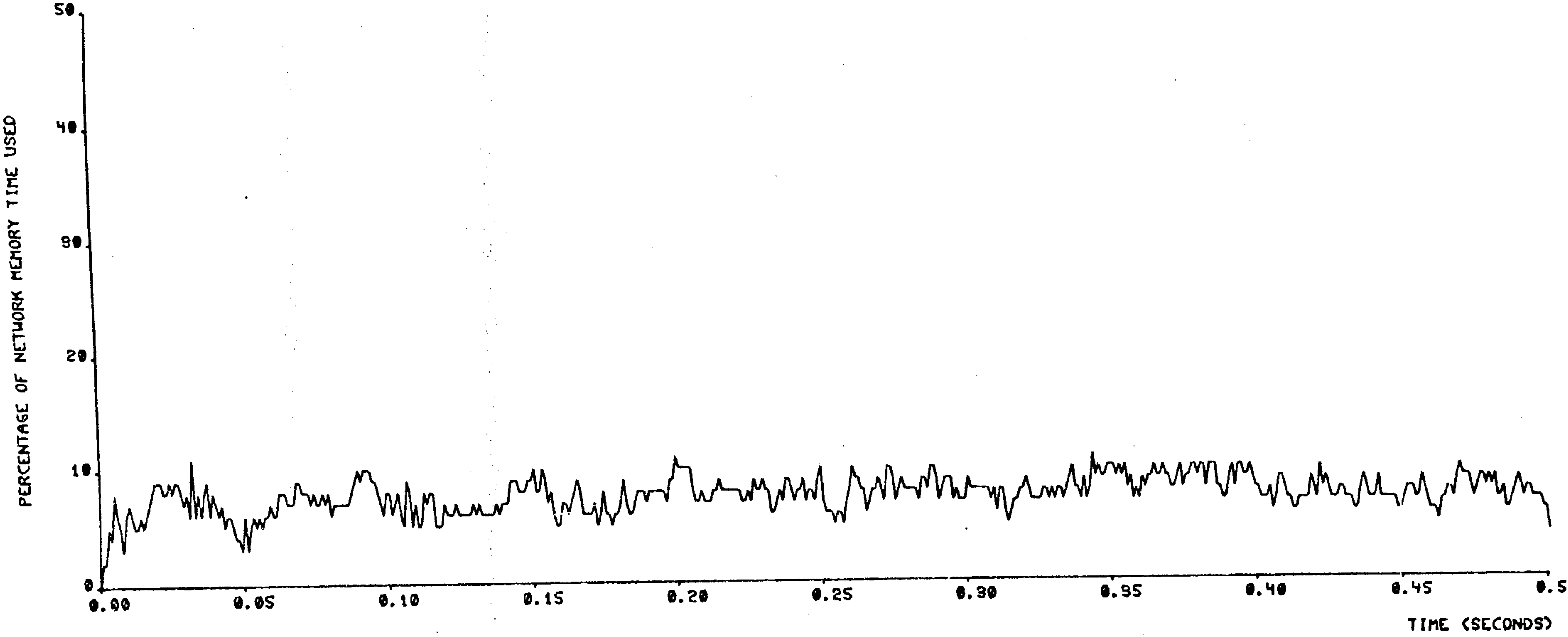
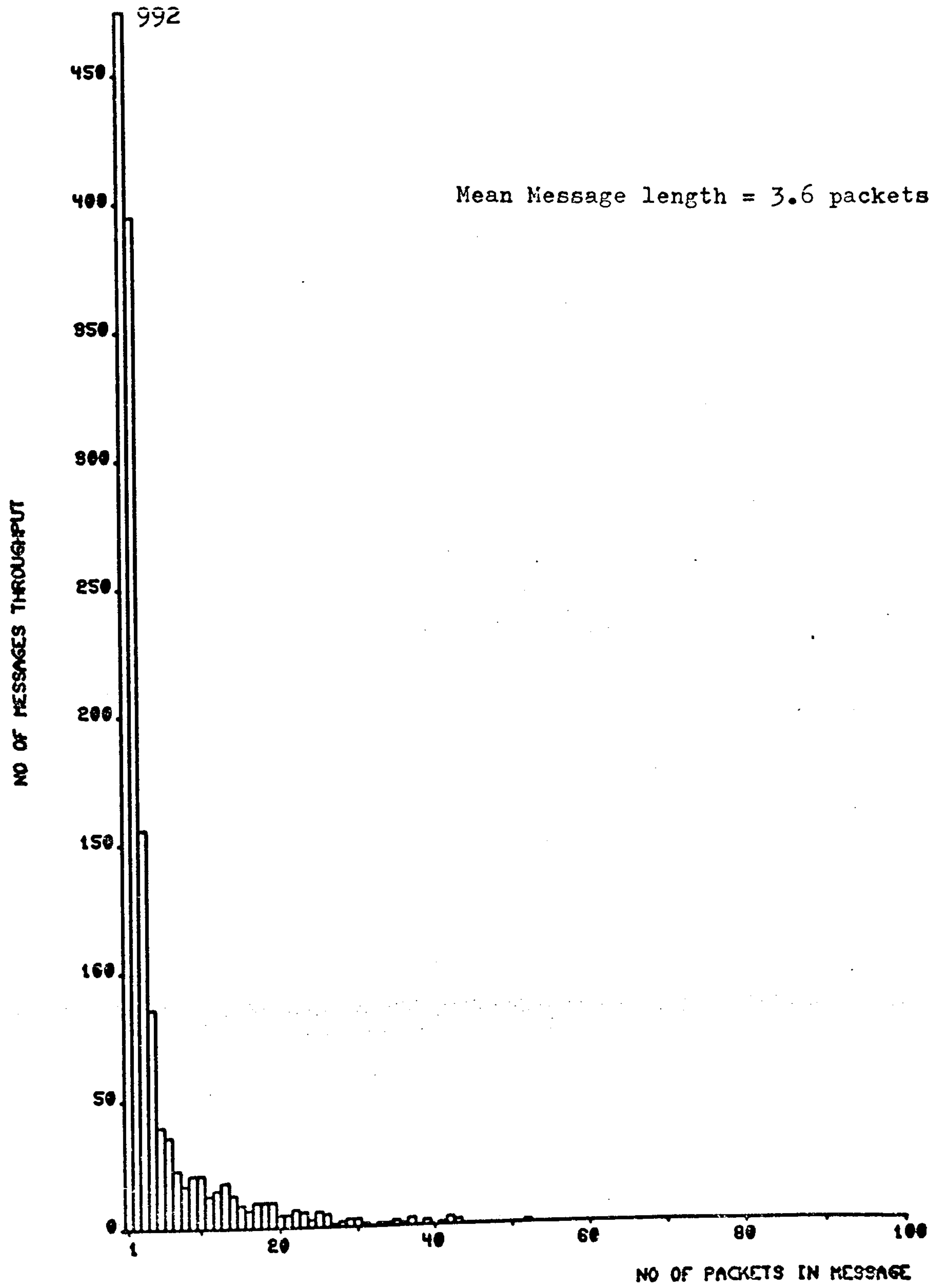


Figure 5.31 Standard Network - 128 word memory module. Percentage of network processor time used per millisecond sample interval.



<u>Figure 5.32</u> Standard Network - 128 word memory module. Percentage of network memory time used per millisecond sample interval.



Mean Message length = 3.6 packets

Figure 5.33 Standard Network - 128 word memory module. Distribution of message lengths sent through network.

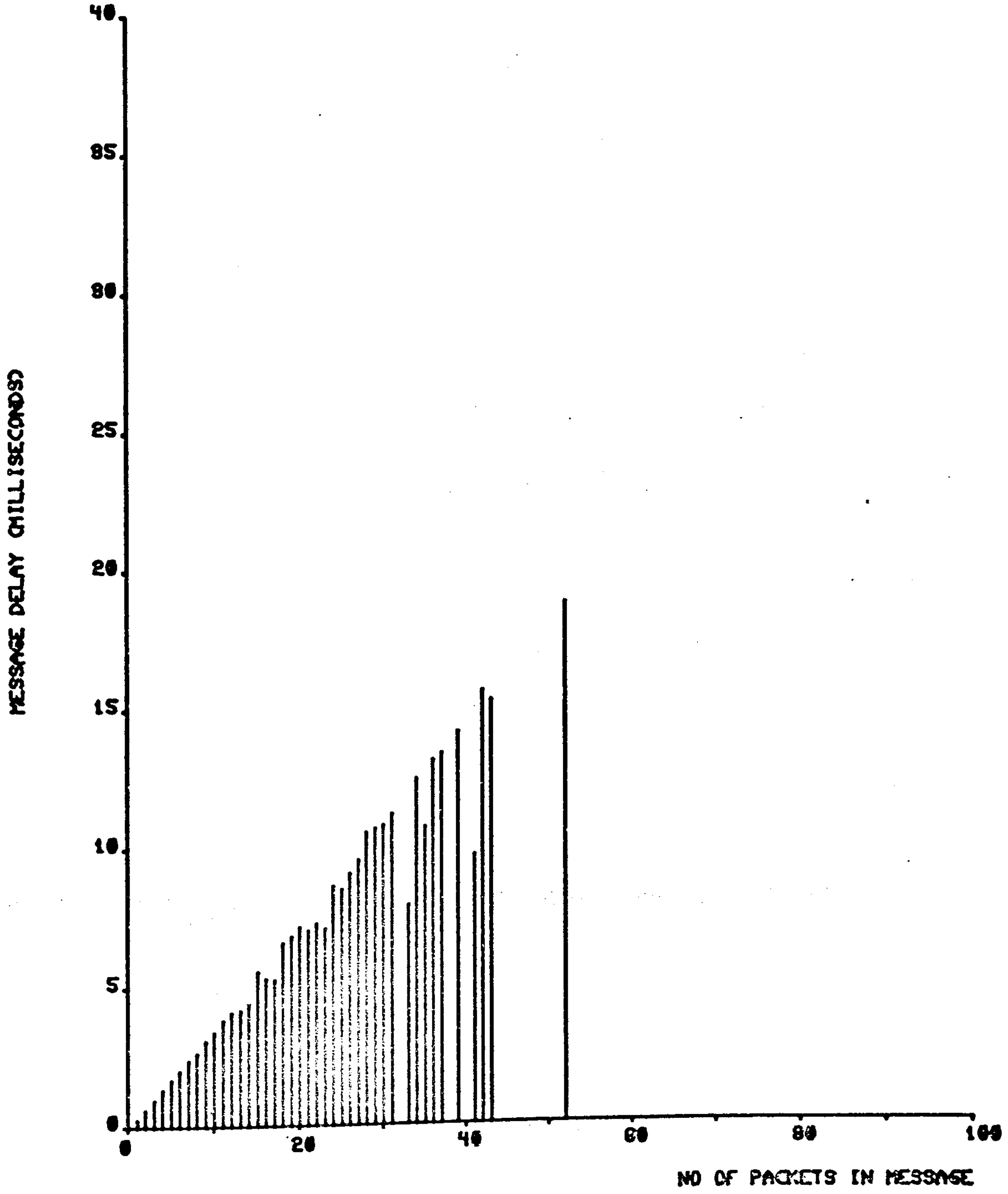


Figure 5.34 Standard Network - 128 word memory module. Effect of message length on message throughput time.

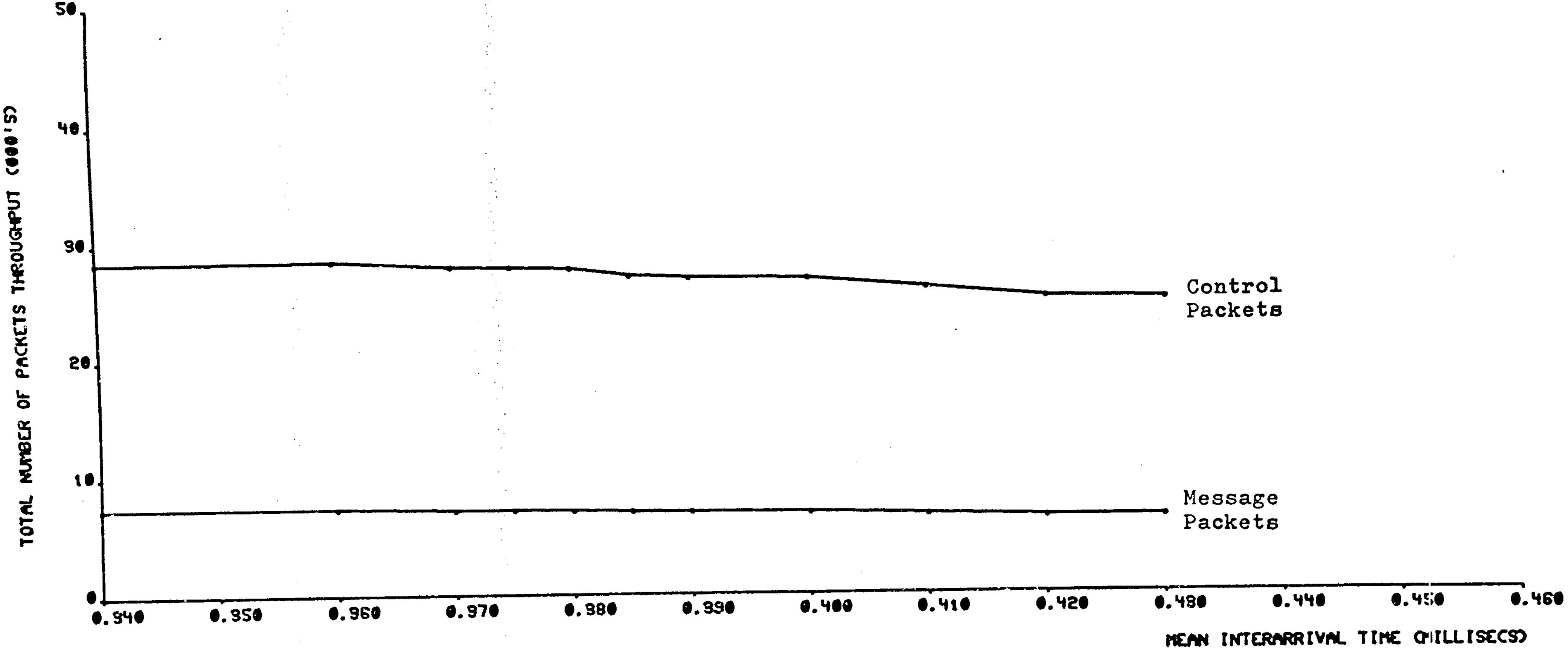


Figure 5.35 Standard Network - 128 word memory module. Effect of varying message mean interarrival rate on network traffic.

5.8 Conclusions

The effects of node hardware have been considered in this chapter. It is apparent that the addition of more processors and memory modules will improve the response time on message throughput to a certain degree. Similarly using slower memory/processors will result in a much poorer throughput. It should be noted that 2 or 3 processors produce better throughput after which the line speeds are the limiting factor. The failure of a processor in a three processor node would not have a catastropic effect and the throughput should remain fairly constant. Since there is no throughput difference due to memory size, it would seem sensible to have memory units capable of holding one packet and there should be at least as many memory modules as there are processors. Also since the network will not be working under such heavy constant workload as have been imposed, surges in traffic intensity should be dealt with quite adequately.

CHAPTER 6

HIGH LEVEL NETWORK PARAMETERS

6.1 Introduction

This chapter is devoted to a consideration of the effects of the high level network. The chapter begins by discussing a problem that was encountered early on regarding a simple buffer lockup. Several solutions to this problem are suggested. The remaining part of the chapter is concerned with how many packets the system should be able to support and reasons are given to explain why the network is not supporting the maximum packet throughput. There are four sets of results contained in this chapter.

1) 3 hosts/node (Figures 6.4 - 6.10)

2) 2 node network (Figures 6.12 - 6.18)

3) 1 megabit lines (Figures 6.19 - 6.25)

4) 100 kilobit lines (Figures 6.26 - 6.31)

6.2 Buffer Lockups

During early simulation runs a simple buffer lockup was encountered as shown in Figure 6.1.

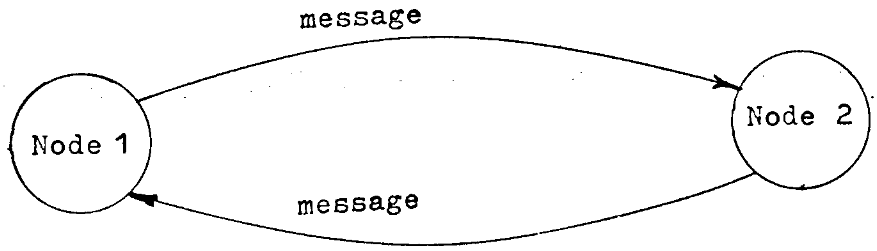


Figure 6.1 Simple buffer lockup

This arose in the situation when two nodes began to send a message packet to each other. When the packets reached the destination nodes acknowledgements could not be sent since the output buffers would retain a copy of the message packet until a positive acknowledgement was received. This resulted in a rapid buildup in queue lengths on the two nodes that issued these message packets. Four solutions were considered to solve this problem:

1) Double buffering
2) Retaining copy of packet in memory
3) Increasing buffer size to accommodate a message packet and a control packet
4) Using a message buffer and a control packet buffer.

Under the current arrangement, whenever a packet needs to be retransmitted it would have to be brought back into memory to have a bit set to indicate that it was a duplicate packet which in turn would necessitate the checksum being recalculated. With the first solution to the lockup i.e. double buffering, the secondary copy would need a bit set and another checksum carried out. Although an error situation would be

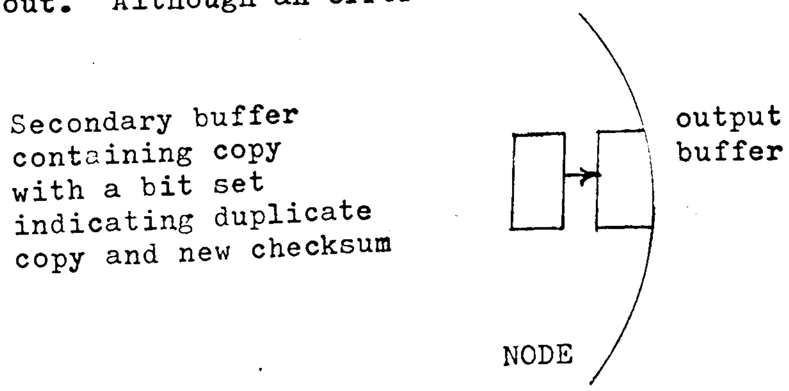


Figure 6.2 Double buffering

rare this would result in twice as much work being done every time a packet is sent out. Figure 6.2 shows double buffering.

The second solution would have a high cost. For each O/P buffer (there could be up to 10 O/P buffers for hosts and node communication) there would have to be a backup memory unit. The multibus would also be unnecessarily complicated to handle the extra memory modules.

The third solution involves extending the buffer size as shown in Figure 6.3 to accommodate a message packet and a control packet i.e. extending the buffer size from 128 words (for a 128 word packet) to 128 + 8 words. By rotating the shift register the correct number of bits a control packet could be slotted in.

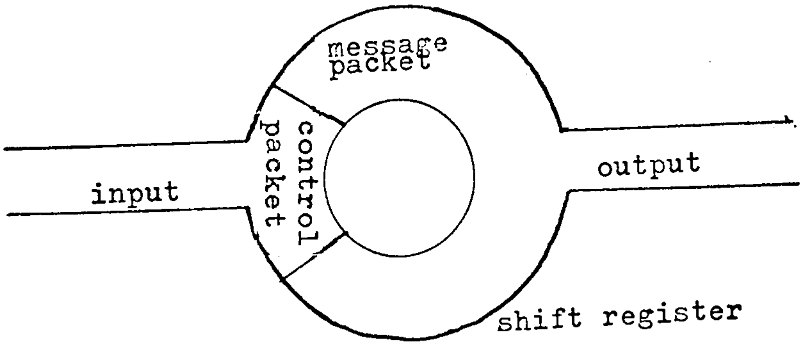


Figure 6.3 Circular shift register capable of storing message and control packet

A modification of solution one is also possible where there are separate buffers for message and control packets. A switch determines which buffer will be used for output. For the purposes of the simulation a copy of the message packet is placed into a secondary buffer and the control packet transmitted via the primary buffer.

6.3 Line utilisation

In a fully connected network where each packet can be transmitted directly from source to destination node, except in the case of a failure, the number of packets that the network can support is given approximately by:

$$\frac{\text{line speed X no of nodes}}{\text{message packet length + control packet length}}$$

where the packet lengths are given in bits. The control packet length must be taken into account since no other message packet may be transmitted until the current message packet has been acknowledged. For the topology considered the best throughput rate, where the acknowledgement may be placed into the output buffer immediately, is given by:

$$\frac{10000000 \text{ X } 3}{(128 + 8) \text{ X } 8} \simeq 27.6 \text{ X } 10^3 \text{ packets/second.}$$

The worst case is where a message packet transmission has

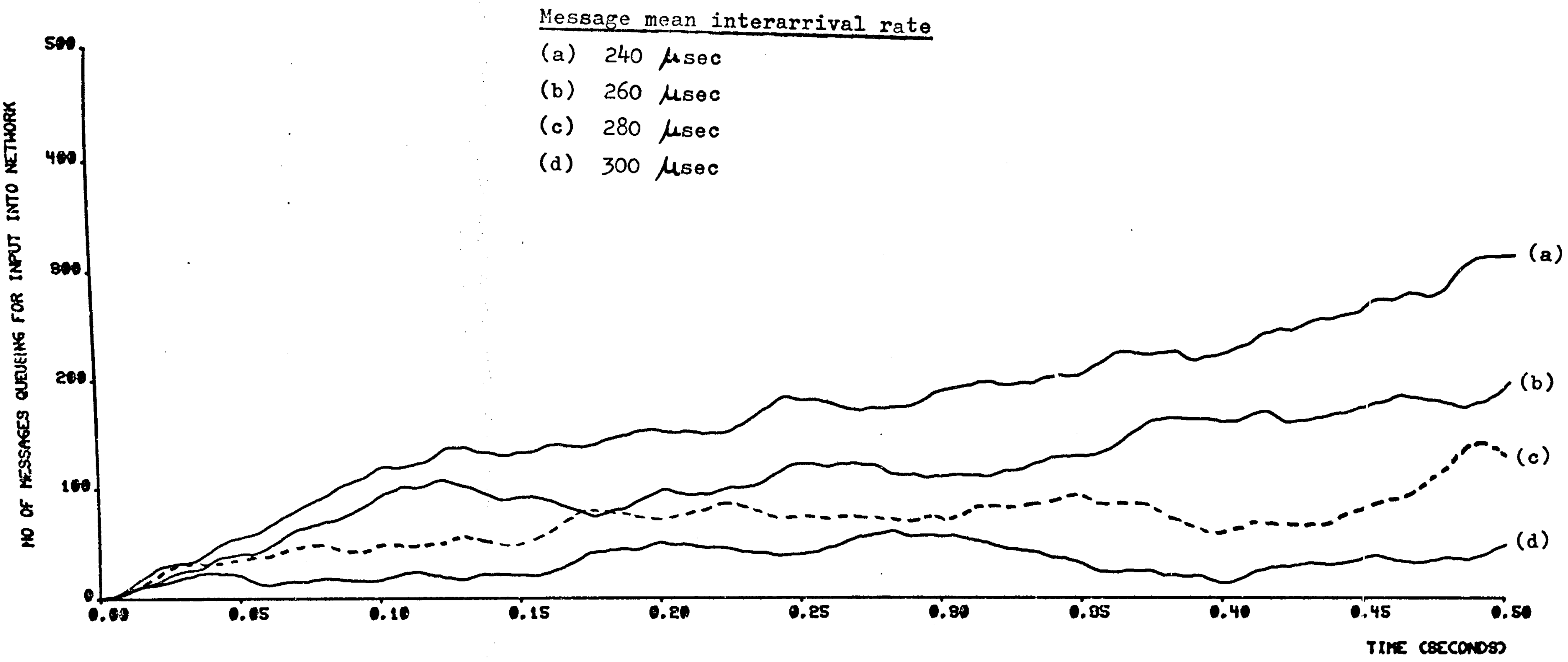


Figure 6.4 EWMA Standard Network - 3 hosts/node.
Effect of varying message mean interarrival
rate on message queue for input into network.

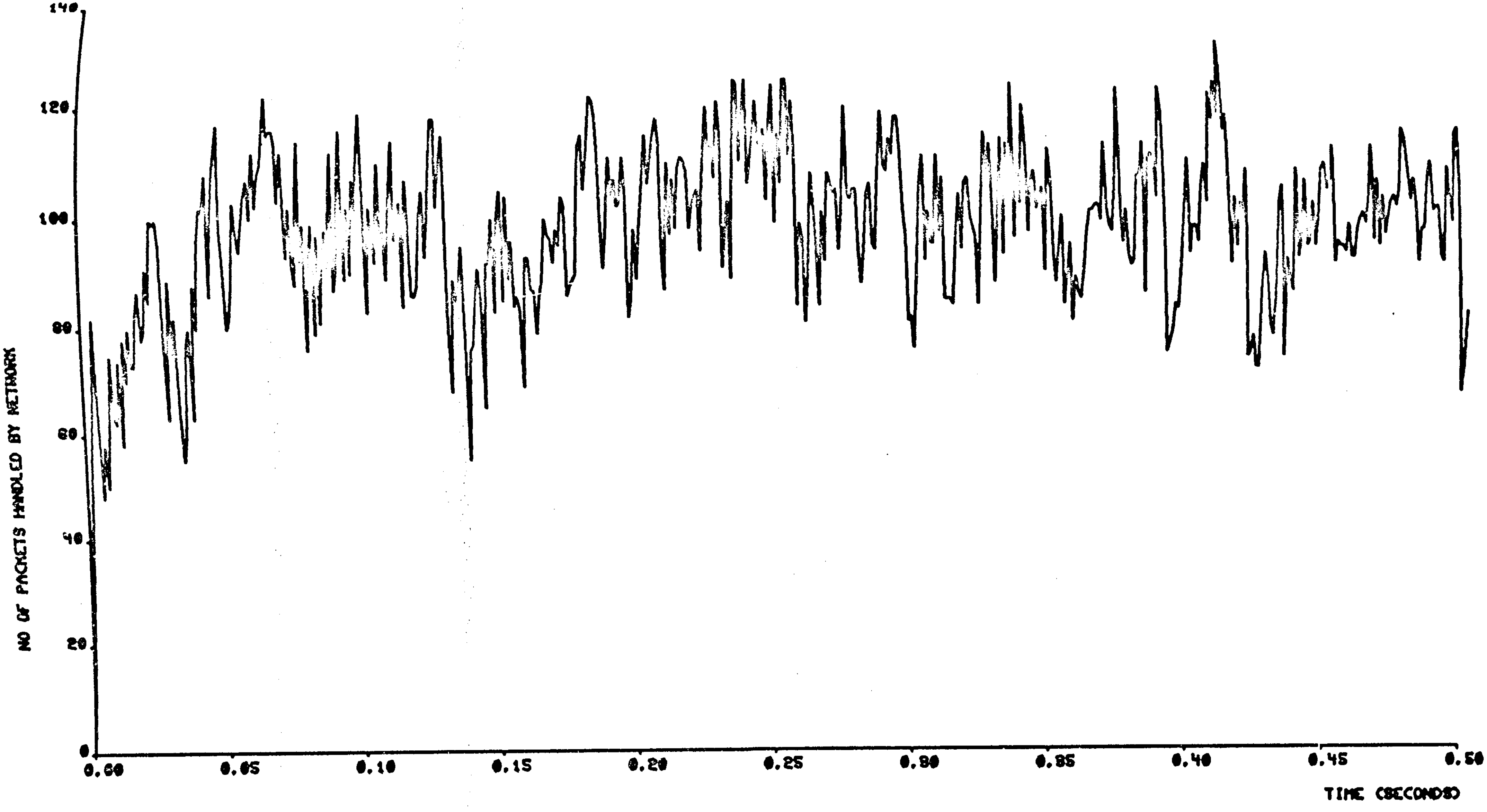


Figure 6.5 Standard Network - 3 hosts/node.
Number of packets processed by network
per millisecond sample interval.

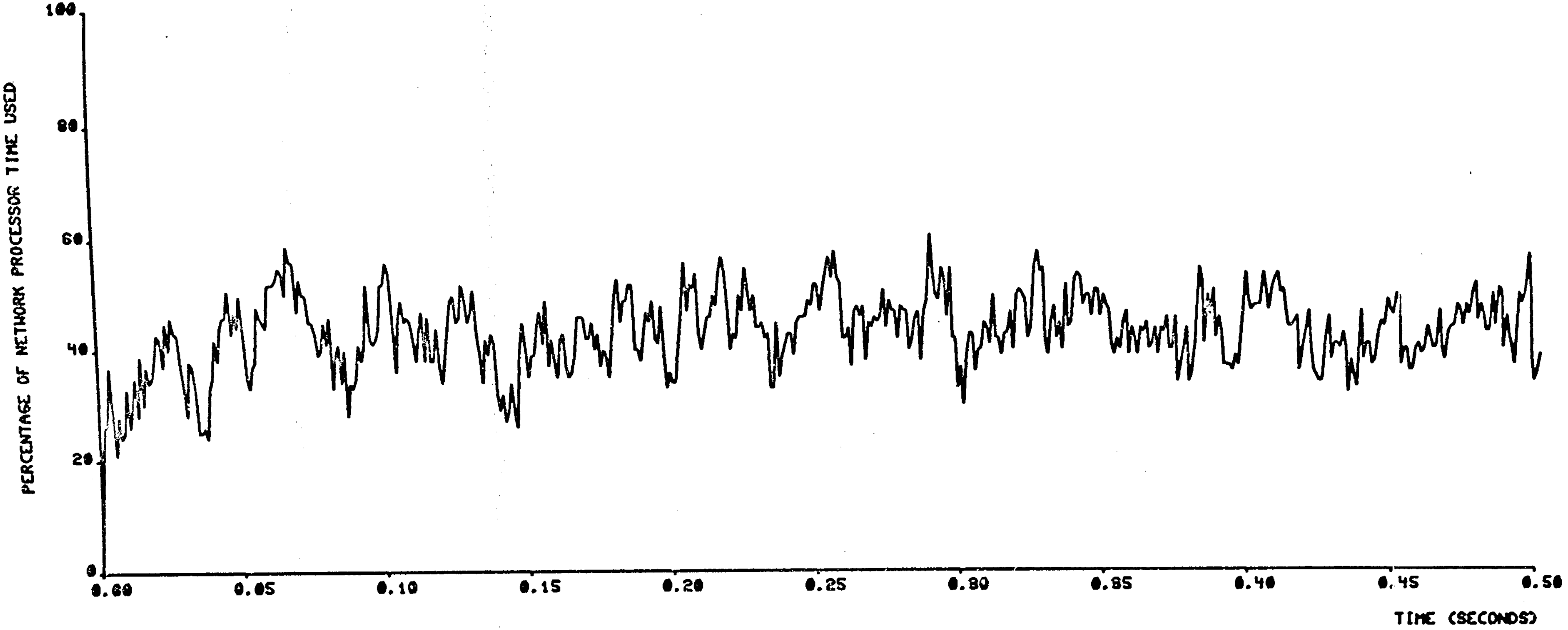


Figure 6.6 Standard Network - 3 hosts/node.
Percentage of network processor time used
per millisecond sample interval.

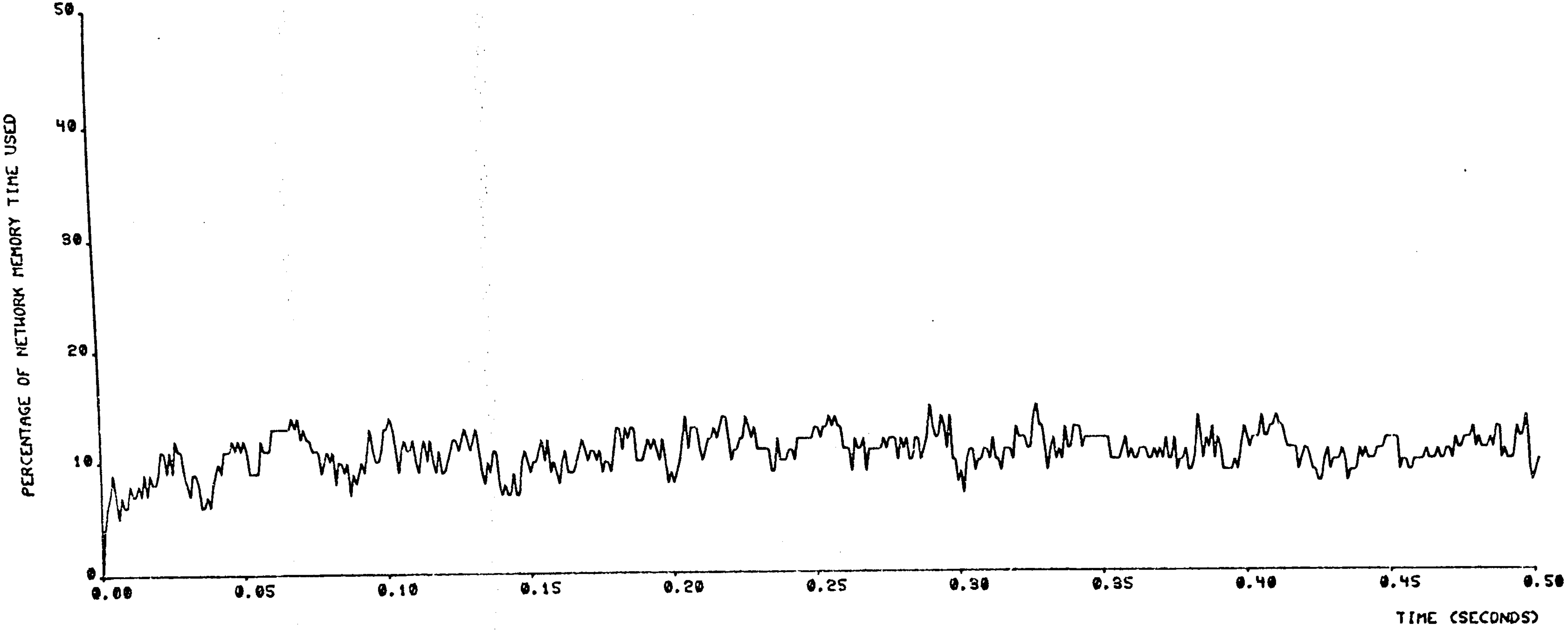


Figure 6.7 Standard Network - 3 hosts/node.
Percentage of network memory time used
per millisecond sample interval.

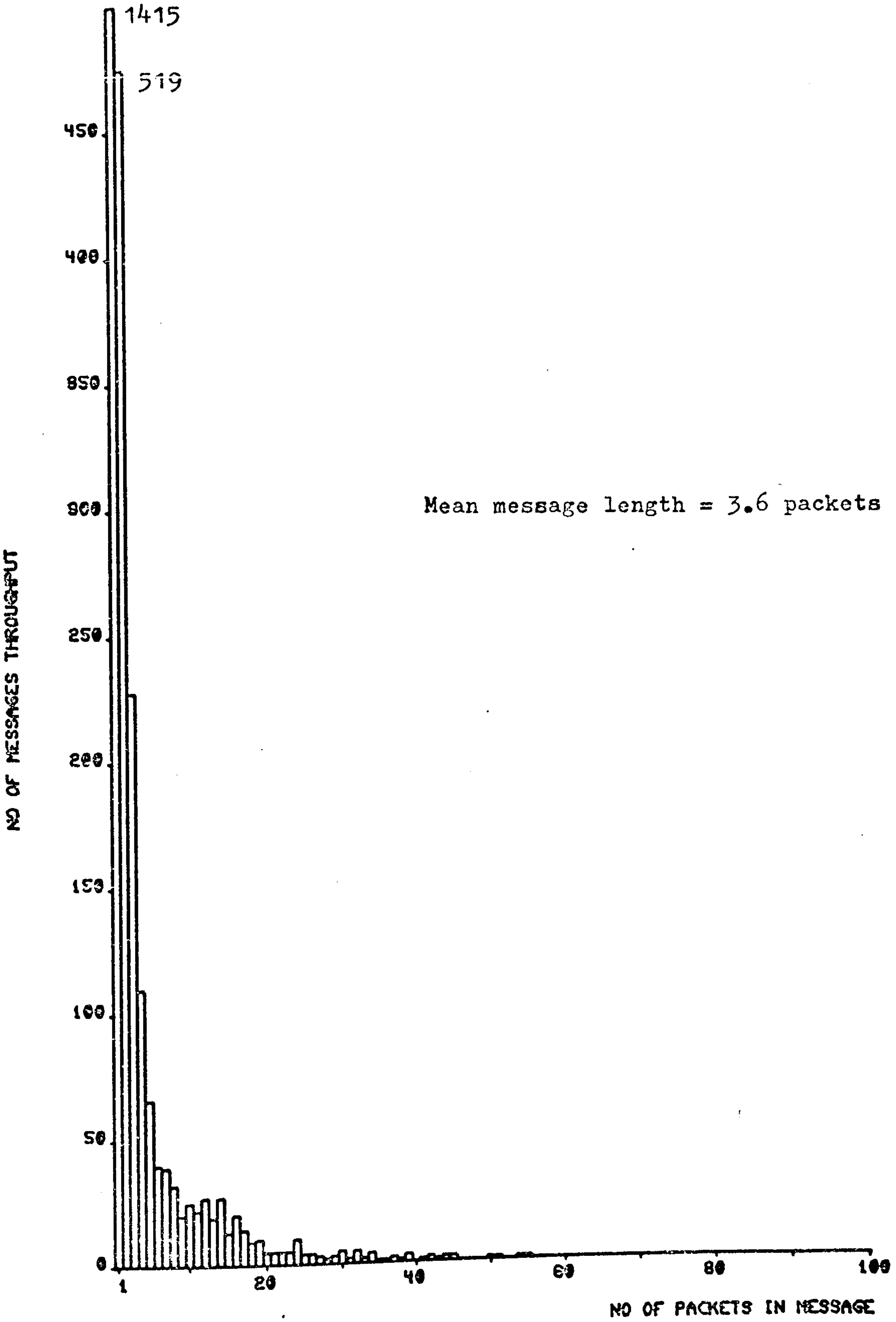


Figure 6.8 Standard Network - 3 hosts/node.
Distribution of message lengths
sent through network.

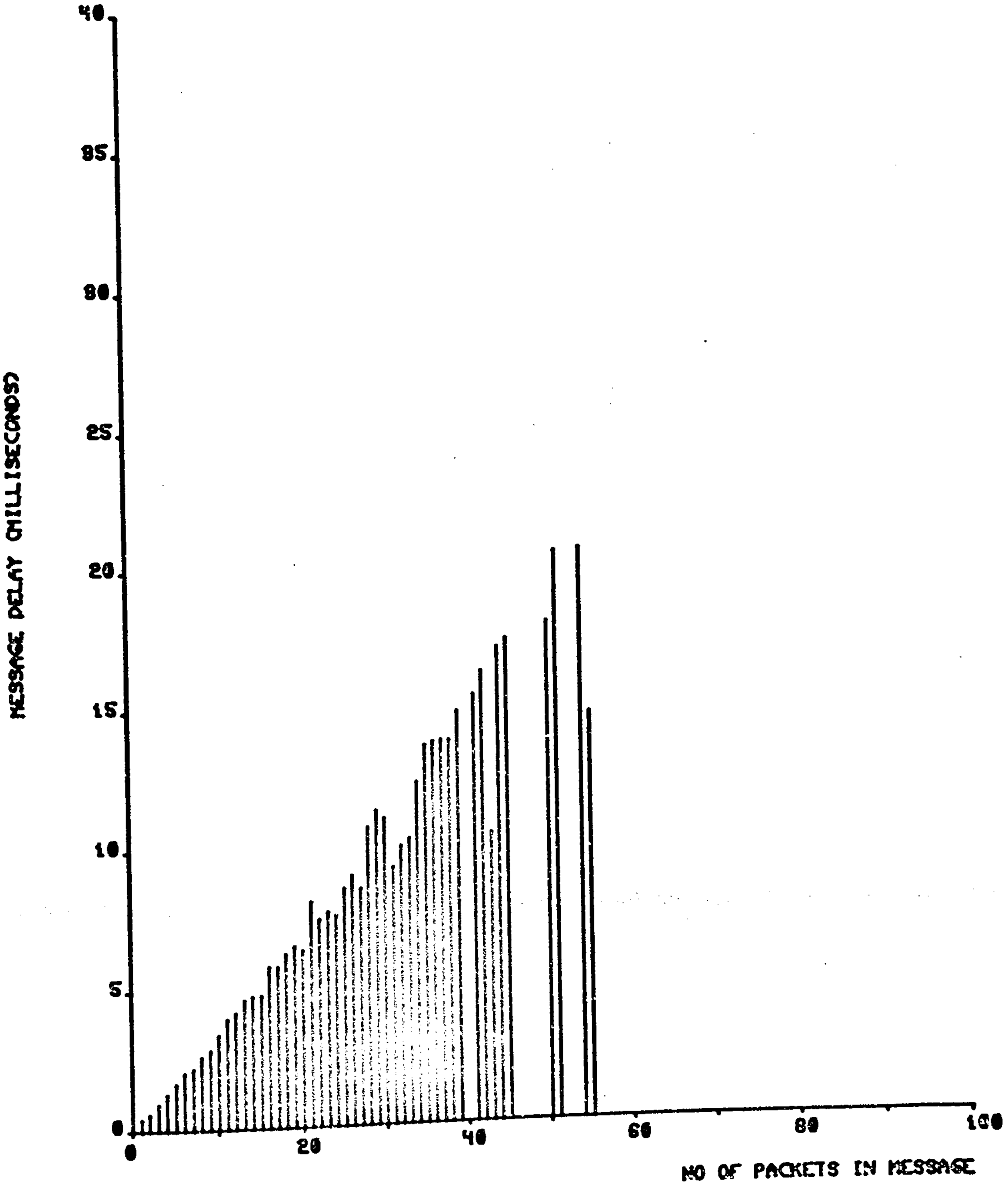


Figure 6.9 Standard Network - 3 hosts/node. Effect of message length on message throughput time.

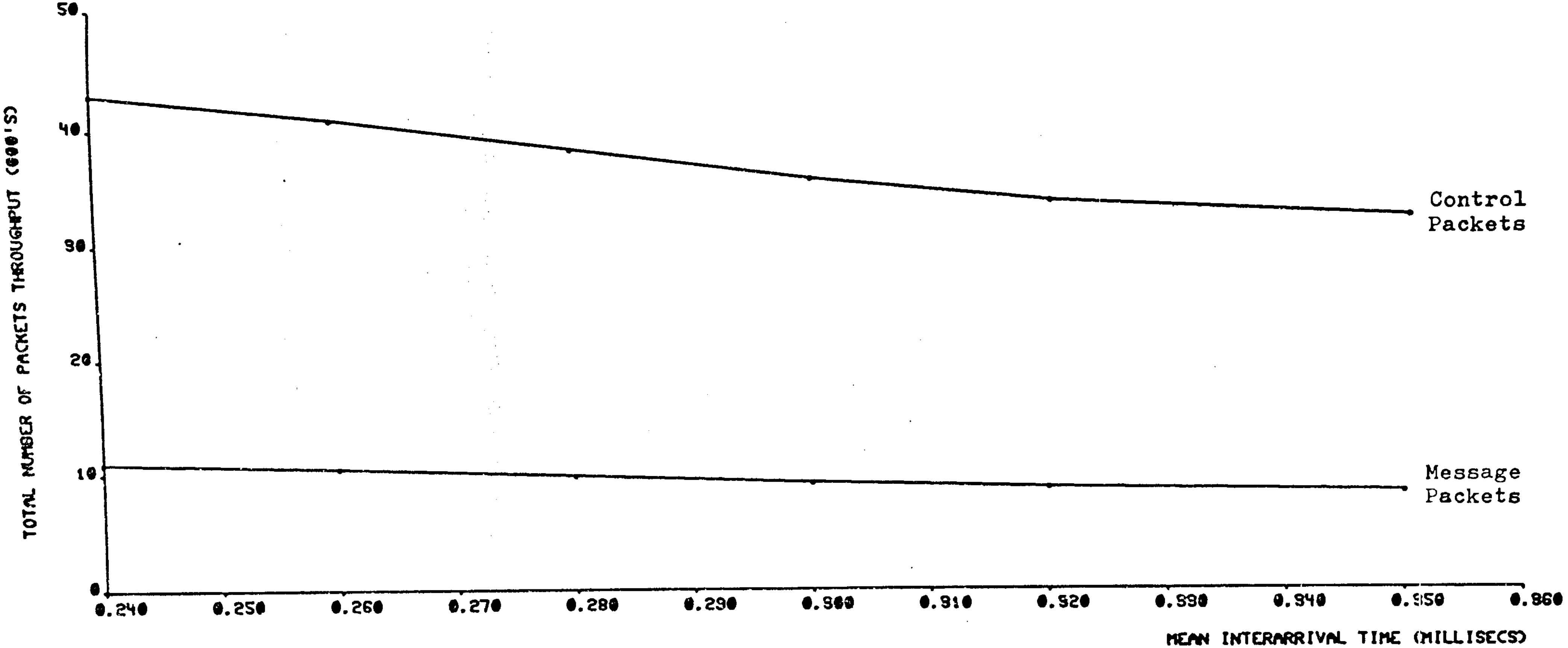


Figure 6.10 Standard Network - 3 hosts/node. Effect of varying message mean interarrival rate on network traffic.

been started just as the acknowledgement needs to be sent which is given by:

$$\frac{10000000 \text{ X } 3}{(128 + 128 + 8) \text{ X } 8} = 14.2 \text{ X } 10^3 \text{ packets/second}$$

giving a mean throughput of 20.9 X 10^3 packets/second.

With the standard network (2 hosts/node) saturation occurred when the network was throughputting 14 X 10^3 packets/second as shown in figure 5.13. When the standard network topology was modified to support 3 hosts per node saturation occurred when the network was throughputting 20 X 10^3 packets/second as shown in figure 6.18.

However, with both network topologies the system is not supporting as many packets as it should be, although the minimum number of packets are being throughput in the first case and the mean number of packets in the second case. Consider the following case of the standard network with only one host per node. Once the host has sent the first packet of the message it does not send any further packets until it receives a 'send next packet' from the destination host as shown in figure 6.11. It has been assumed that there is only one message being transmitted from the host at any time.

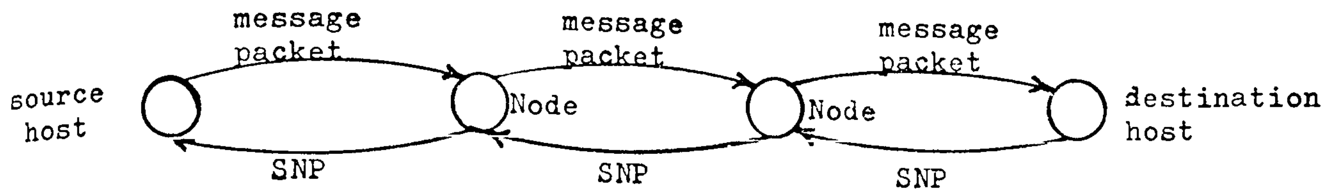


Figure 6.11 Number of paths transversed during a typical host/host transaction

Assuming 10 Megabit lines, it takes the message packet:

$$\frac{3 \times 128 \times 8}{10000000} = 310 \mu\text{secs}$$

to reach the destination host and a further 19 μsecs to receive the 'send next packet', assuming no other delays. In other words it takes 330 μsecs before the host transmits its next packet.

Looking at the source host → node transaction, the host buffer may be freed after the node has sent an acknowledgement to the message packet. This transaction takes 110 μsecs, or a third of the time for which the simulation has tied up the buffer. This implies that the host has only one user at any time, whereas the host may be supporting many terminals. It can be seen from the trend of the throughput figures that the addition of another host per node would maximise line usage. Alternatively, each node could support 2 hosts each of 2 terminals.

Figures 6.12to 6.18 show that this would be the case. Whereas

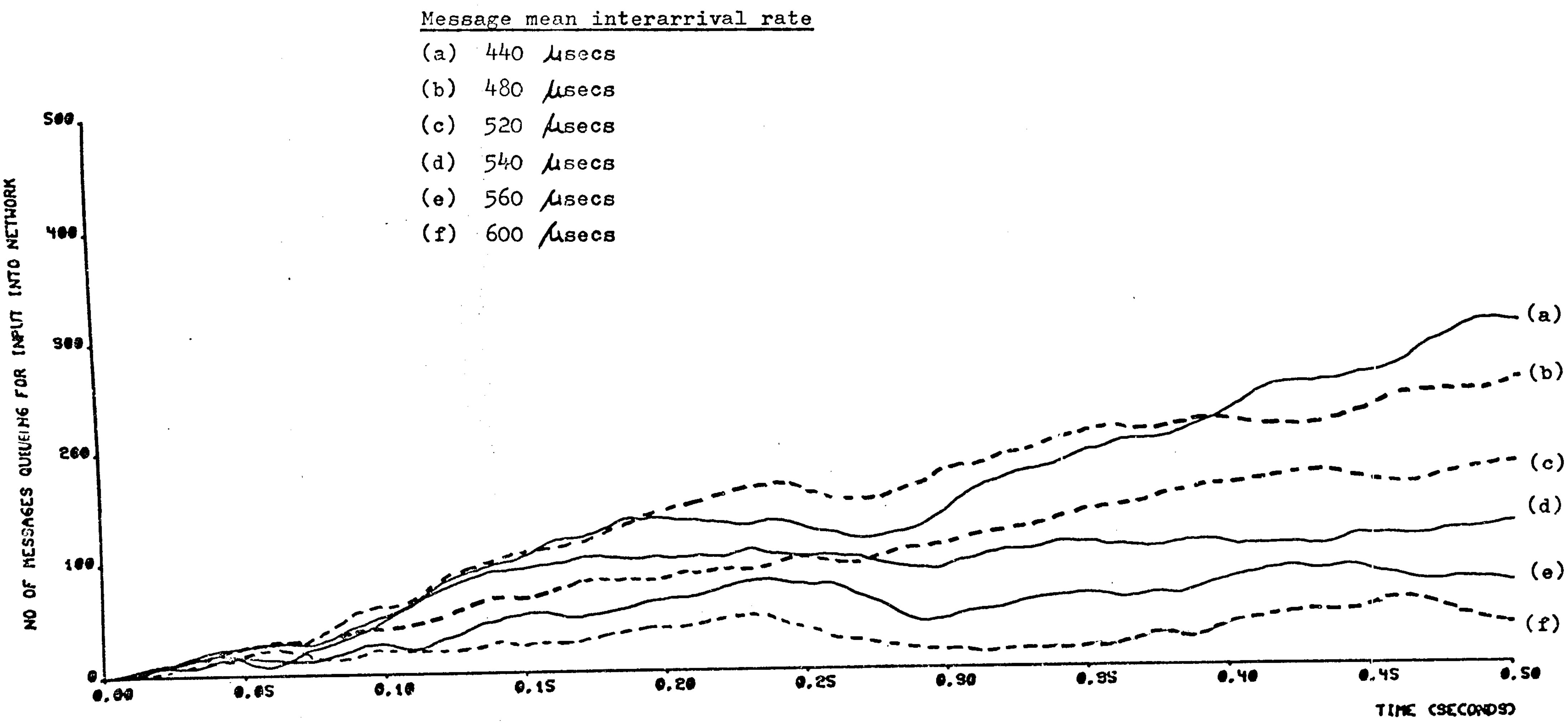


Figure 6.12 EWMA Standard Network - 2 nodes.
Effect of varying message mean interarrival rate on message queue for input into network.

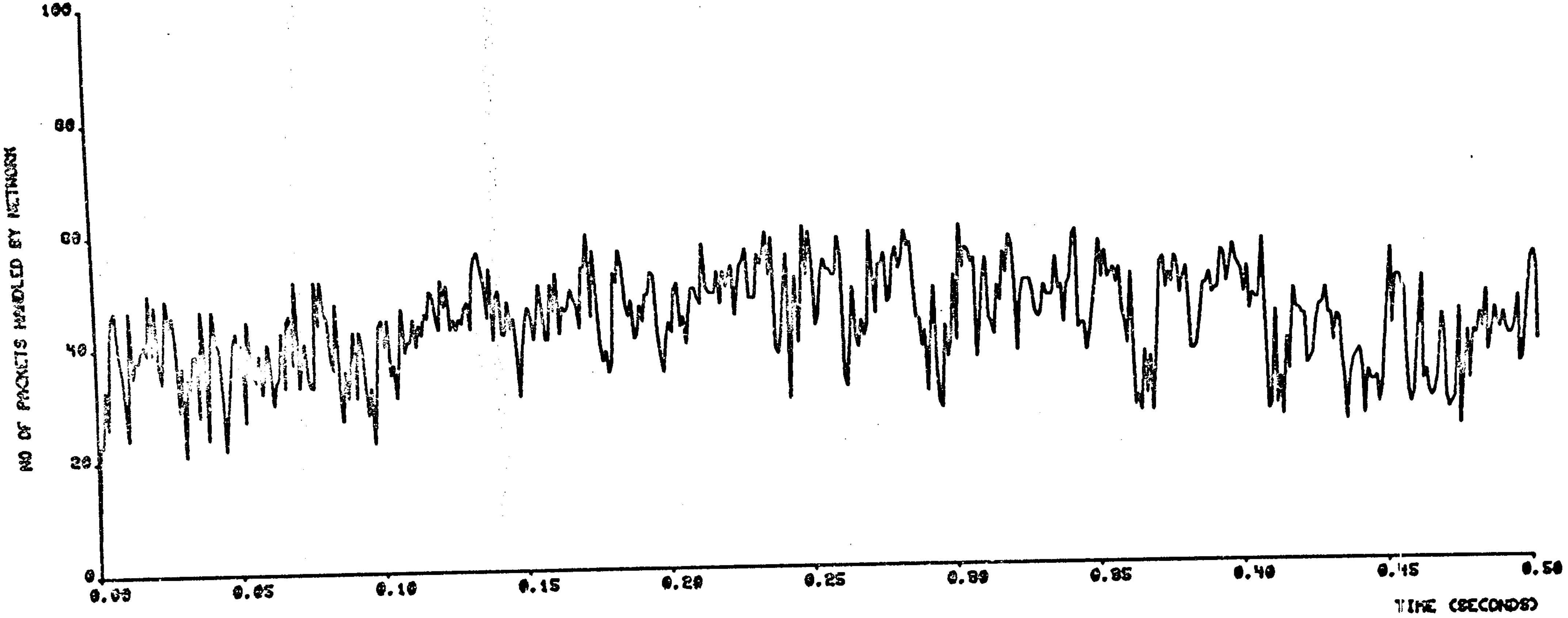


Figure 6.13 Standard Network - 2 nodes.
Number of packets processed by network
per millisecond sample interval.

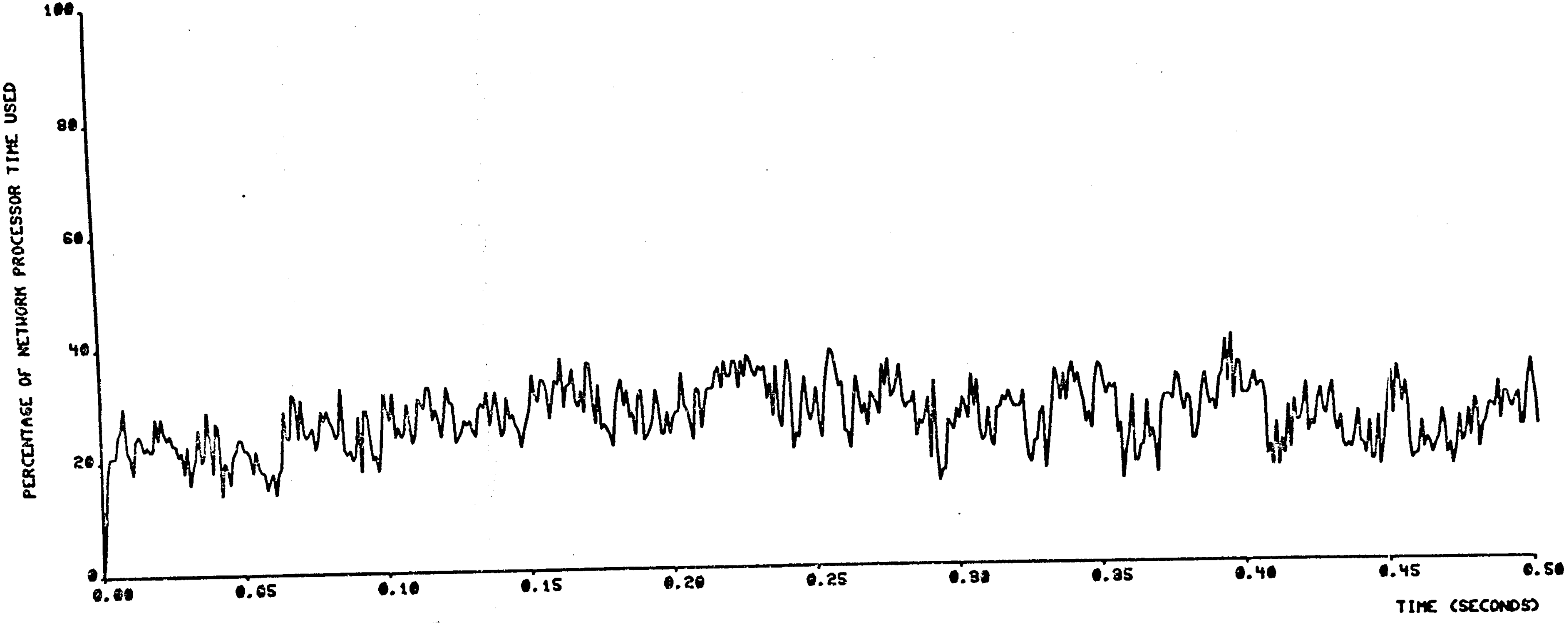


Figure 6.14 Standard Network - 2 nodes.
Percentage of network processor time used
per millisecond sample interval.

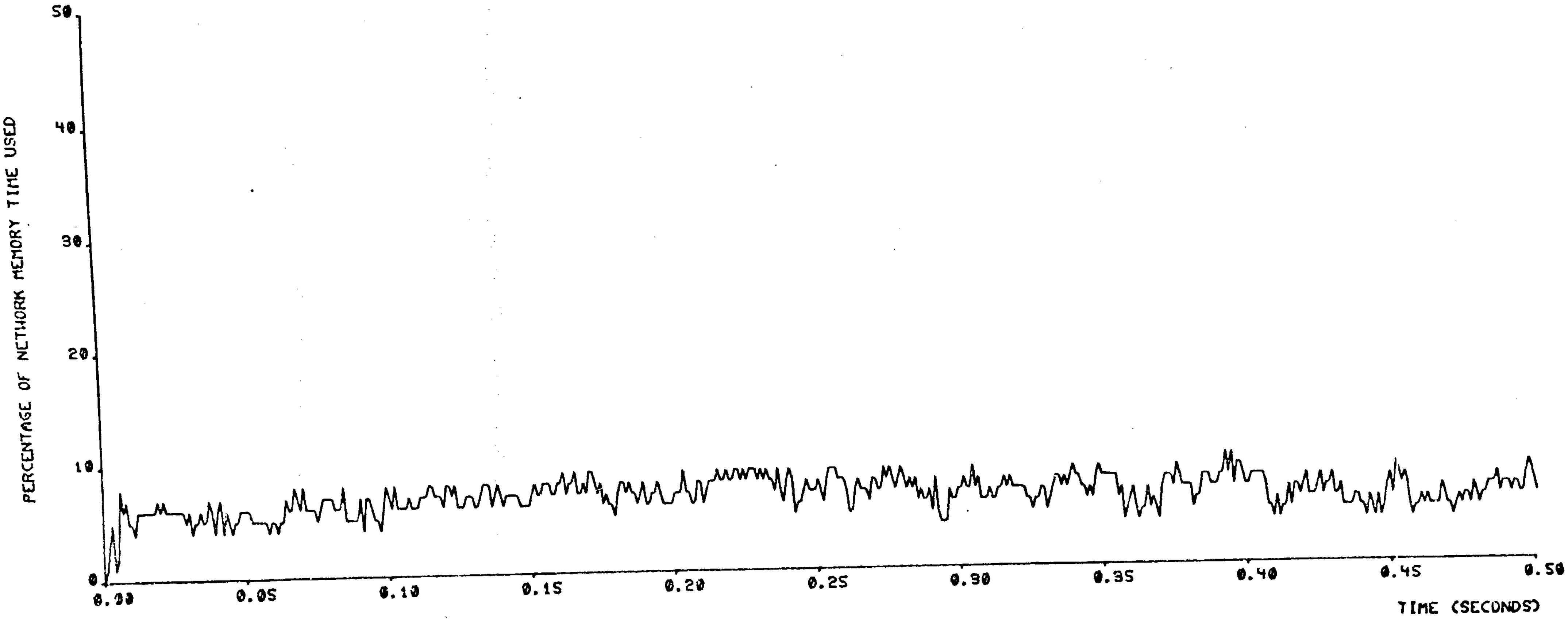


Figure 6.15 Standard Network - 2 nodes.
Percentage of network memory time used
per millisecond sample interval.

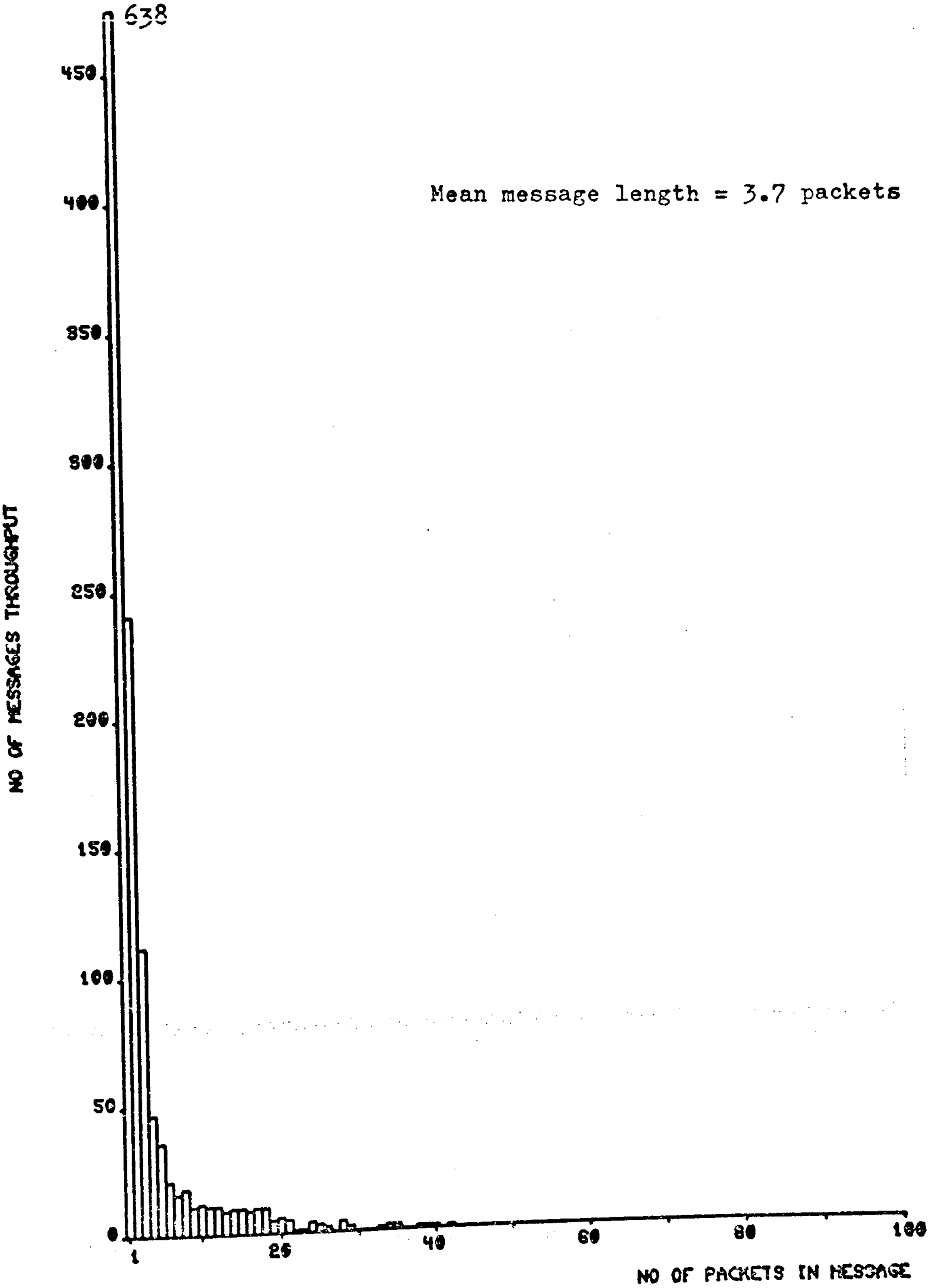


Figure 6.16 Standard Network - 2 nodes. Distribution of message lengths sent through network.

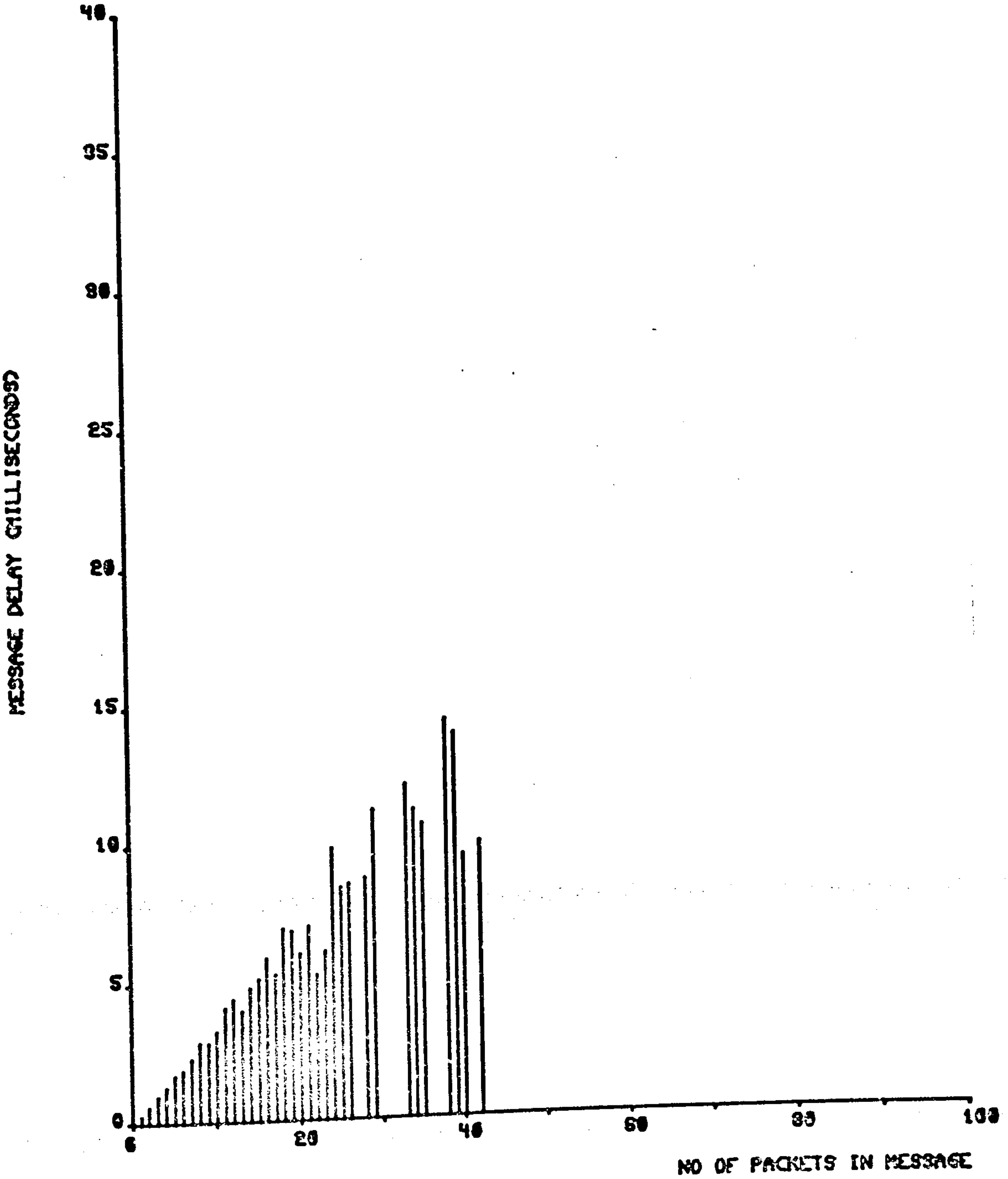


Figure 6.17 Standard Network - 2 nodes.
Effect of message length on message
throughput time.

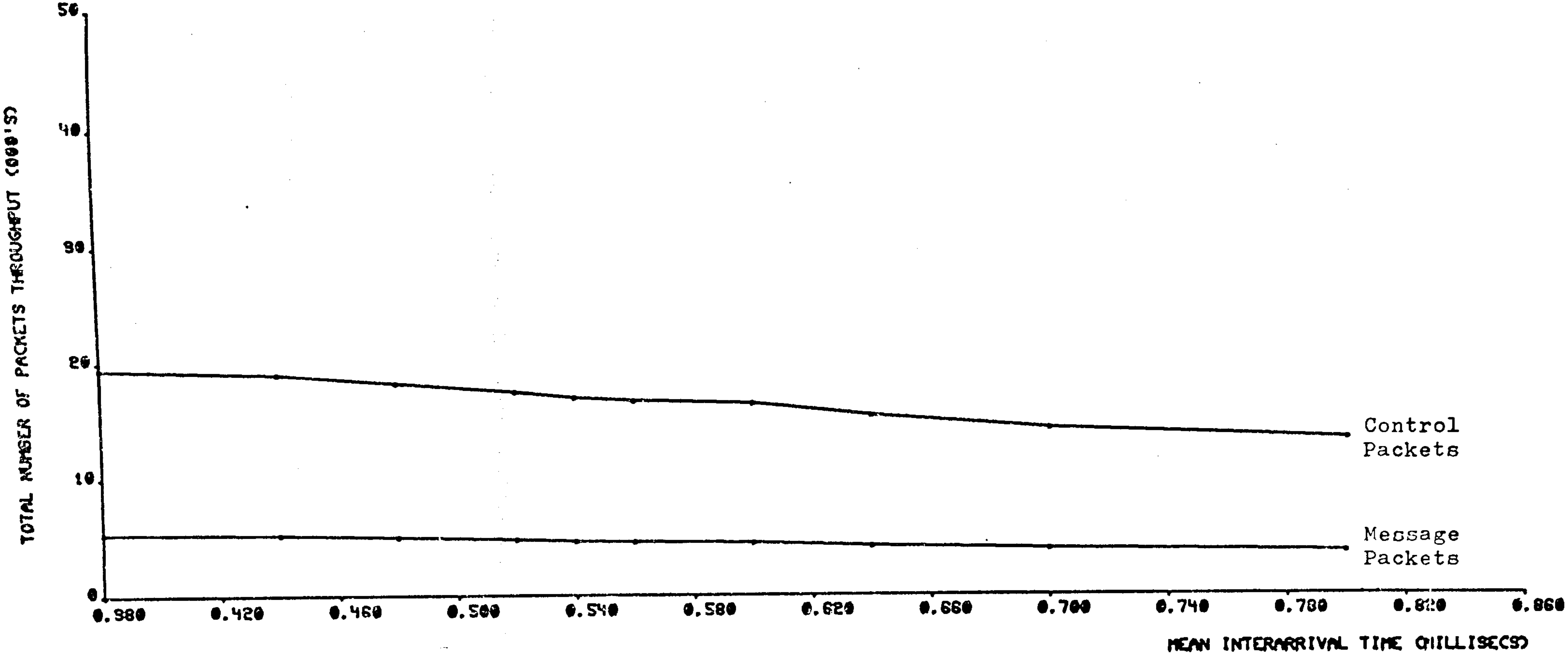


Figure 6.18 Standard Network - 2 nodes.
Effect of varying message mean
interarrival rate on network traffic.

the standard network had three nodes each supporting two hosts, the case now is of two nodes each supporting two hosts, resulting in four hosts per line as suggested before. The maximum number of packets that the line could support is 9.2×10^3 whereas 9.4×10^3 packets were transmitted on the network. This is accounted for by "incest" where a host may communicate with any other host. This results in communication with local hosts and so use of internodal lines is not made by these packets. In the two node network the line speed was the limiting factor.

6.4 Conclusions

The experiments carried out in this chapter have shown that if the user is to have a good quality of service, then the line speeds should operate at as high a speed as possible. User traffic that might be generated on the operational network is very difficult to predict, but it would seem that 100 kilobit lines would be unsatisfactory during periods where large files were being transmitted over the network. It has been shown that the throughput time of a message is directly proportional to the line speed.

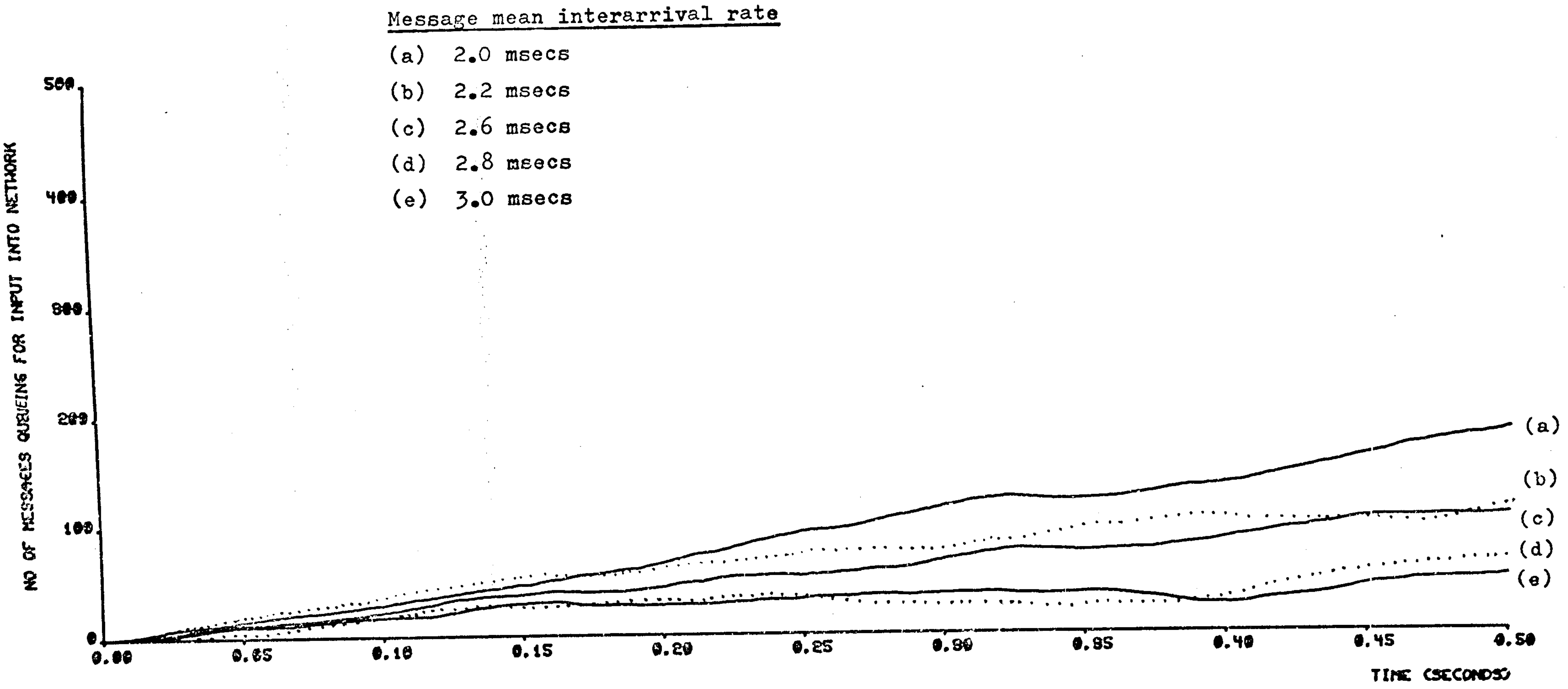


Figure 6.19 EWMA Standard Network - 1 megabit lines. Effect of varying message mean interarrival rate on message queue for input into network.

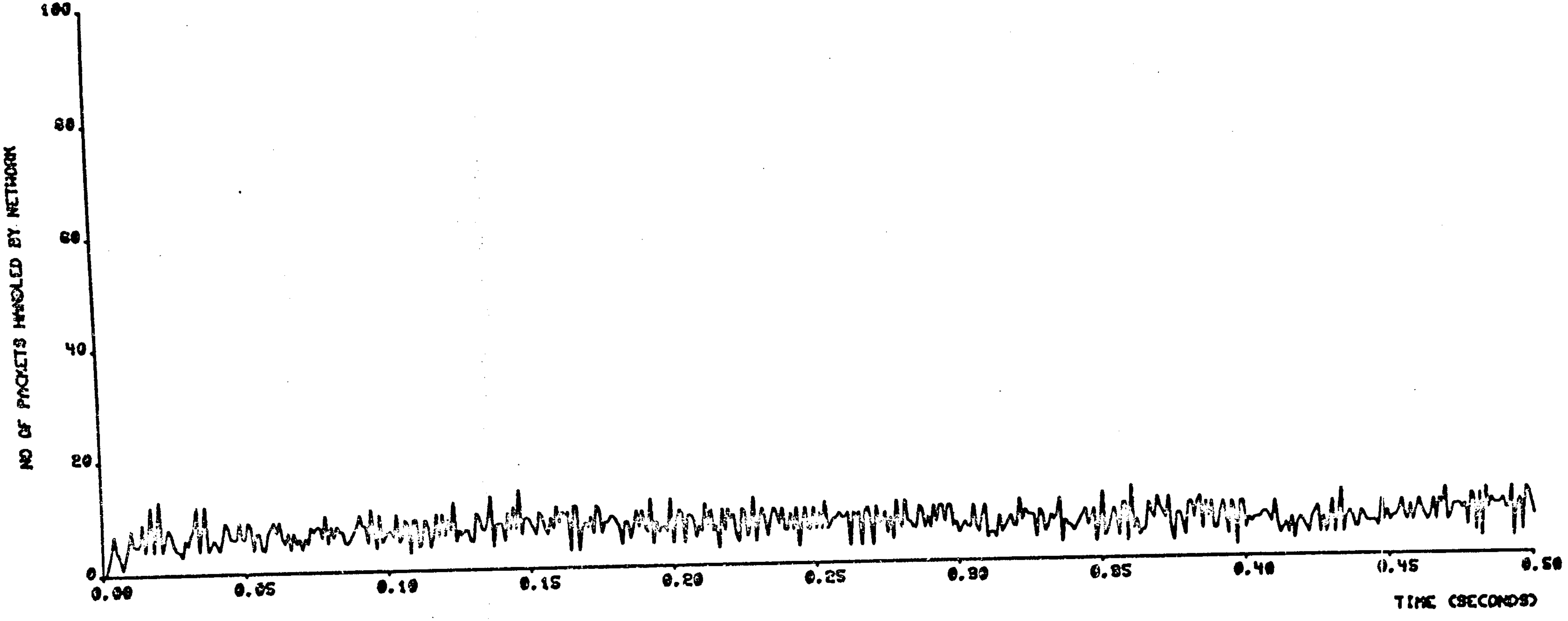


Figure 6.20 Standard Network - 1 megabit lines.
Number of packets processed by network
per millisecond sample interval.

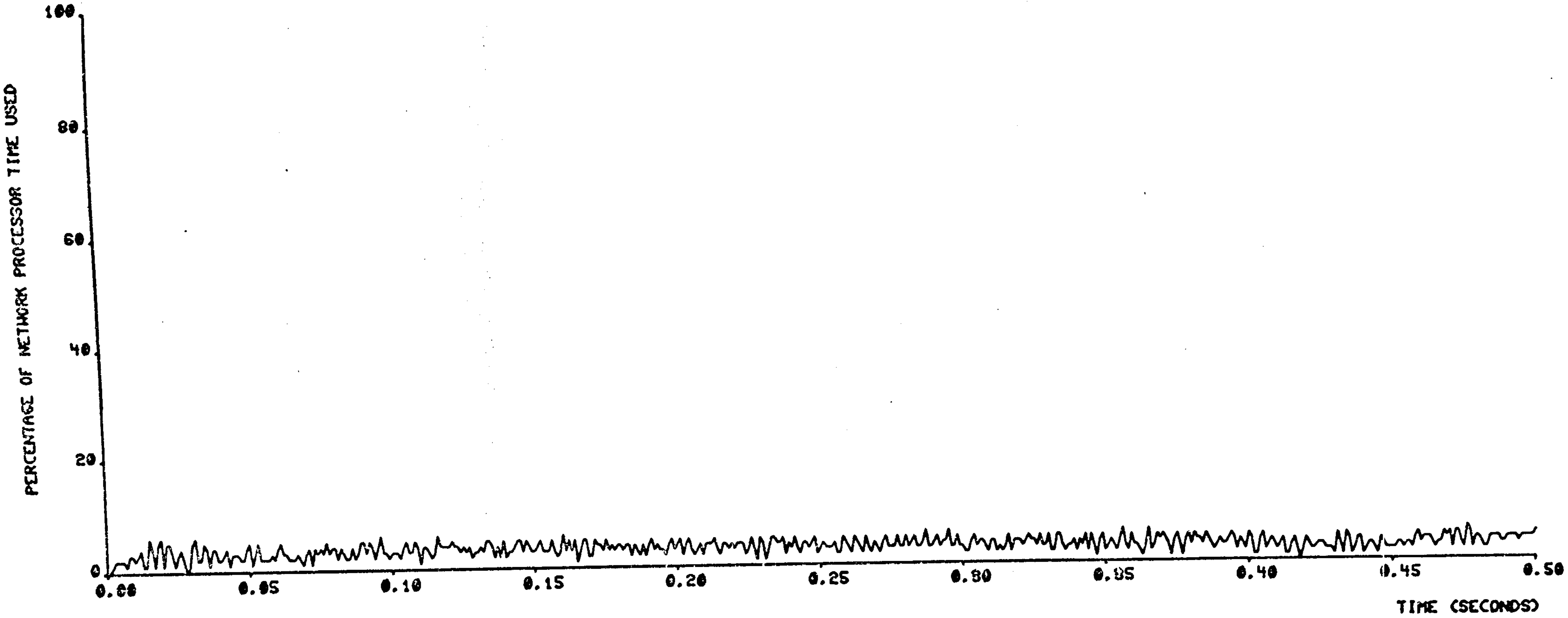


Figure 6.21 Standard Network - 1 megabit lines.
Percentage of network processor time used
per millisecond sample interval.

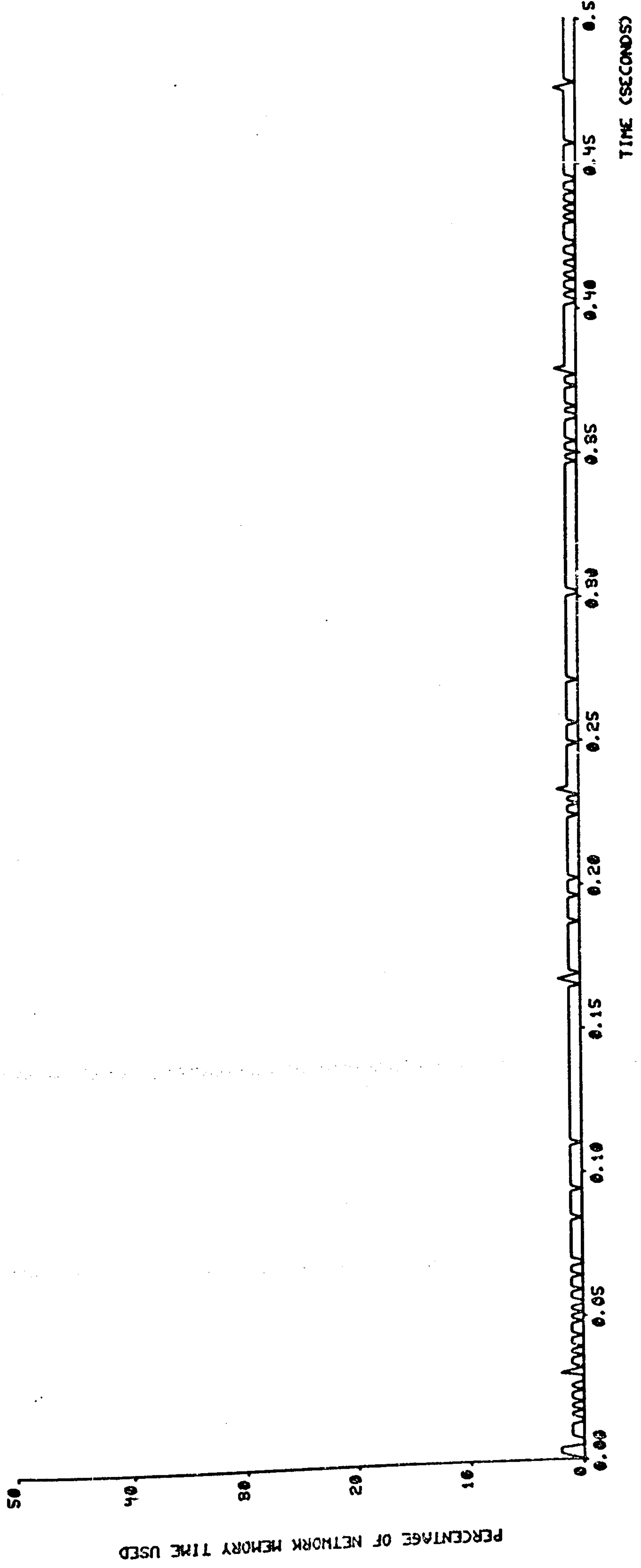


Figure 6.22 Standard Network - 1 megabit lines. Percentage of network memory time used per millisecond sample interval.

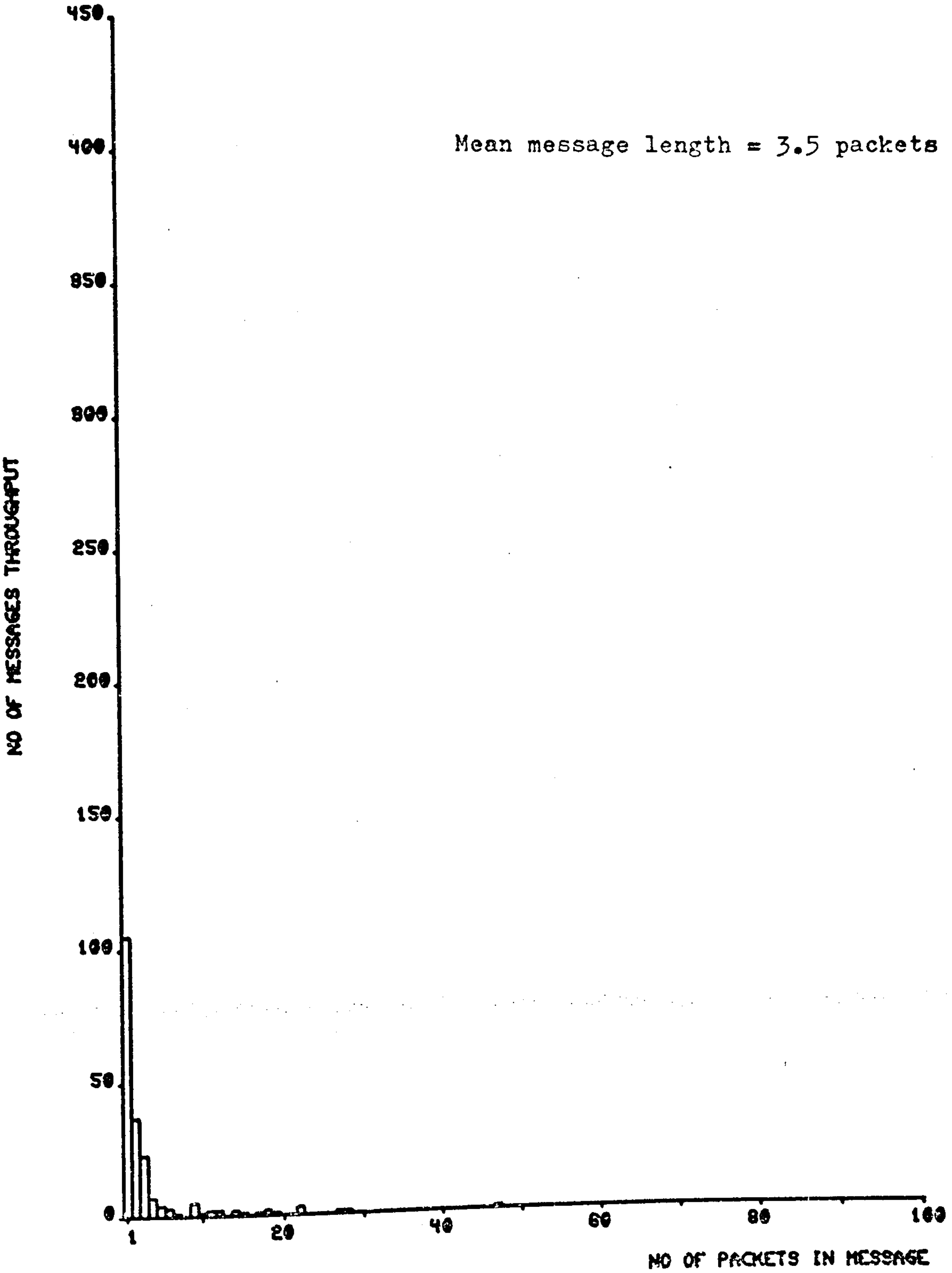


Figure 6.23 Standard Network - 1 megabit lines. Distribution of message lengths sent through network.

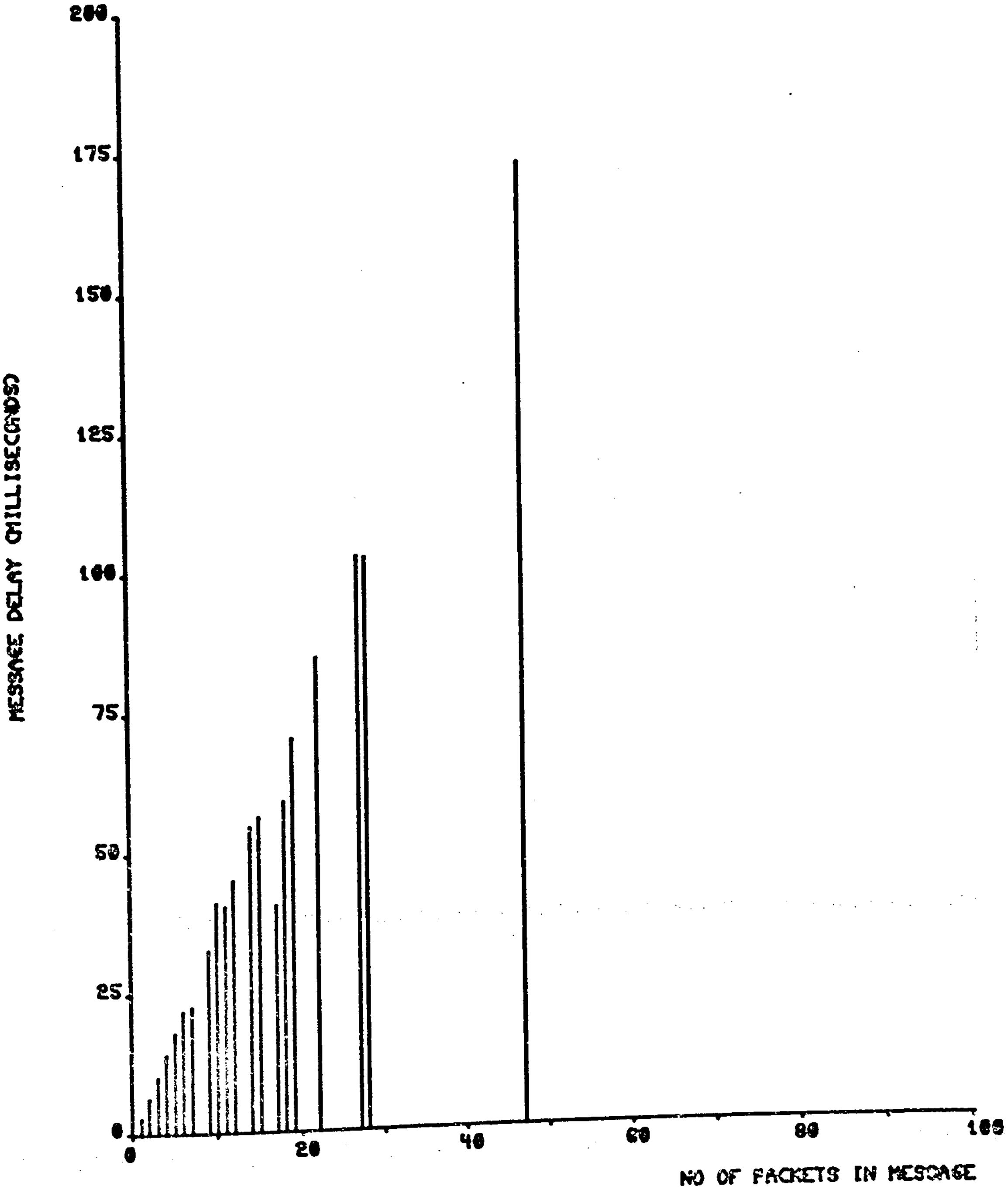


Figure 6.24 Standard Network - 1 megabit lines. Effect of message length on message throughput time.

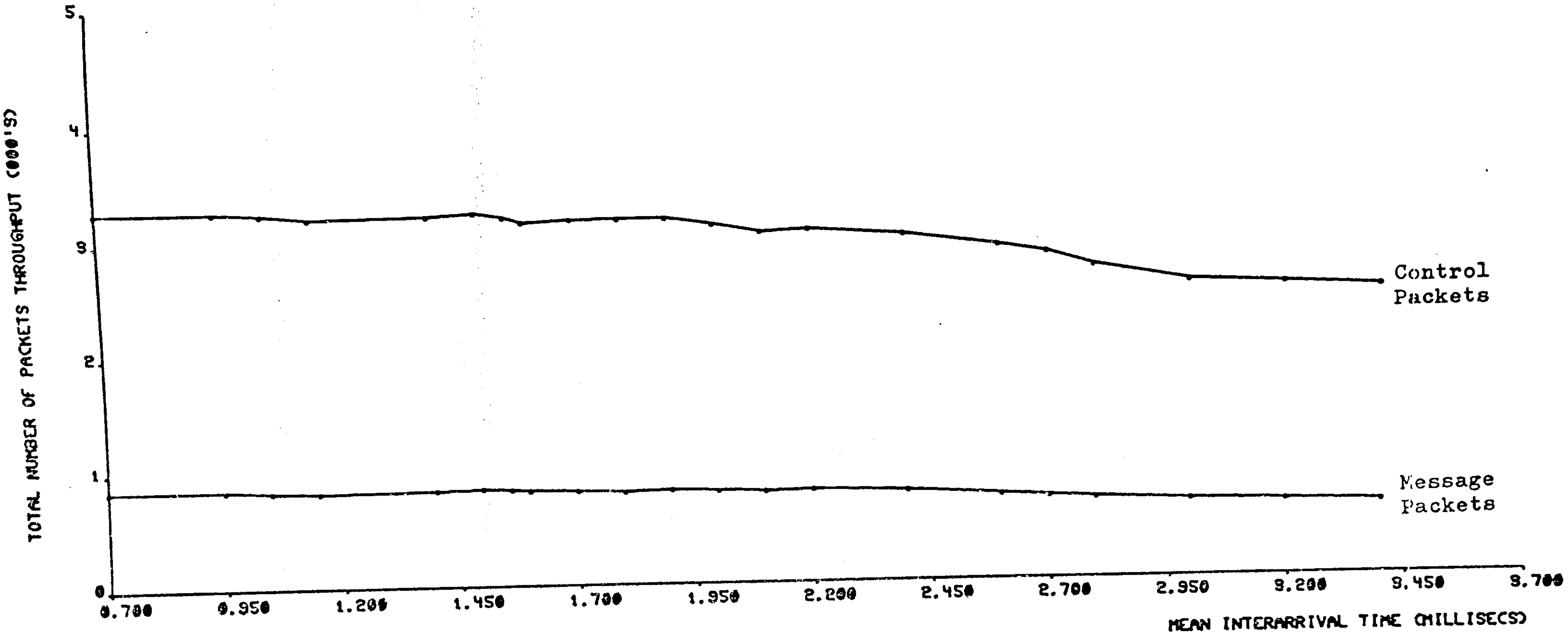


Figure 6.25 Standard Network - 1 megabit lines.
Effect of varying message mean
interarrival rate on network traffic.

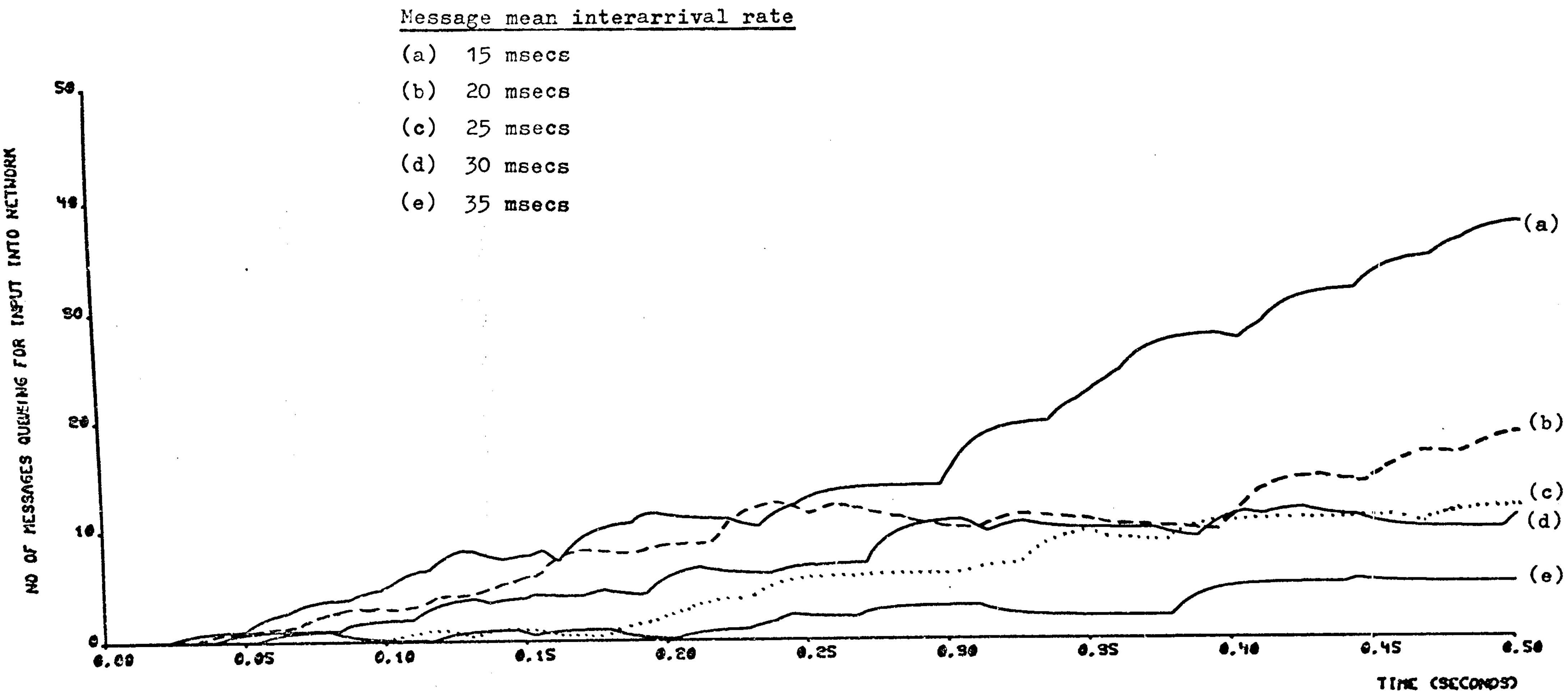


Figure 6.26 EWMA Standard Network - 100k bit lines.
Effect of varying message mean interarrival rate on message queue for input into network.

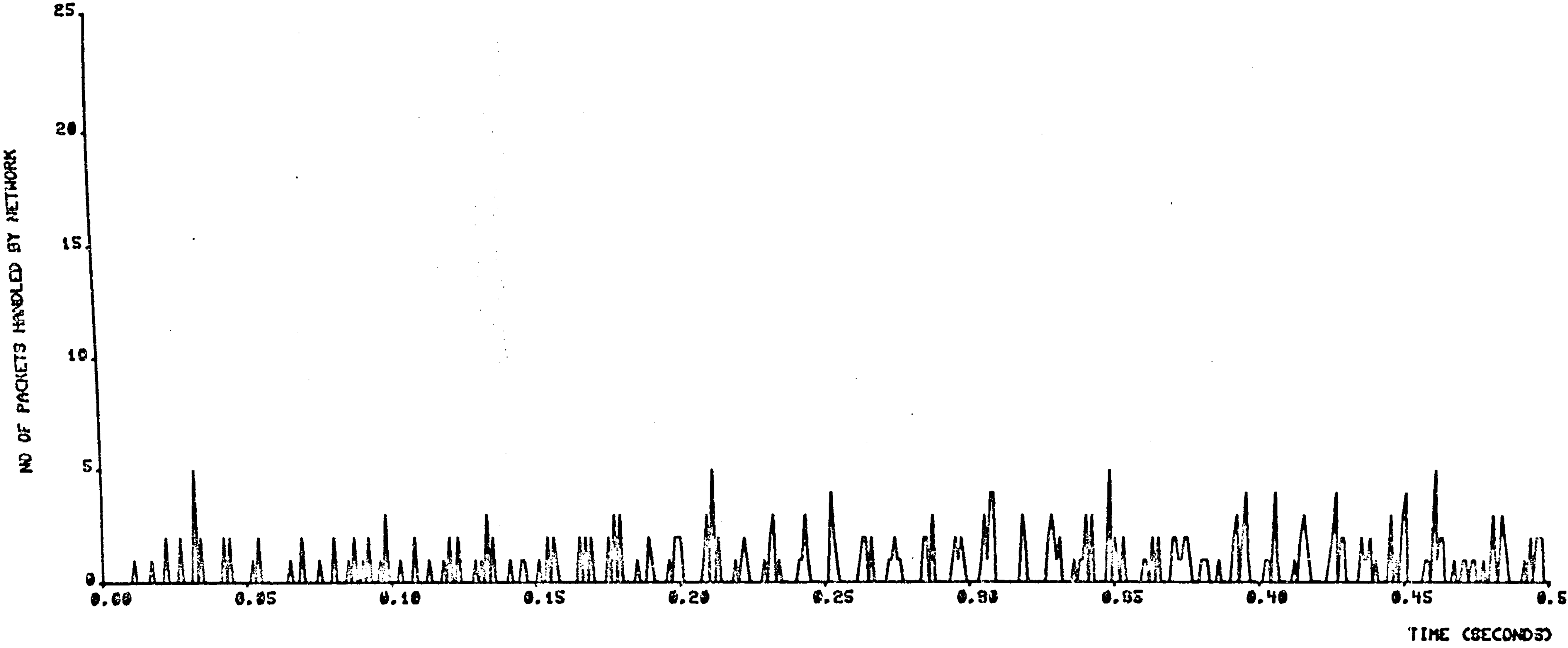


Figure 6.27 Standard Network - 100k bit lines.
Number of packets processed by network
per millisecond sample interval.

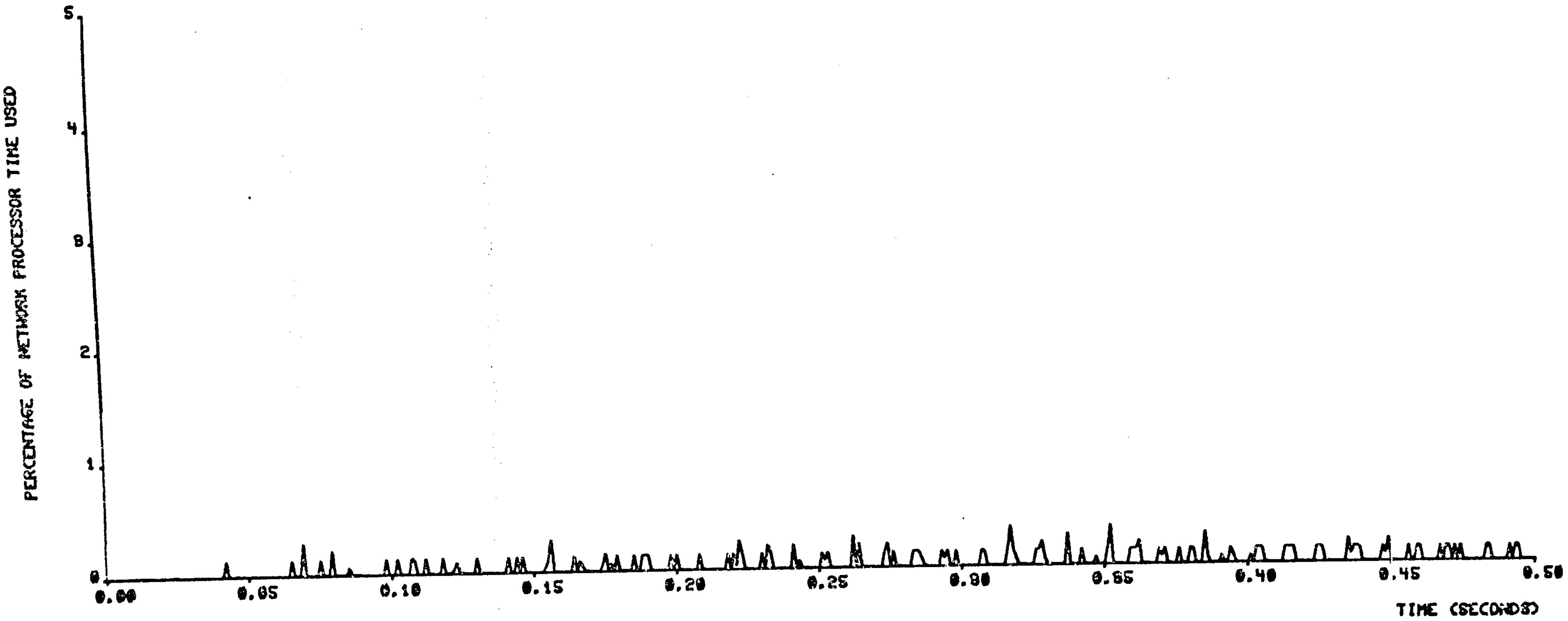


Figure 6.28 Standard Network - 100k bit lines.
Percentage of network processor time used
per millisecond sample interval.

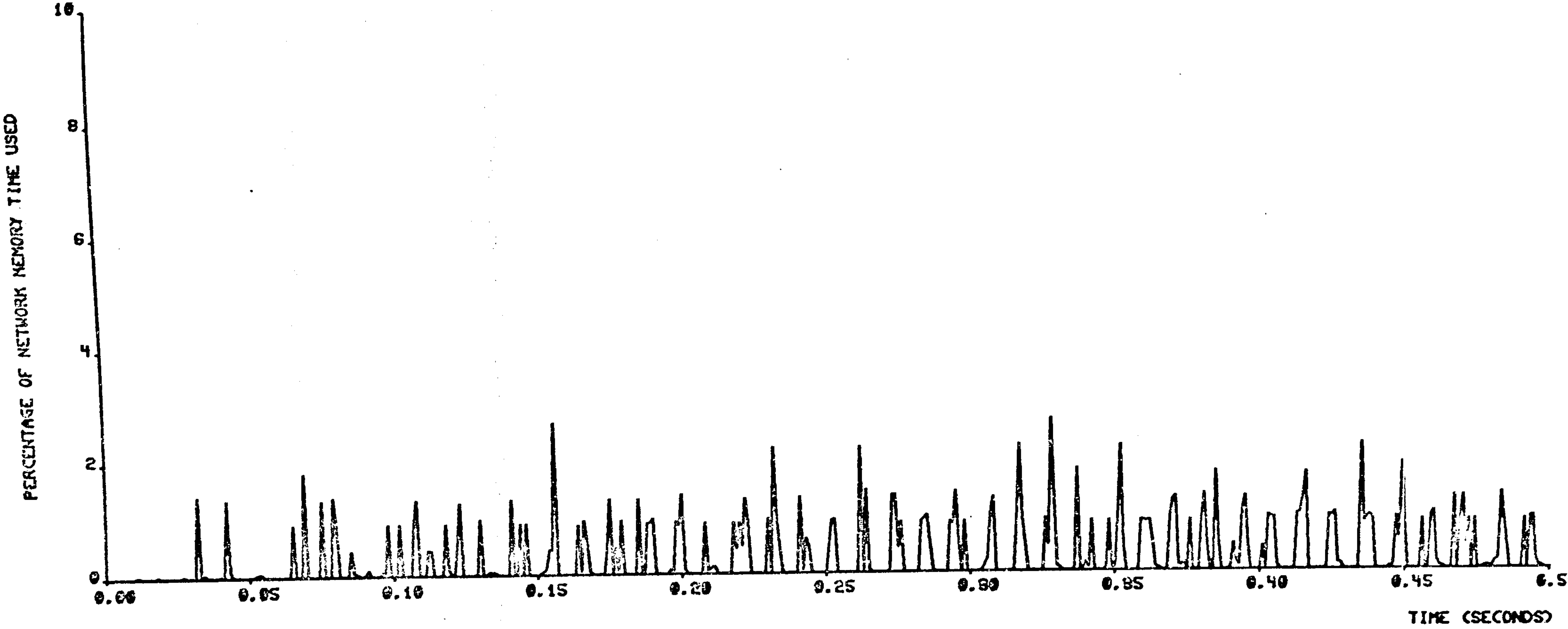


Figure 6.29 Standard Network - 100k bit lines. Percentage of network memory time used per millisecond sample interval.

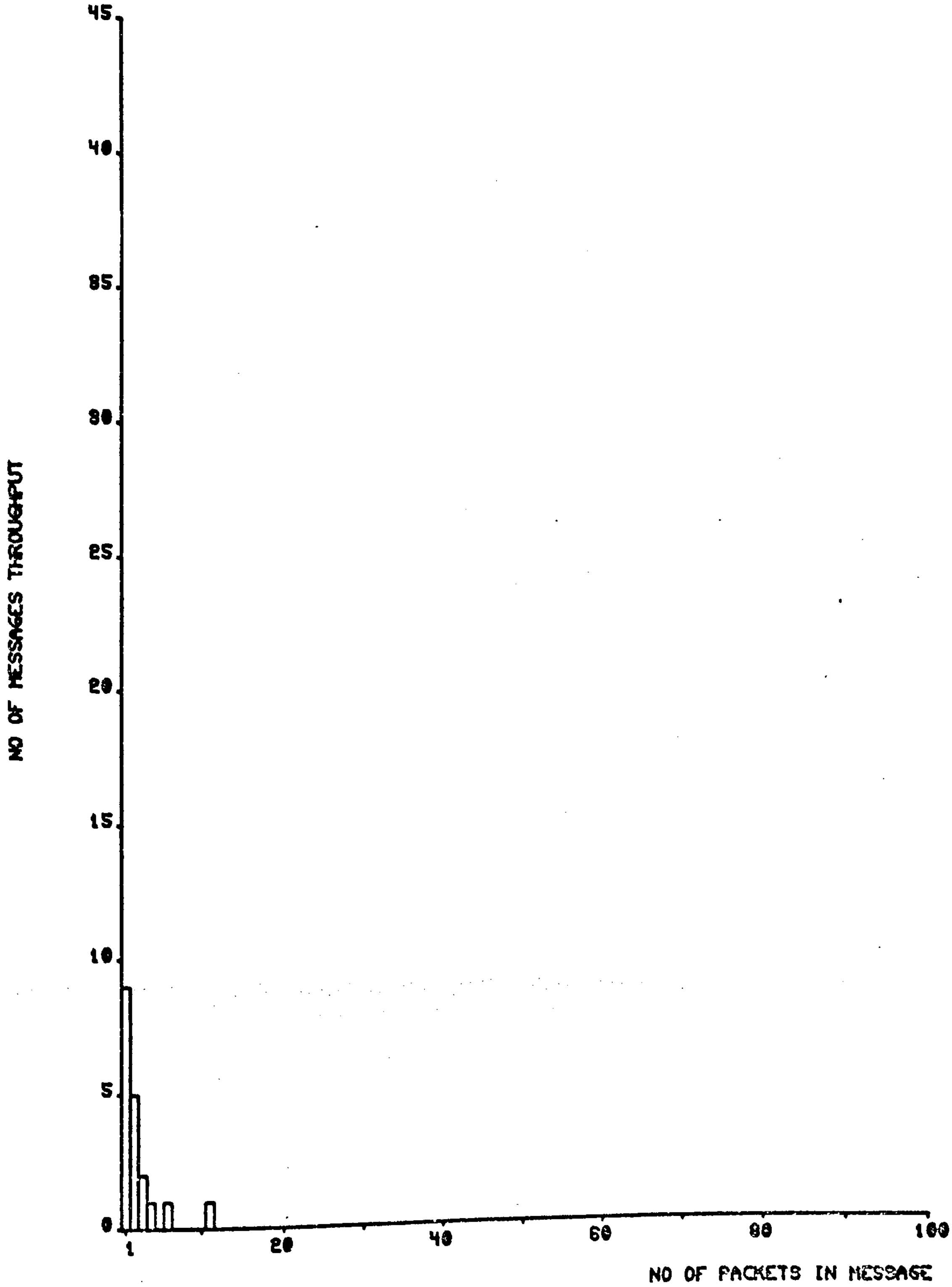


Figure 6.30 Standard Network - 100k bit lines. Distribution of message length sent through network.

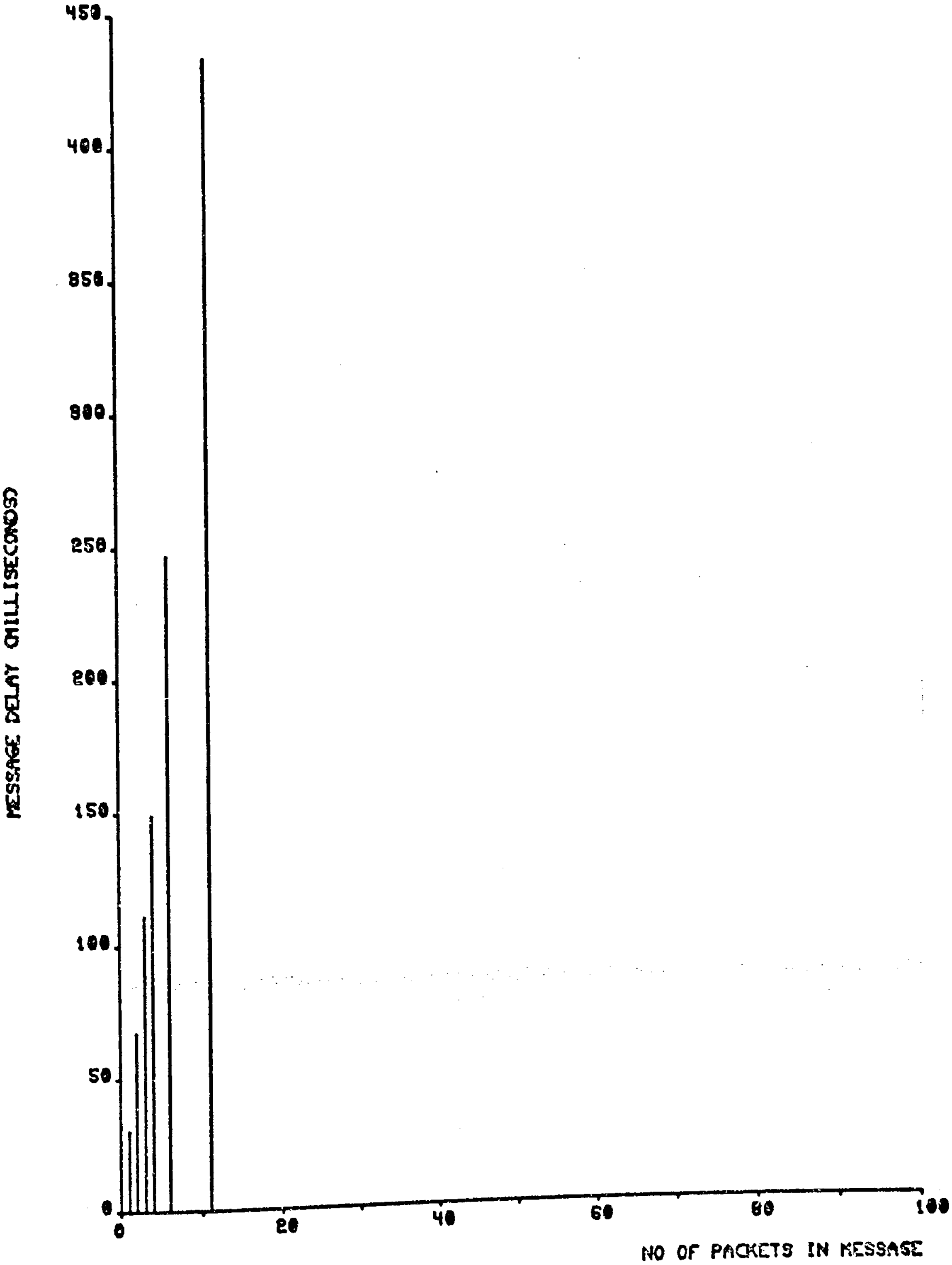


Figure 6.31 Standard Network - 100k bit lines.
Effect of message length on message
throughput time.

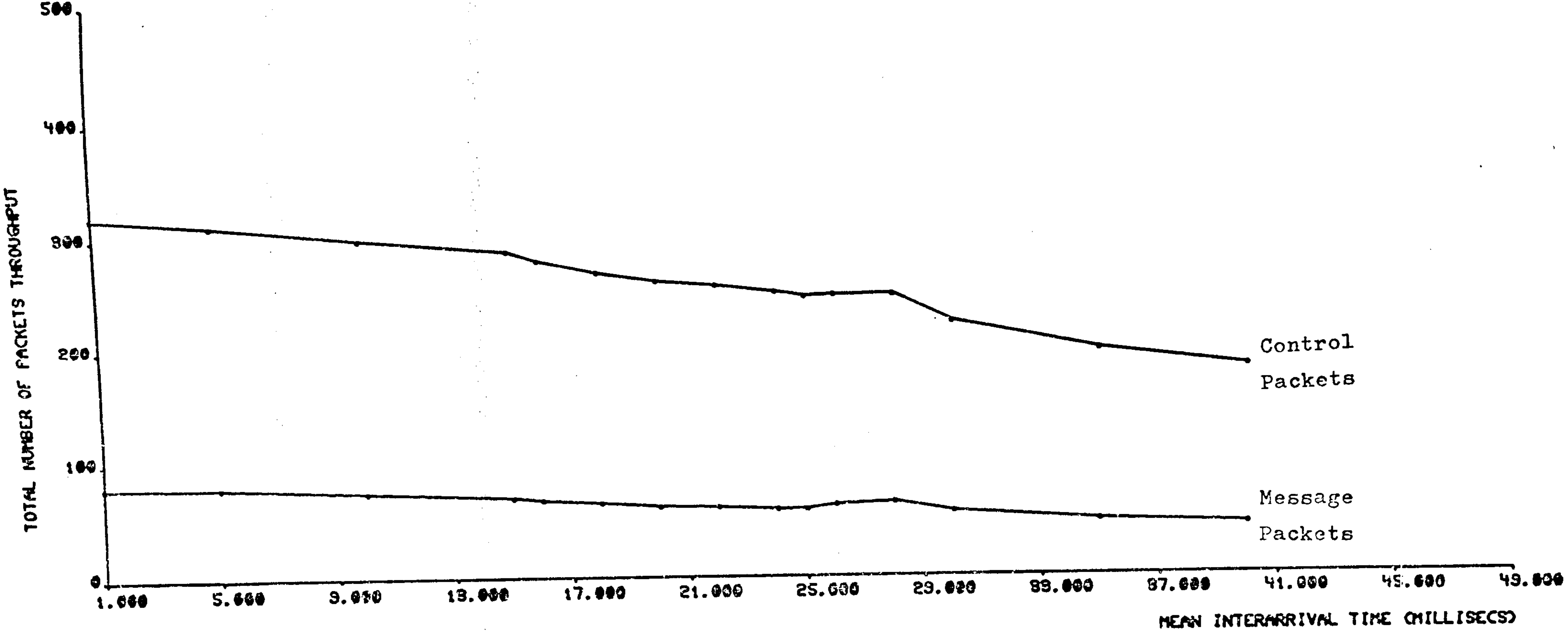


Figure 6.32 Standard Network - 100k bit lines.
Effect of varying message mean
interarrival rate on network traffic.

CHAPTER 7

LOW LEVEL NETWORK PARAMETERS

7.1 Introduction

The purpose of this chapter is to evaluate the effects of the low level network on the performance of the network. Four sets of results are considered:

1) Average message length (figures 7.1 to 7.7)
2) Message mix ratio (figures 7.8 to 7.14)
3) Mean number of generating hosts (figures 7.15 to 7.21)
4) Packet length (figures 7.22 to 7.28)

Several formulae are derived including an approximation to the mean message delay.

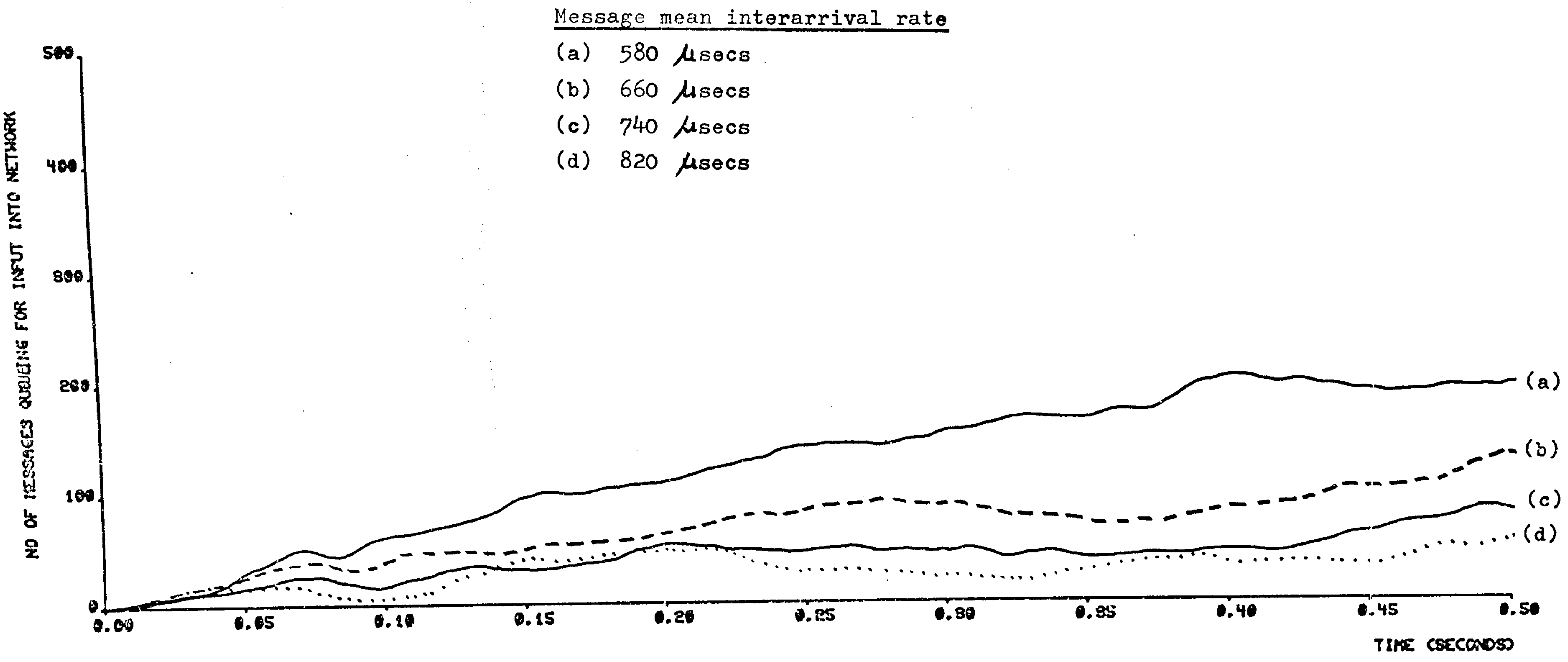


Figure 7.1 EWMA Standard Network - 256 word packet. Effect of varying message mean interarrival rate on message queue for input into network.

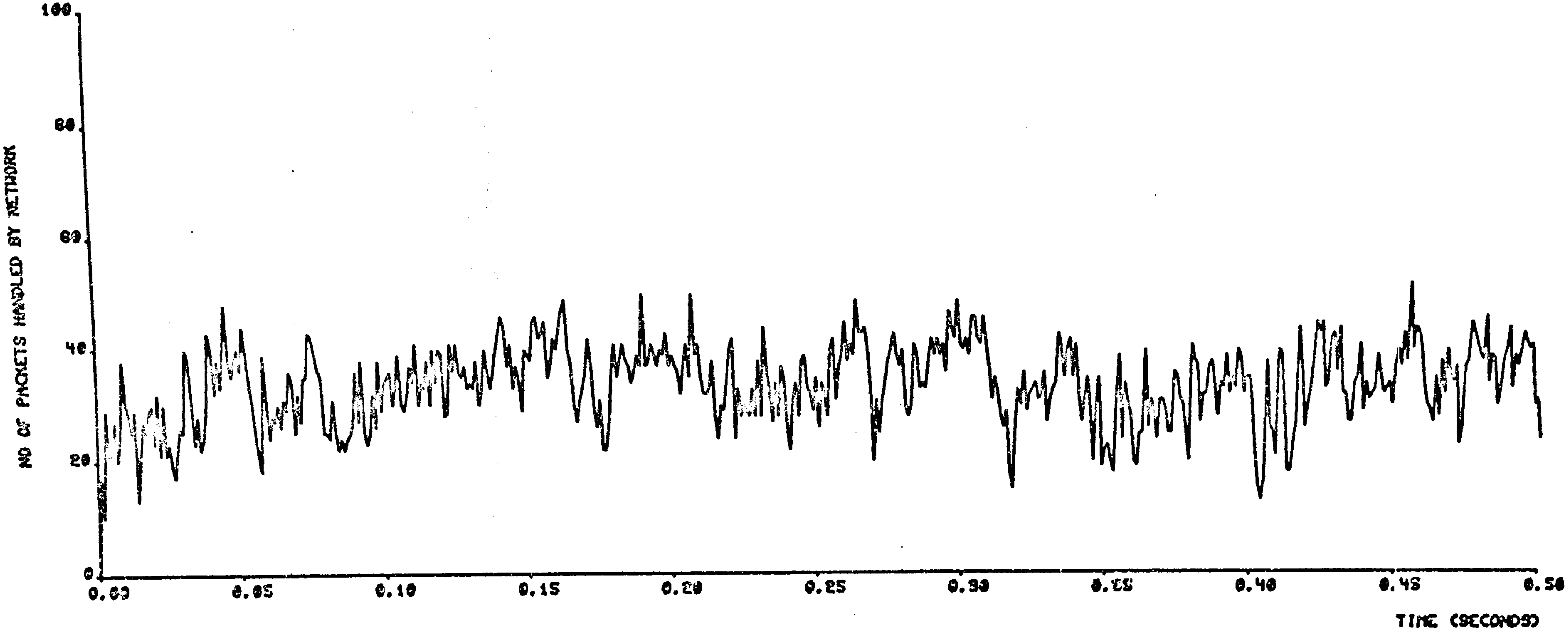


Figure 7.2 Standard Network - 256 word packet. Number of packets processed by network per millisecond sample interval.

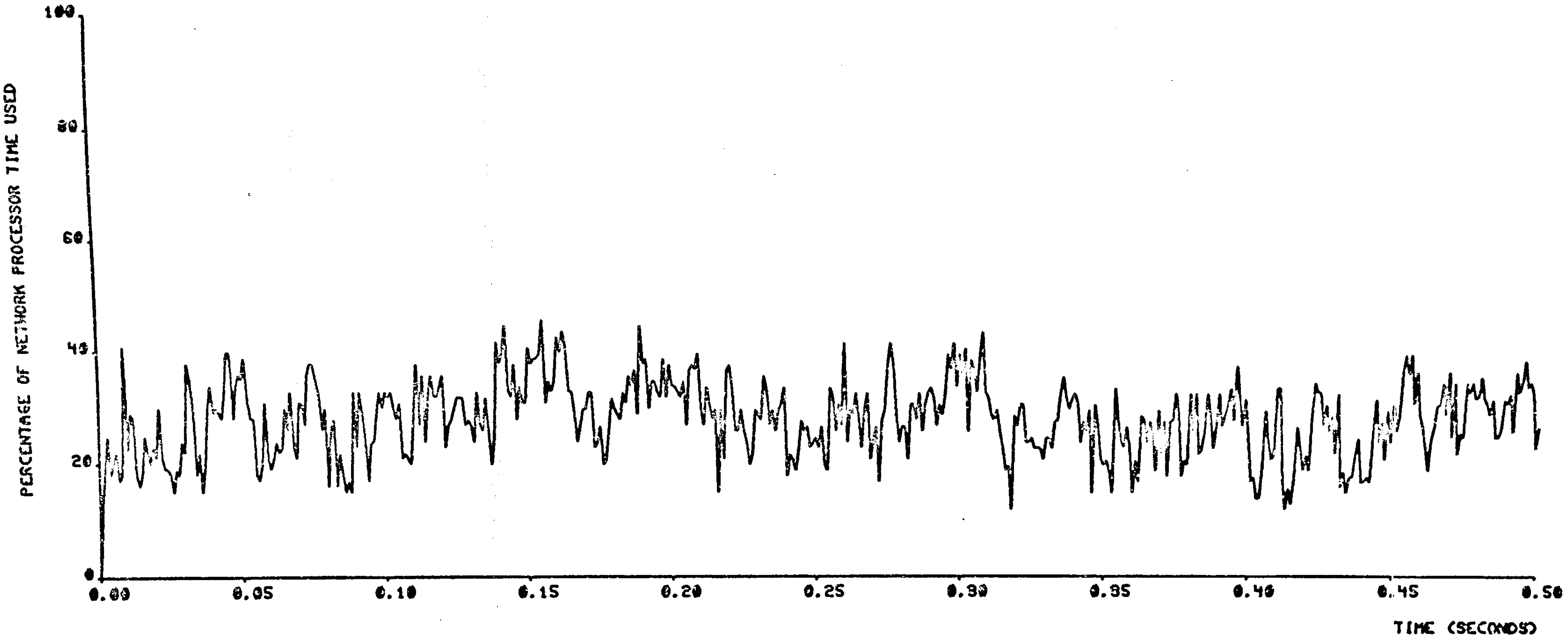
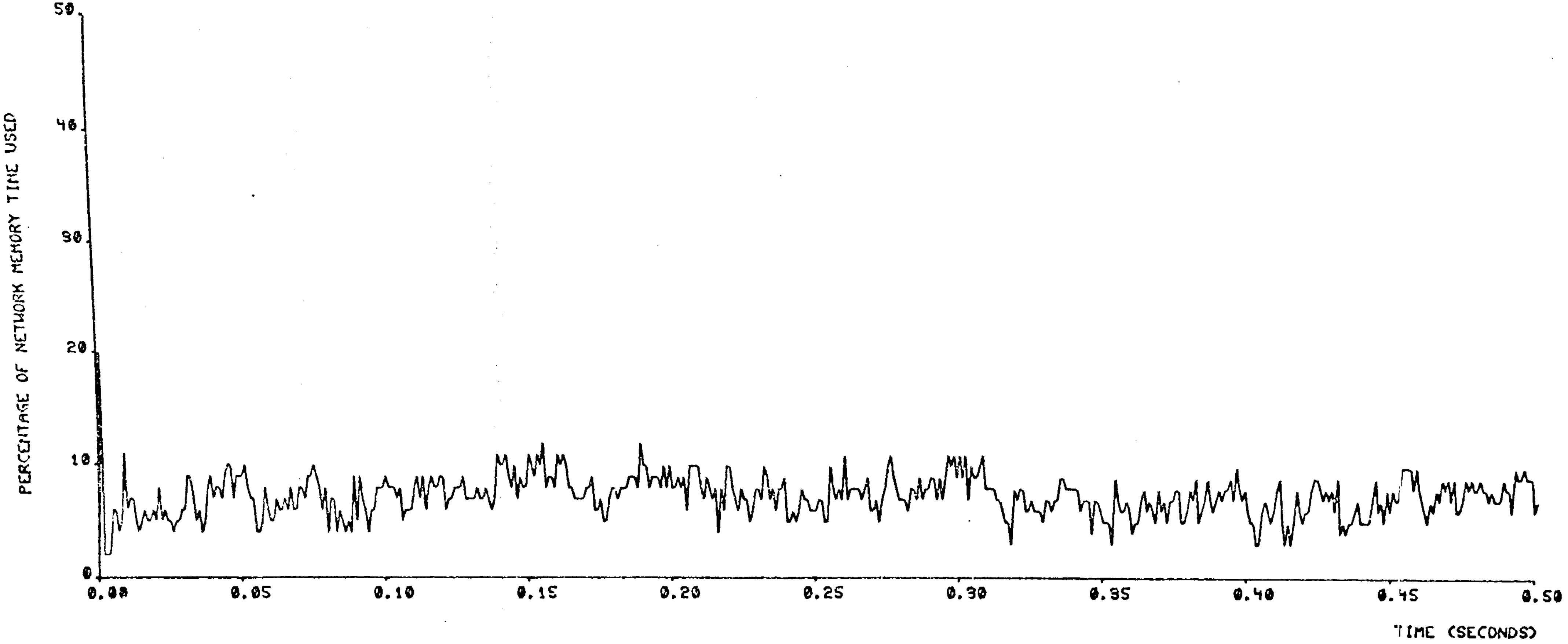


Figure 7.3 Standard Network - 256 word packet. Percentage of network processor time used per millisecond sample interval.



<u>Figure 7.4</u> Standard Network - 256 word packet.
Percentage of network memory time used
per millisecond sample interval.

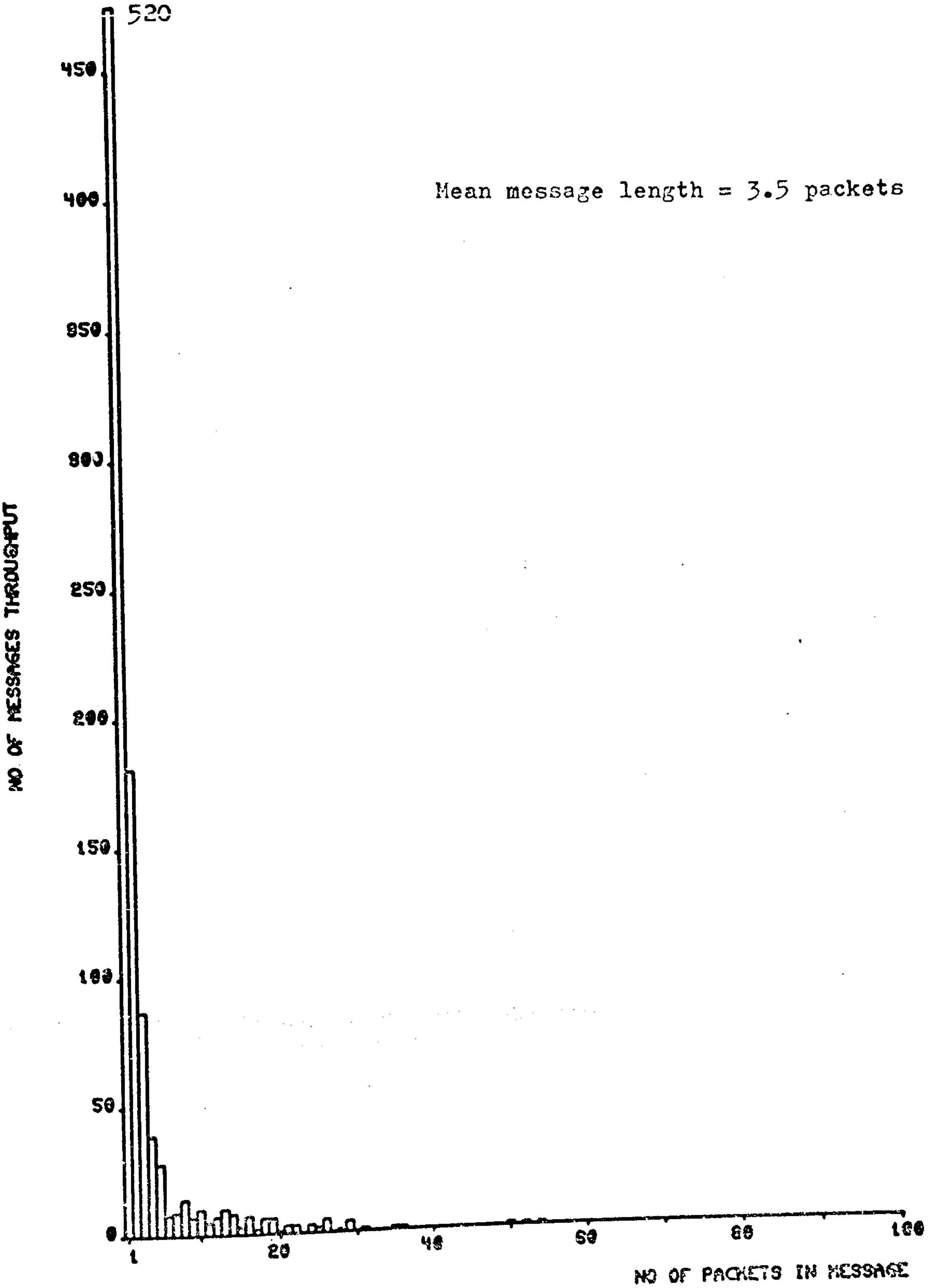


Figure 7.5 Standard Network - 256 word packet. Distribution of message length sent through network.

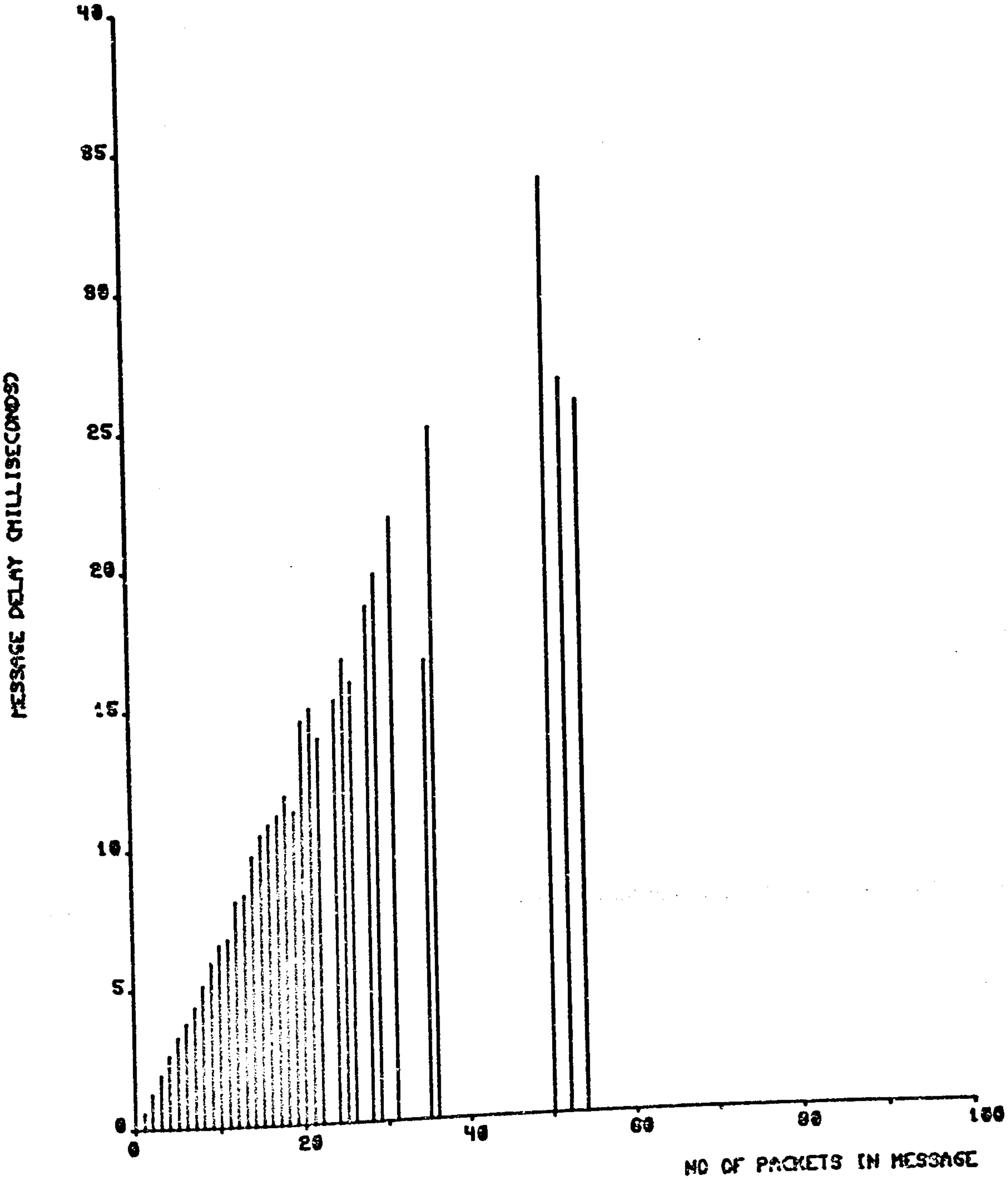


Figure 7.6 Standard Network - 256 word packet. Effect of message length on message throughput time.

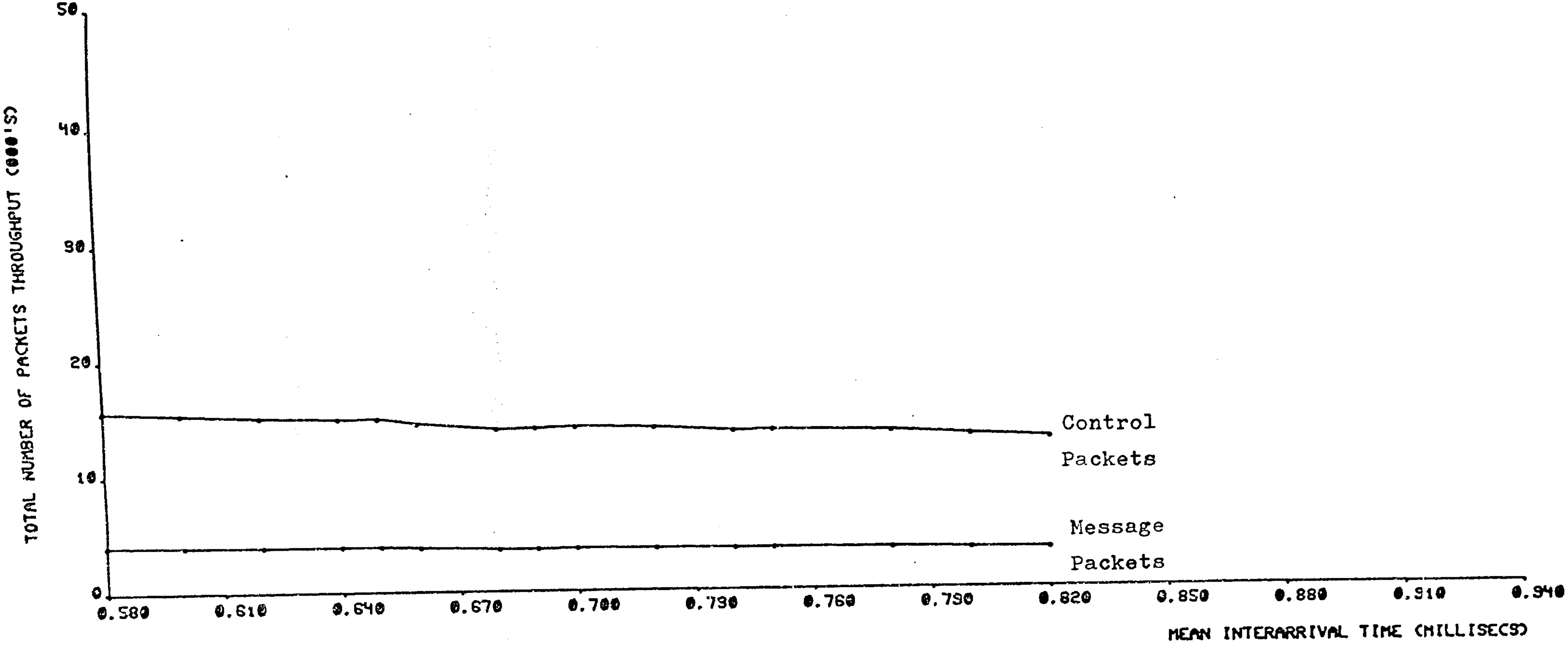


Figure 7.7 Standard Network - 256 word packet. Effect of varying message mean interarrival rate on network traffic.

7.2 Packet length

Packet switching communication systems have two fundemental goals in processing data: low delay and high throughput. For low delay a short packet is needed while for maximum throughput it is necessary to have a large packet size to minimise system overhead. With a short packet there is a higher probability of error free transmission.

Figures 7.22 to 7.28 show the effects of doubling the standard network packet length from 128 words to 256 words. Using the formula for system full capacity:

$$\frac{\text{line speed X nc of nodes}}{\text{(message packet length + control packet length)}}$$

where the packet lengths are given in bits, for the standard network under the minimum time delay condition of an output buffer being available when required, the number of packets handled by the standard network is given by:

$$\frac{10000000 \text{ X } 3}{(128 + 8) \text{ X } 8} = 27.6 \text{ X } 10^3 \text{ packets/second.}$$

For the standard network with a 256 words packet, the number of packets handled by the network is given by:

$$\frac{10000000 \text{ X } 3}{(256 + 8) \text{ X } 8} = 14.2 \text{ X } 10^3 \text{ packets/second.}$$

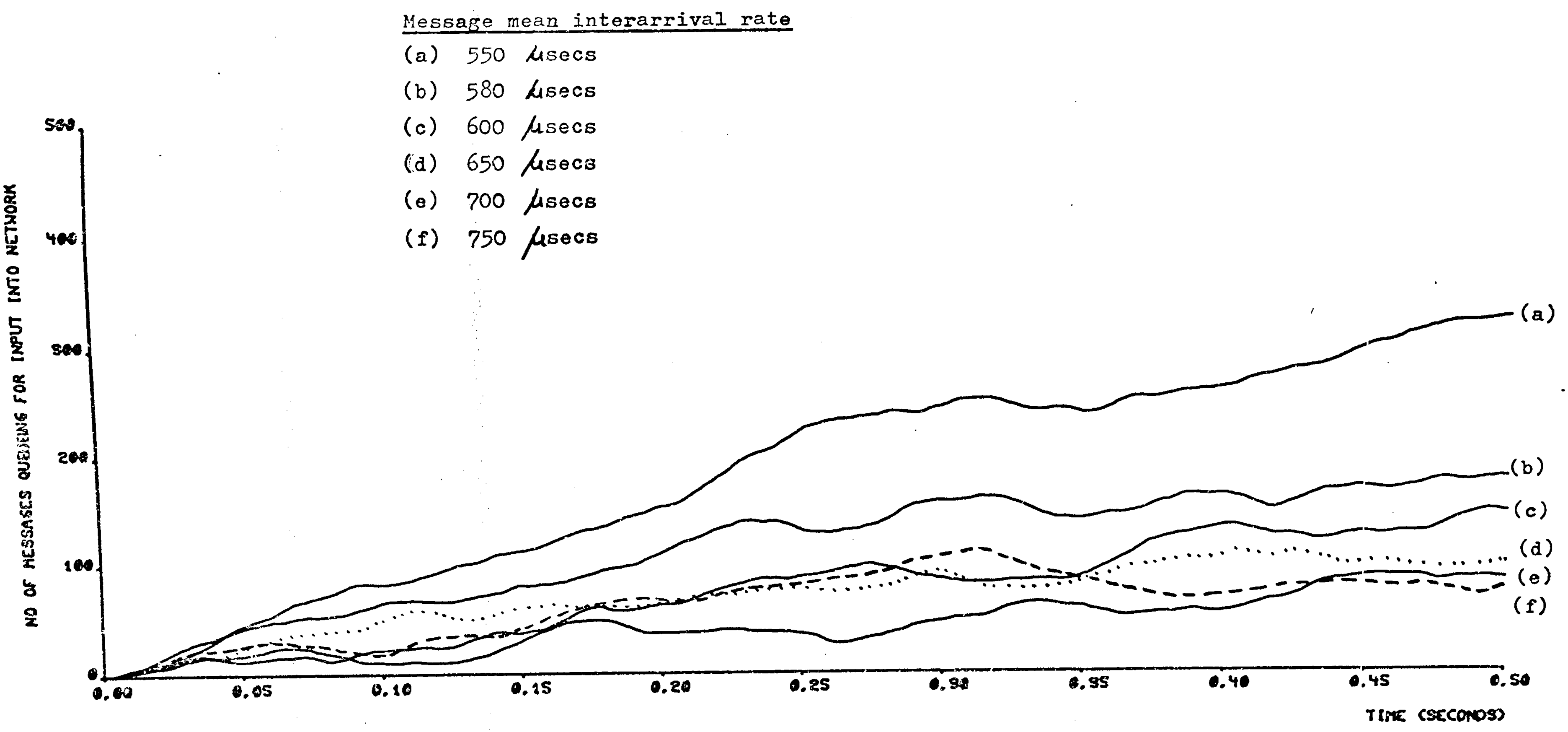


Figure 7.8 EWMA Standard Network - mean short message = 2 packets
- mean long message = 20 packets.

Effect of varying message mean interarrival rate on message queue for input into network.

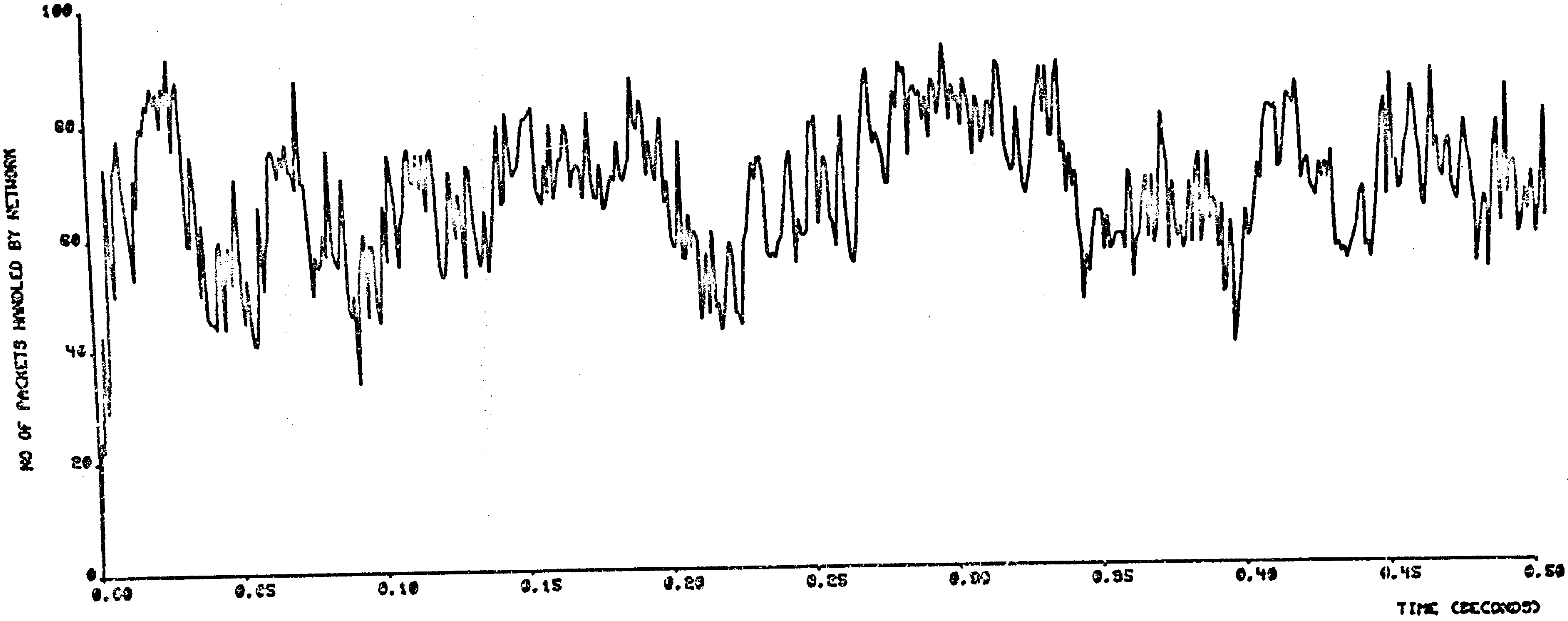


Figure 7.9 Standard Network - mean short message = 2 packets
- mean long message = 20 packets.

Number of packets processed by network
per millisecond sample interval.

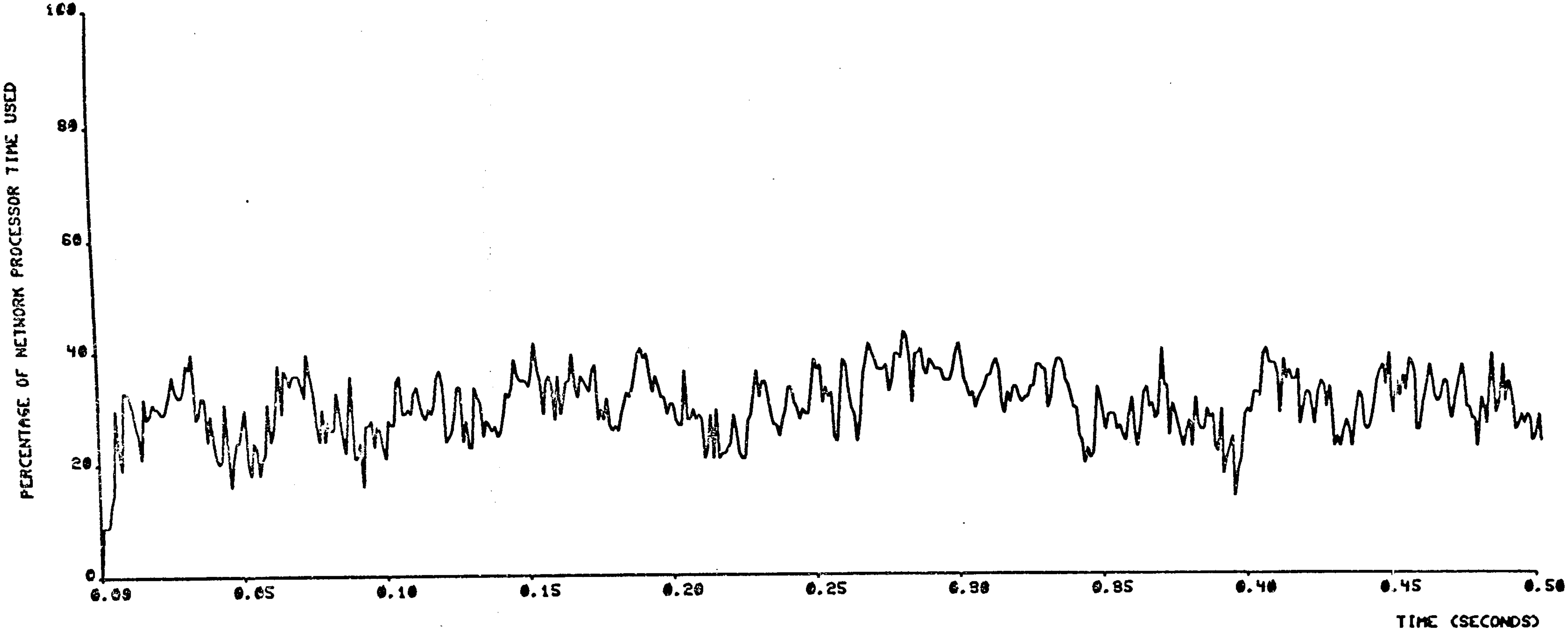


Figure 7.10 Standard Network - mean short message = 2 packets
- mean long message = 20 packets.
Percentage of network processor time used
per millisecond sample interval.

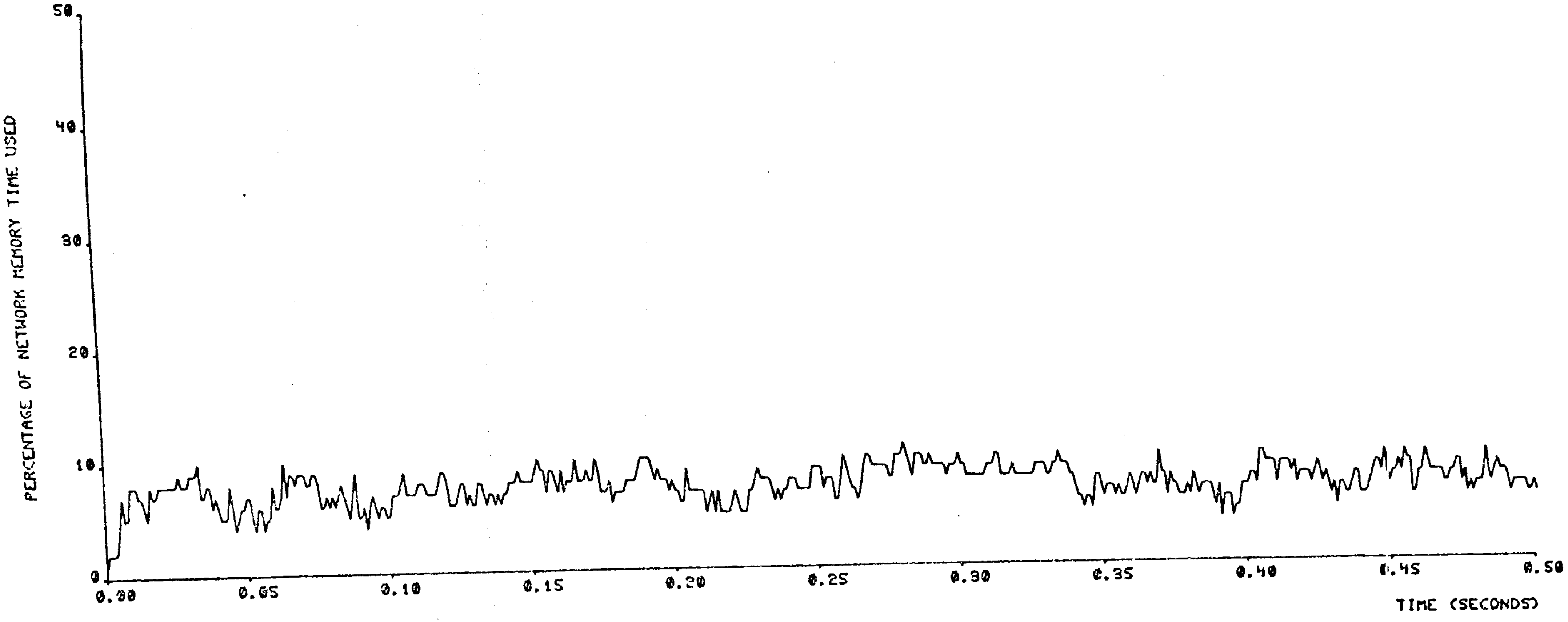


Figure 7.11 Standard Network - mean short message = 2 packets
- mean long message = 20 packets.
Percentage of network memory time used
per millisecond sample interval.

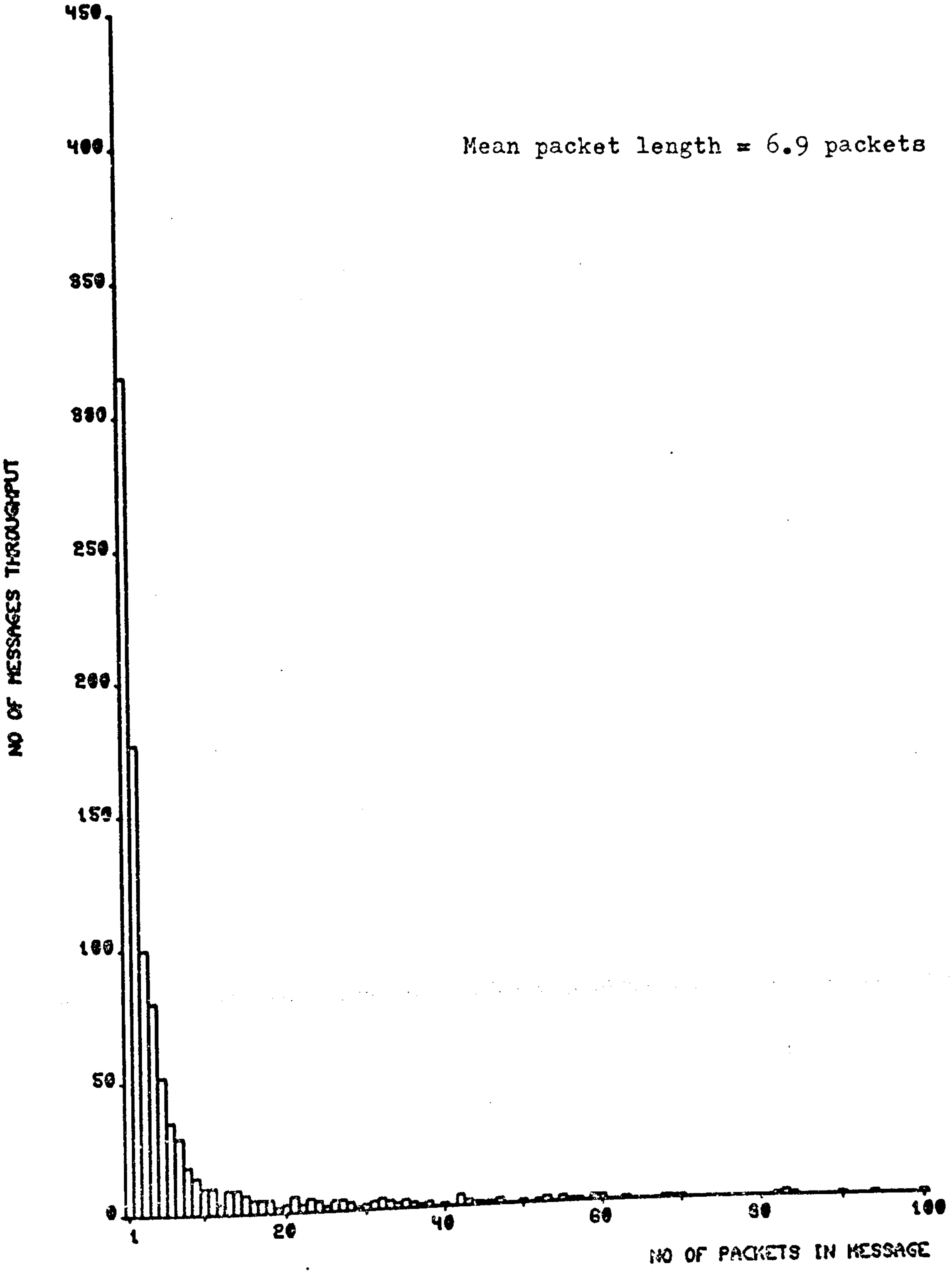


Figure 7.12 Standard Network - mean short message = 2 packets
- mean long message = 20 packets.

Distribution of message lengths sent through network.

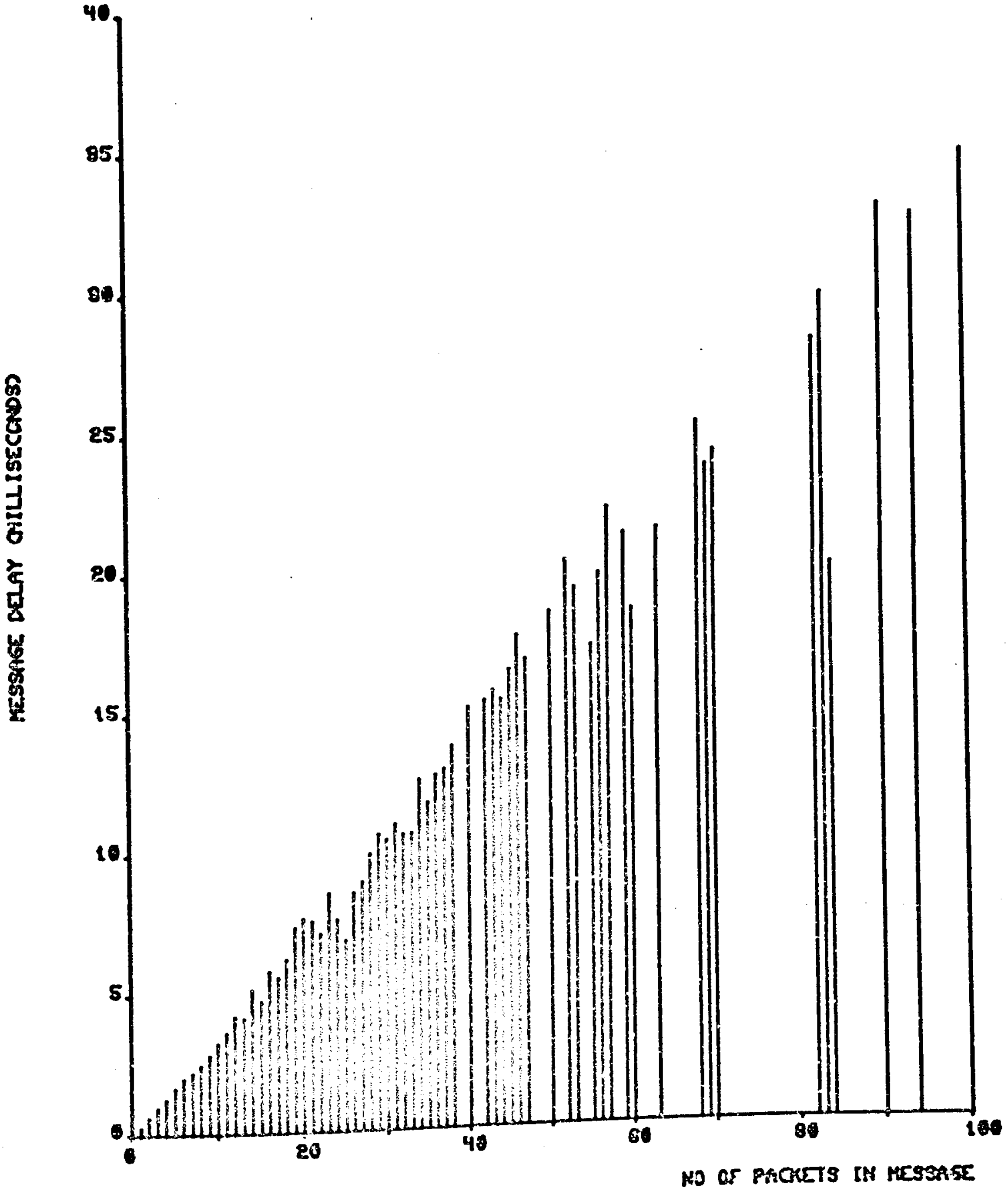


Figure 7.13 Standard Network - mean short message = 2 packets
- mean long message = 20 packets.

Effect of message length on message throughput time.

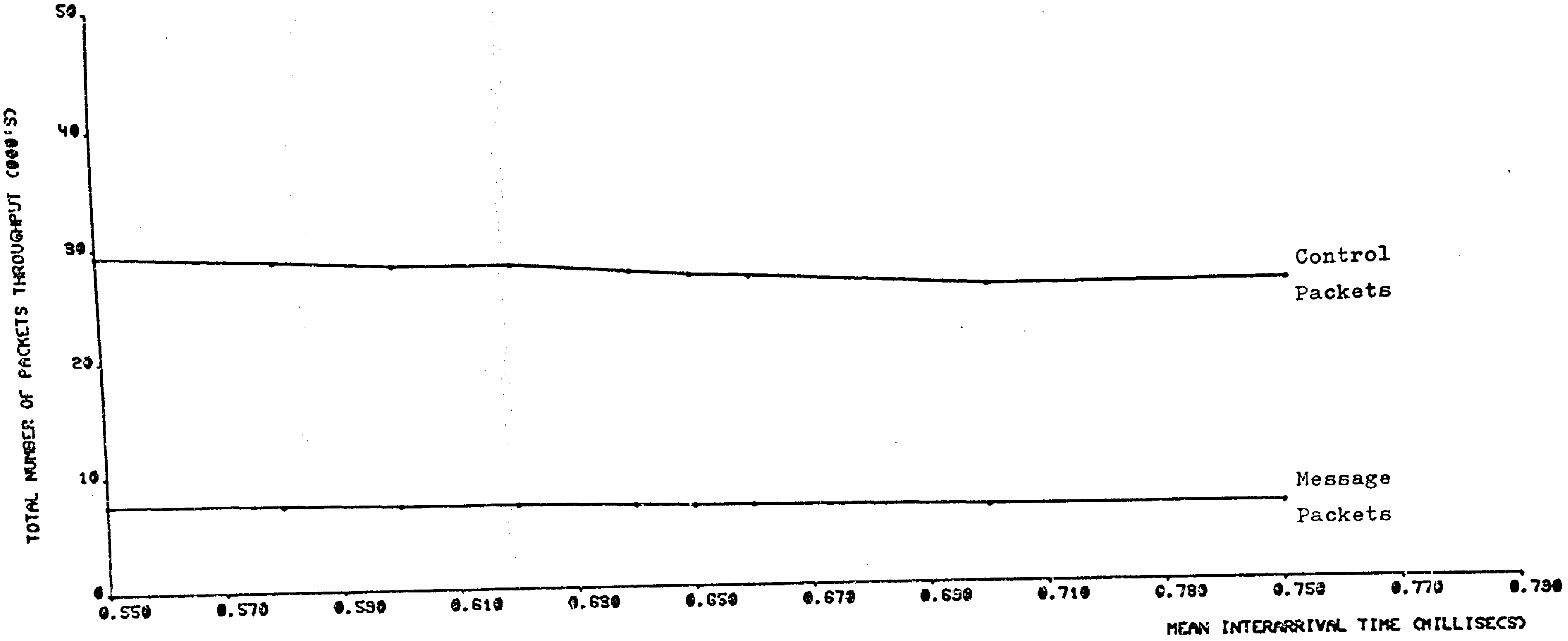


Figure 7.14 Standard Network - mean short message = 2 packets
- mean long message = 20 packets.
Effect of varying message mean
interarrival rate on network traffic.

This gives a 3% increase in throughput without taking nodal processing time into account. In order to obtain this 3% increase, processing time is doubled and memory modules and buffer sizes are doubled. It should also be borne in mind that acknowledgement time may also be higher for the longer packet size. The worst case as before,is for a newly generated acknowledgement where the first bit of a message packet has just been transmitted, thereby causing the buffer to take twice as long to empty.

7.3 Message delay

The time taken to transmit a packet from source to destination is a sum of three factors:

1) Propogation delays for the packet and its acknowledgement
2) Transmission delays for the packet and its acknowledgement
3) Node processing delay before an acknowledgement is sent.

A general approximation may be obtained for the time taken to throughput a message as given by:

$$\frac{\sigma T_s}{\mu_1} + \frac{(1-\sigma).T_l}{\mu_2}$$

where σ is the ratio of short messages to long

T_s is the time to throughput the mean short message

T_l is the time to throughput the mean long message

μ_1 is the mean short message

μ_2 is the mean long message.

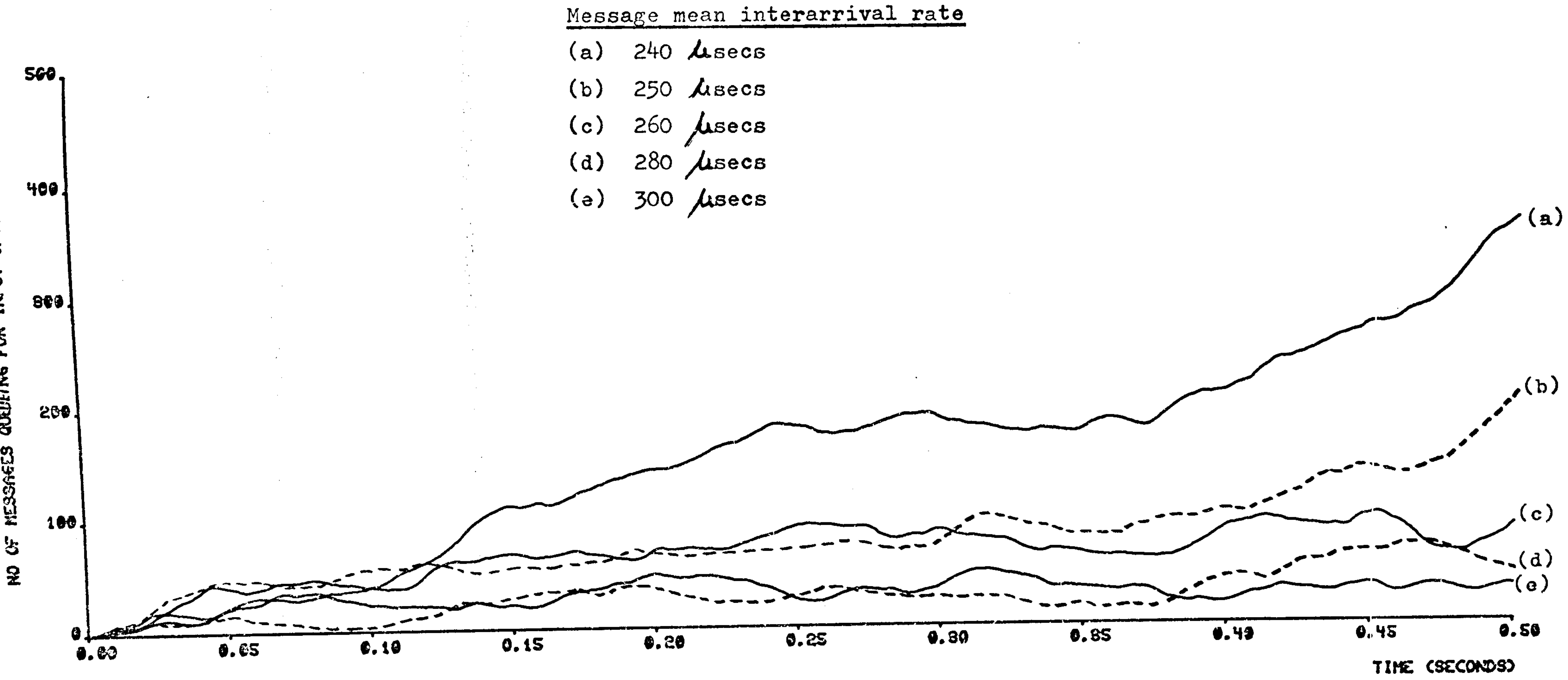


Figure 7.15 EWMA Standard Network - short/long message in ratio 9:1. Effect of varying message mean interarrival rate on message queue for input into network.

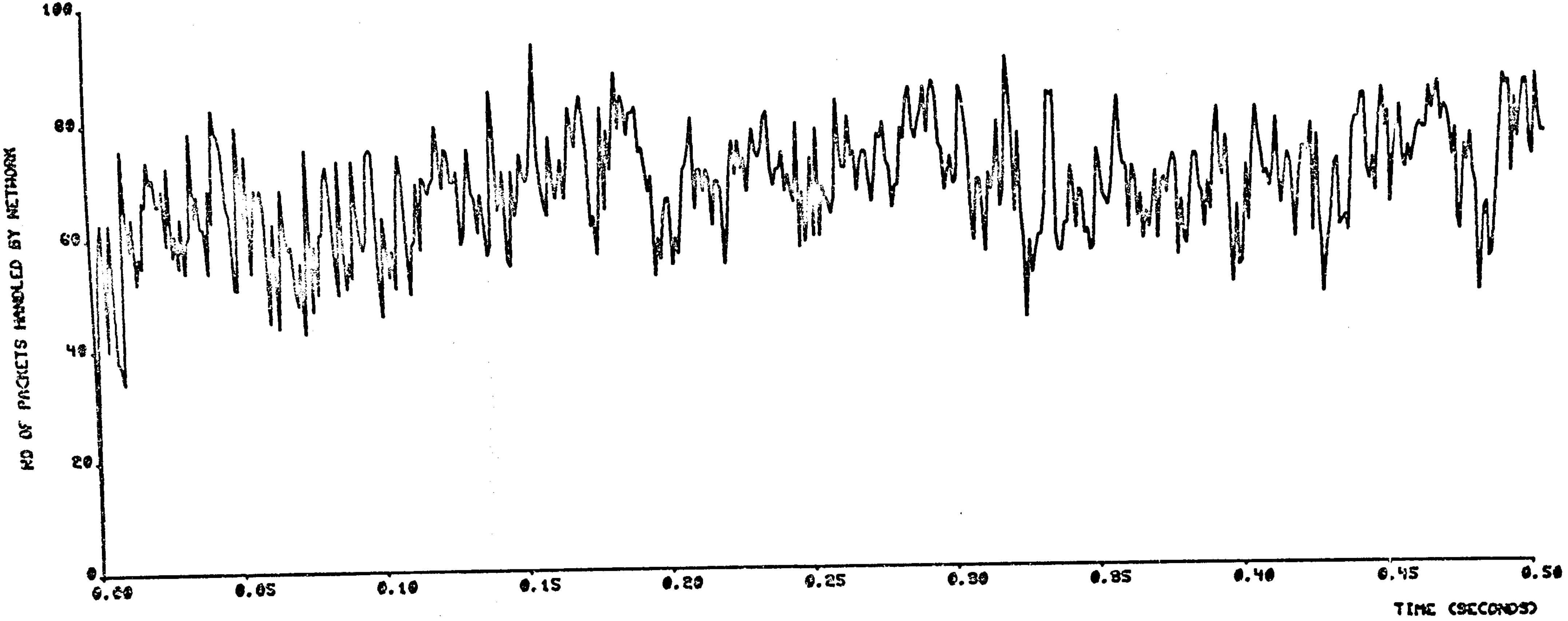


Figure 7.16 Standard Network - short/long message in ratio 9:1. Number of packets processed by network per millisecond sample interval.

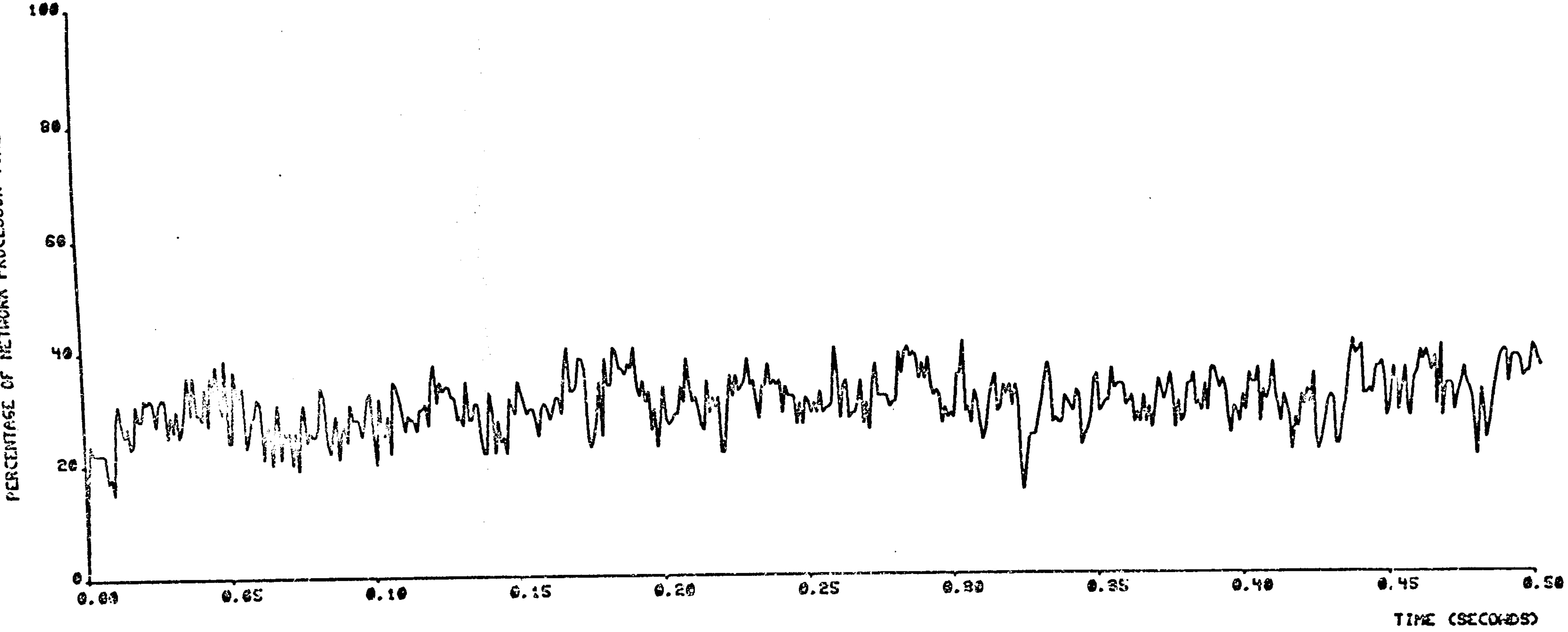


Figure 7.17 Standard Network - short/long message in ratio 9:1. Percentage of network processor time used per millisecond sample interval.

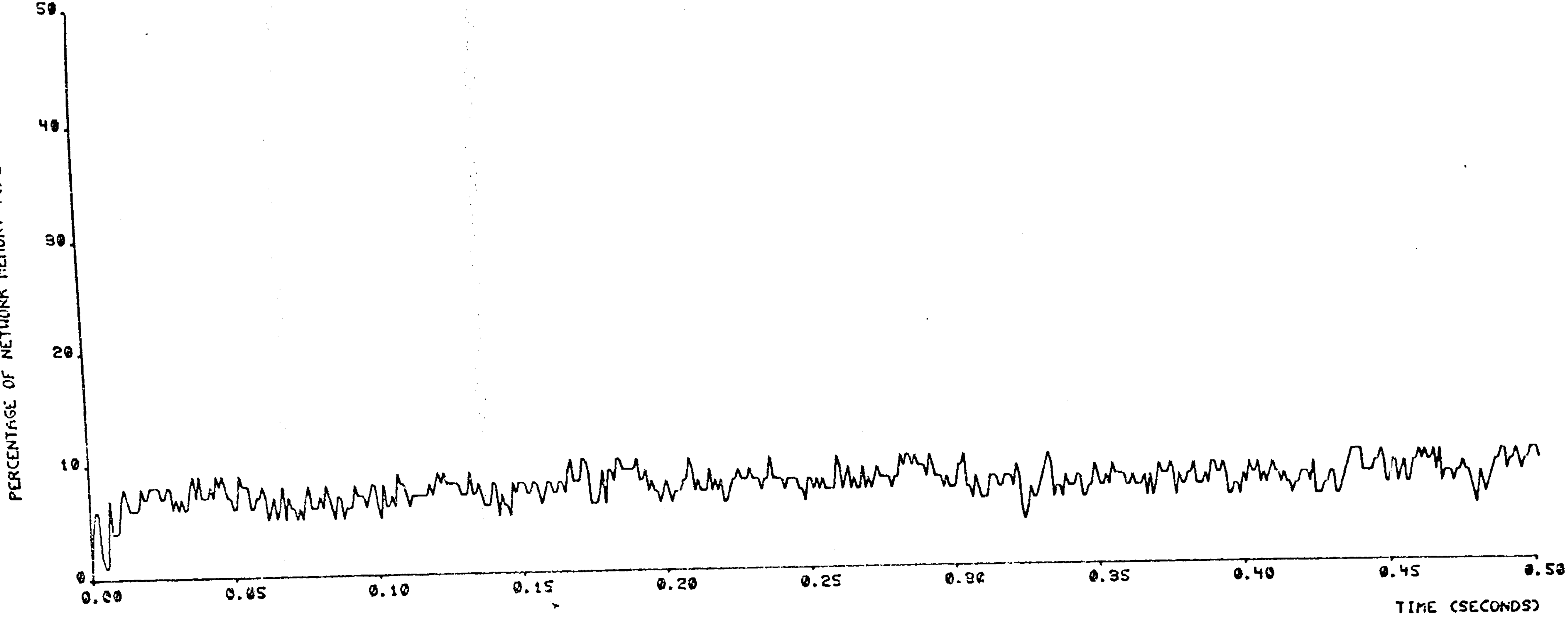


Figure 7.18 Standard Network - short/long message in ratio 9:1.
Percentage of network memory time used
per millisecond sample interval.

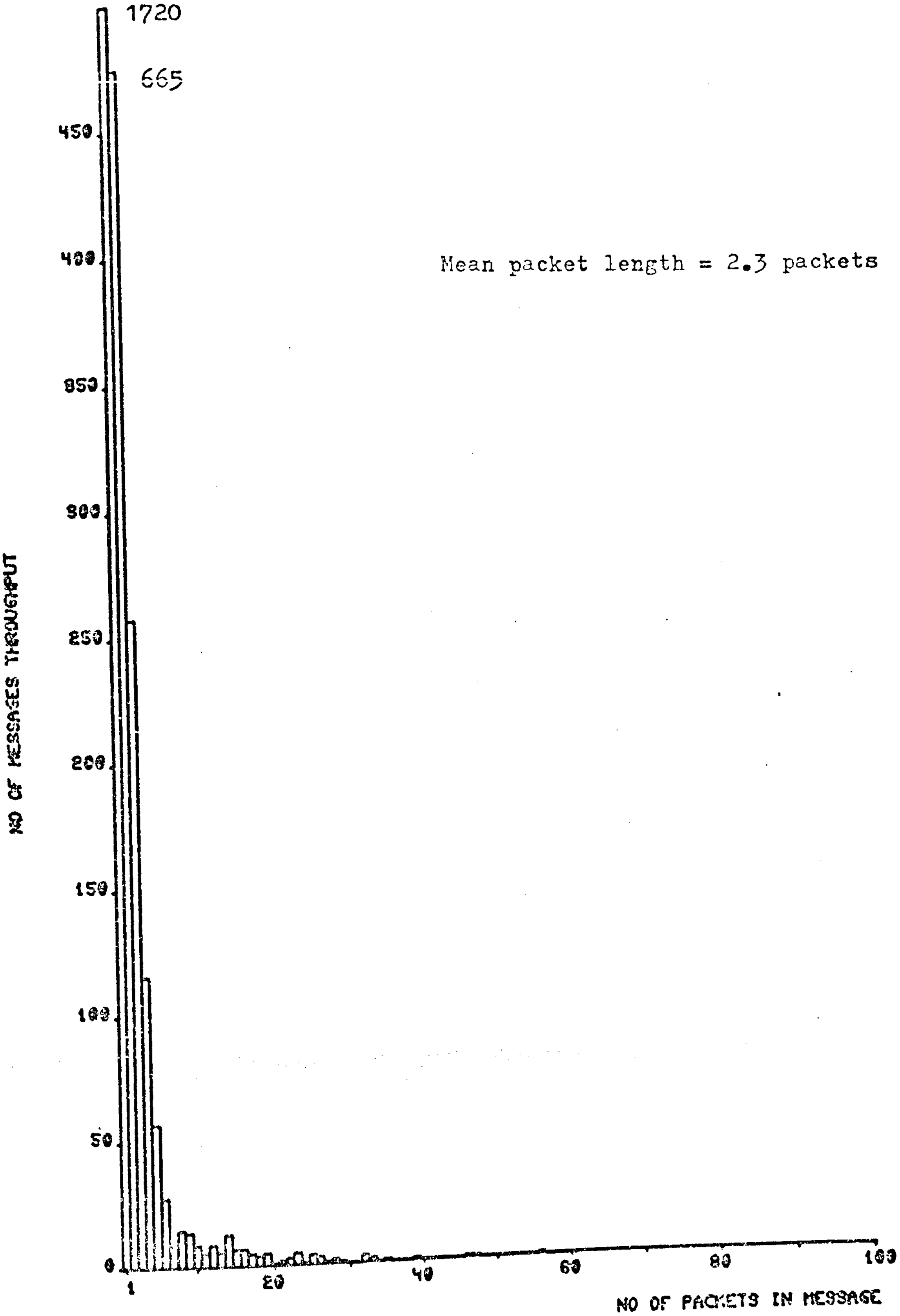


Figure 7.19 Standard Network - short/long message in ratio 9:1. Distribution of message lengths sent through network.

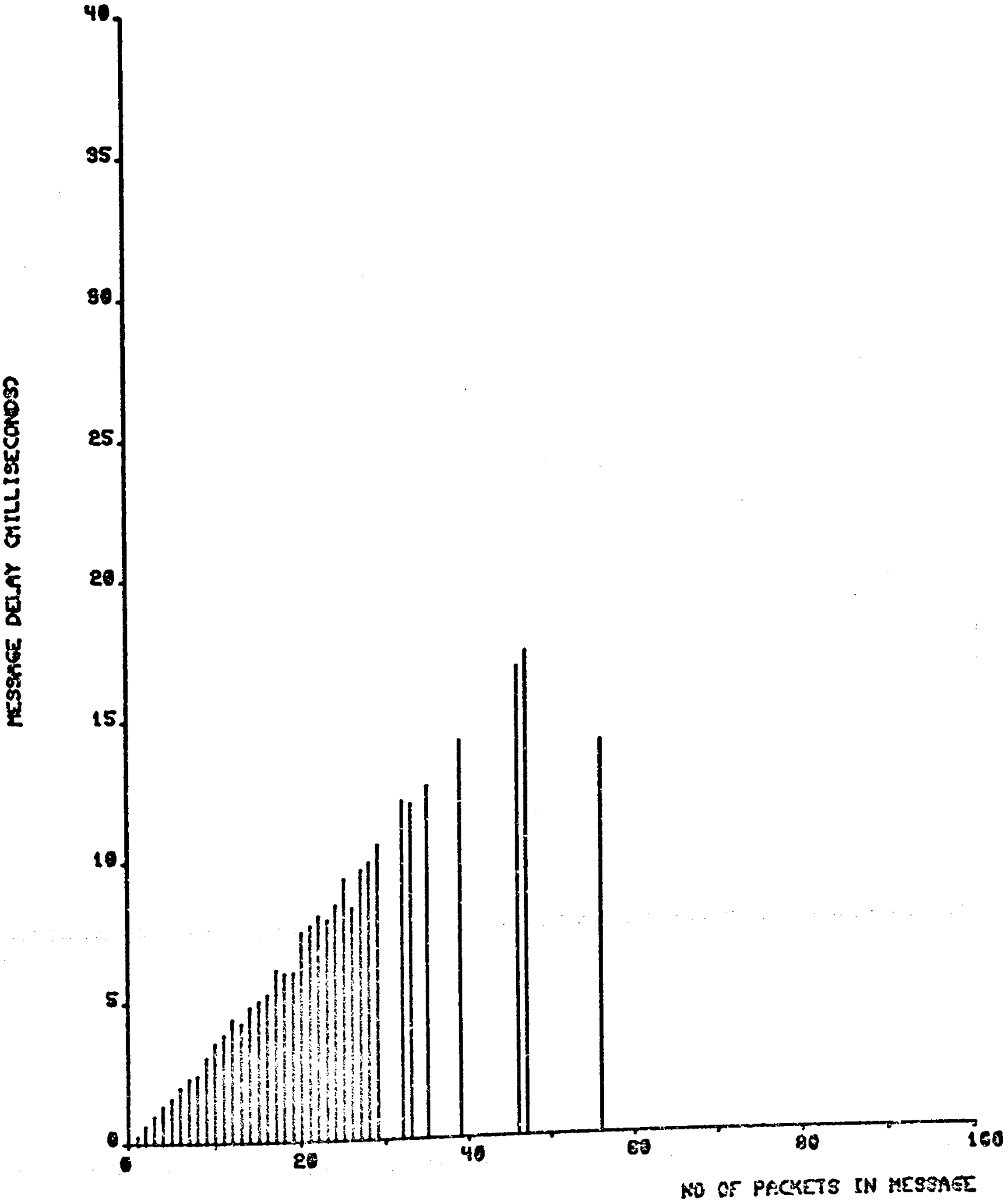


Figure 7.20 Standard Network - short/long message in ratio 9:1. Effect of message length on message throughput time.

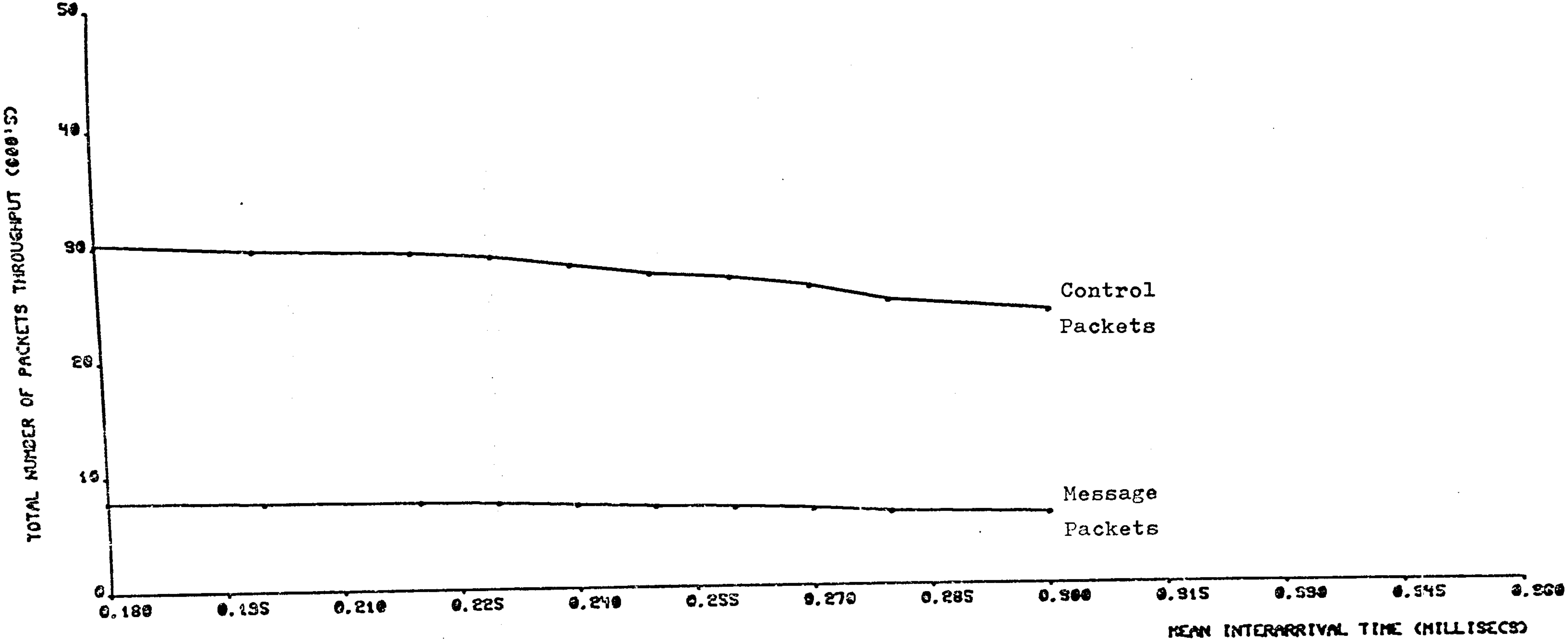


Figure 7.21 Standard Network - short/long message in ratio 9:1. Effect of varying message mean interarrival rate on network traffic.

7.4 Conclusions

This chapter presented the results of experiments on the low level network under a variety of conditions. The relationship between delay and throughput has been given as a linear equation. The equation will remain linear until a node or a line fails necessitating routing through an intermediate node. It has been shown that increasing a packet's size does not bring any great benefit, at a great increase in cost, and under certain conditions may in fact be detrimental by increasing packet time spent on the network.

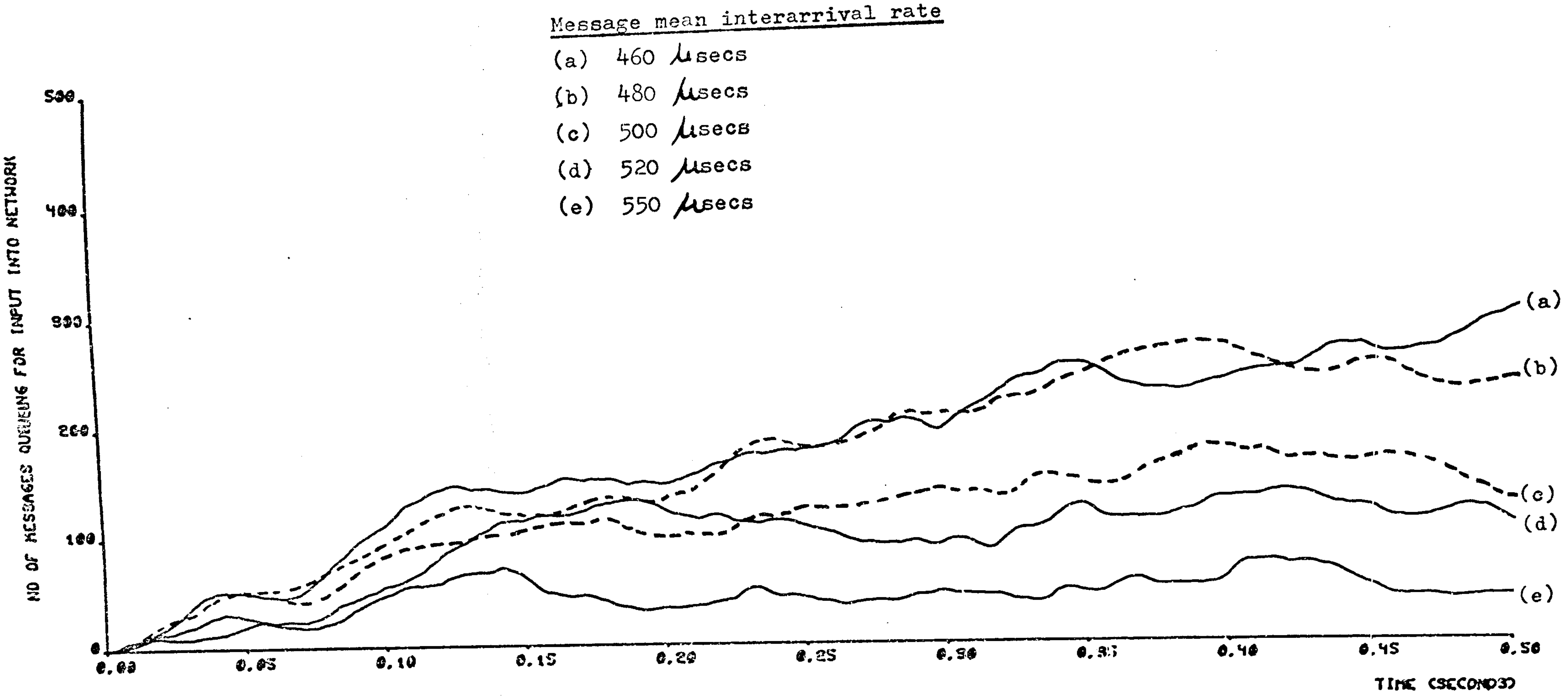


Figure 7.22 EWMA Standard Network - Mean number of generating hcsts = 2. Effect of varying message mean interarrival rate on message queue for input into network.

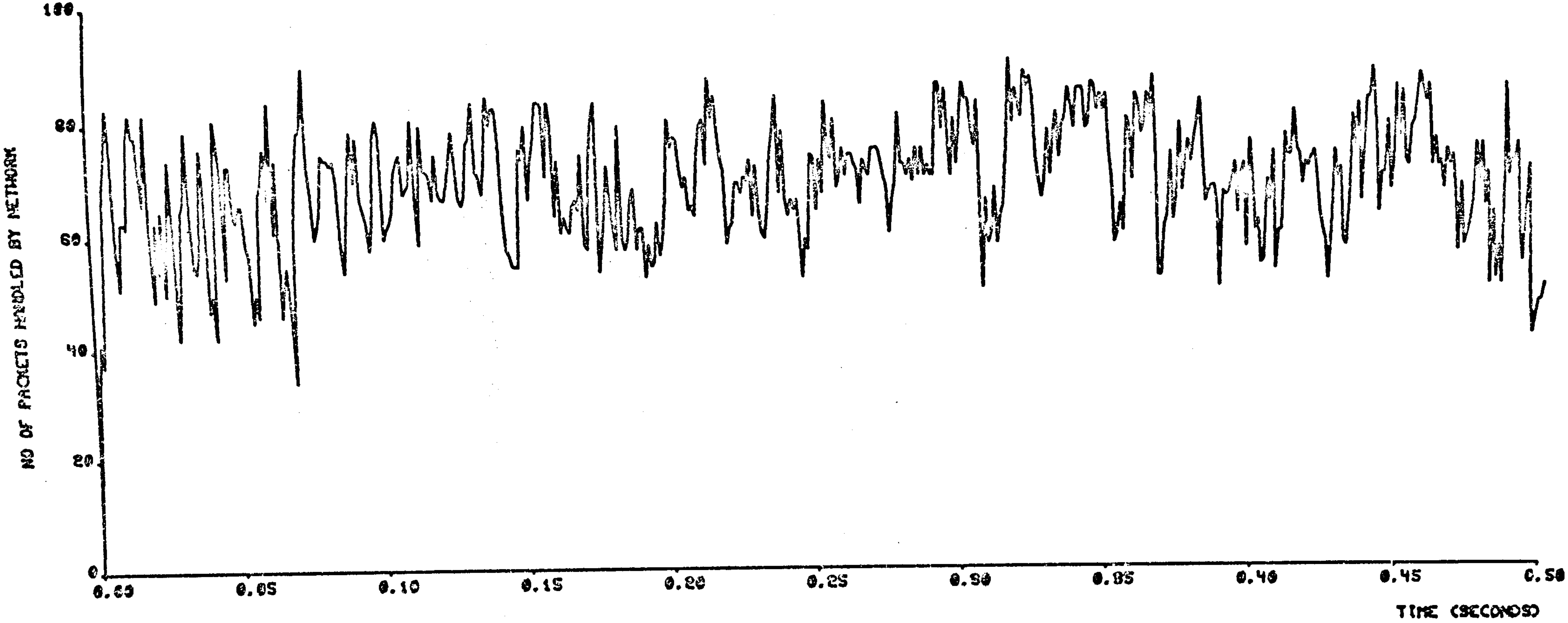


Figure 7.23 Standard Network - Mean number of generating hosts = 2.
Number of packets processed by network
per millisecond sample interval.

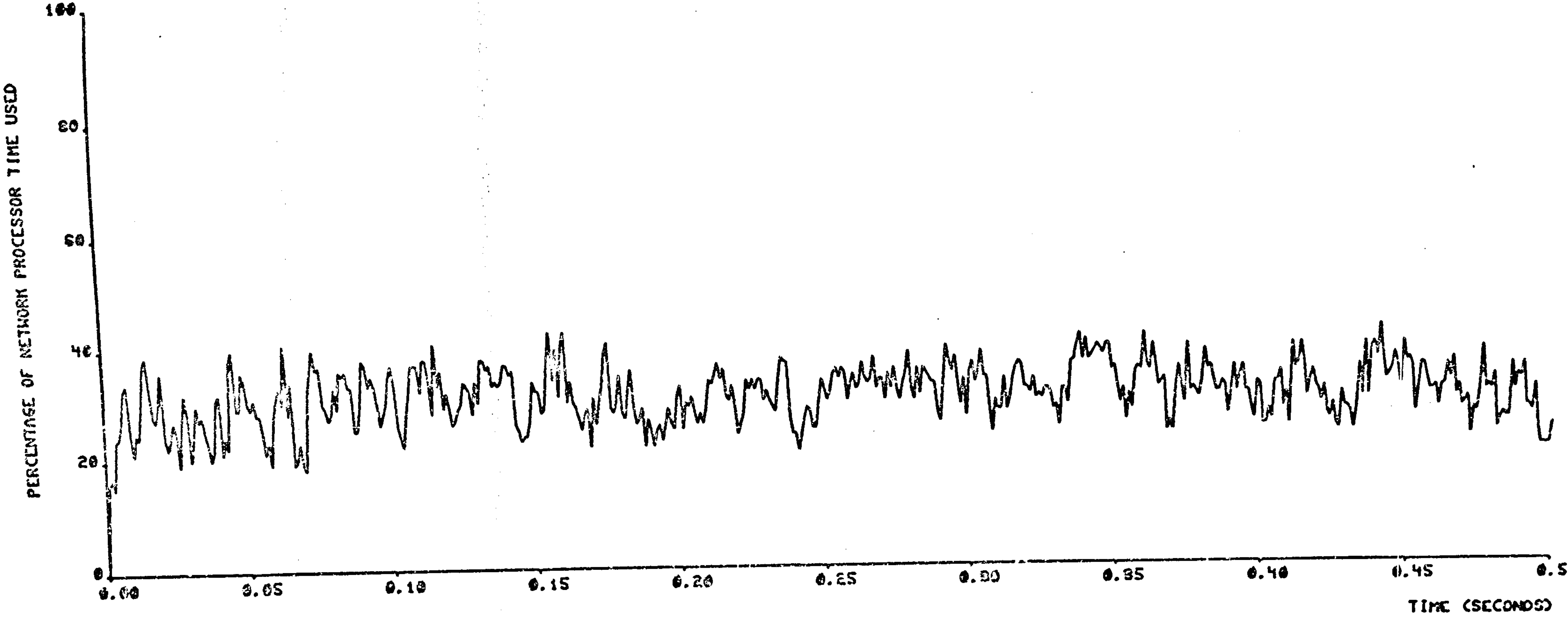


Figure 7.24 Standard Network - Mean number of generating hosts = 2. Percentage of network processor time used per millisecond sample interval.

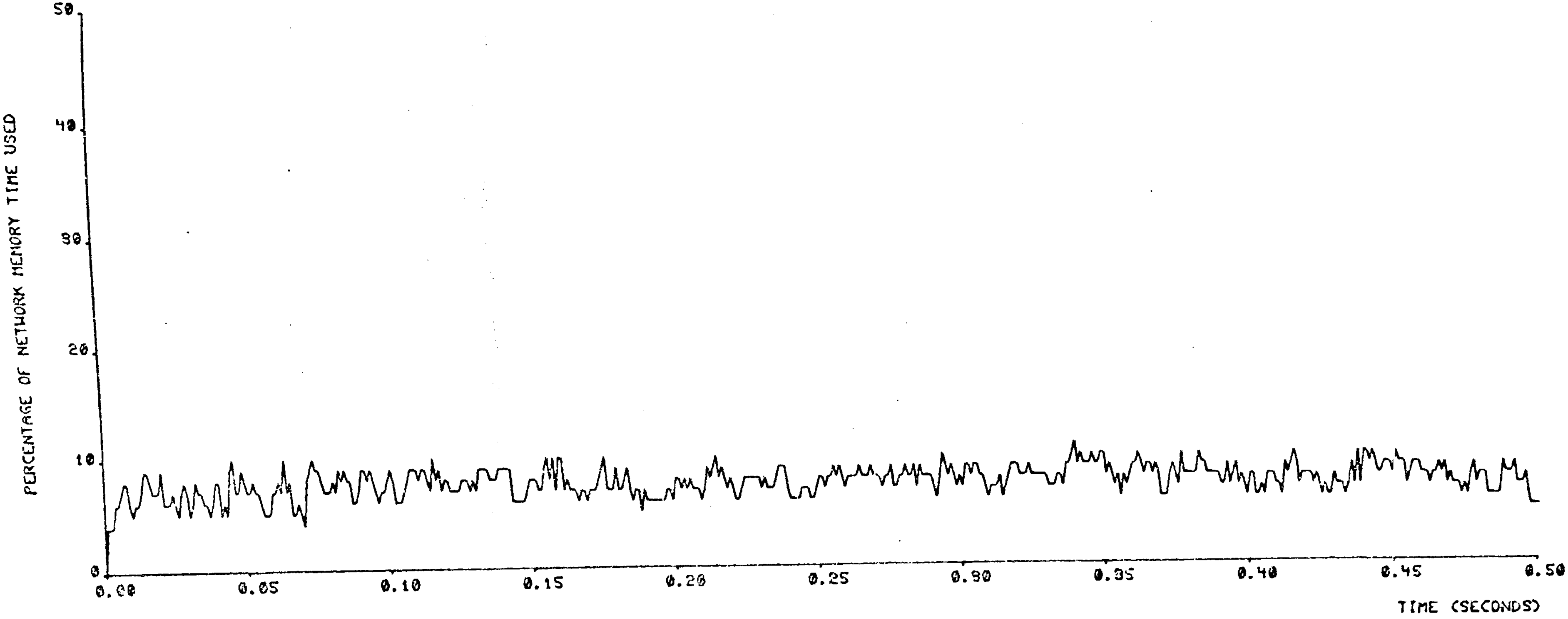


Figure 7.25 Standard Network - Mean number of generating hosts = 2. Percentage of network memory time used per millisecond sample interval.

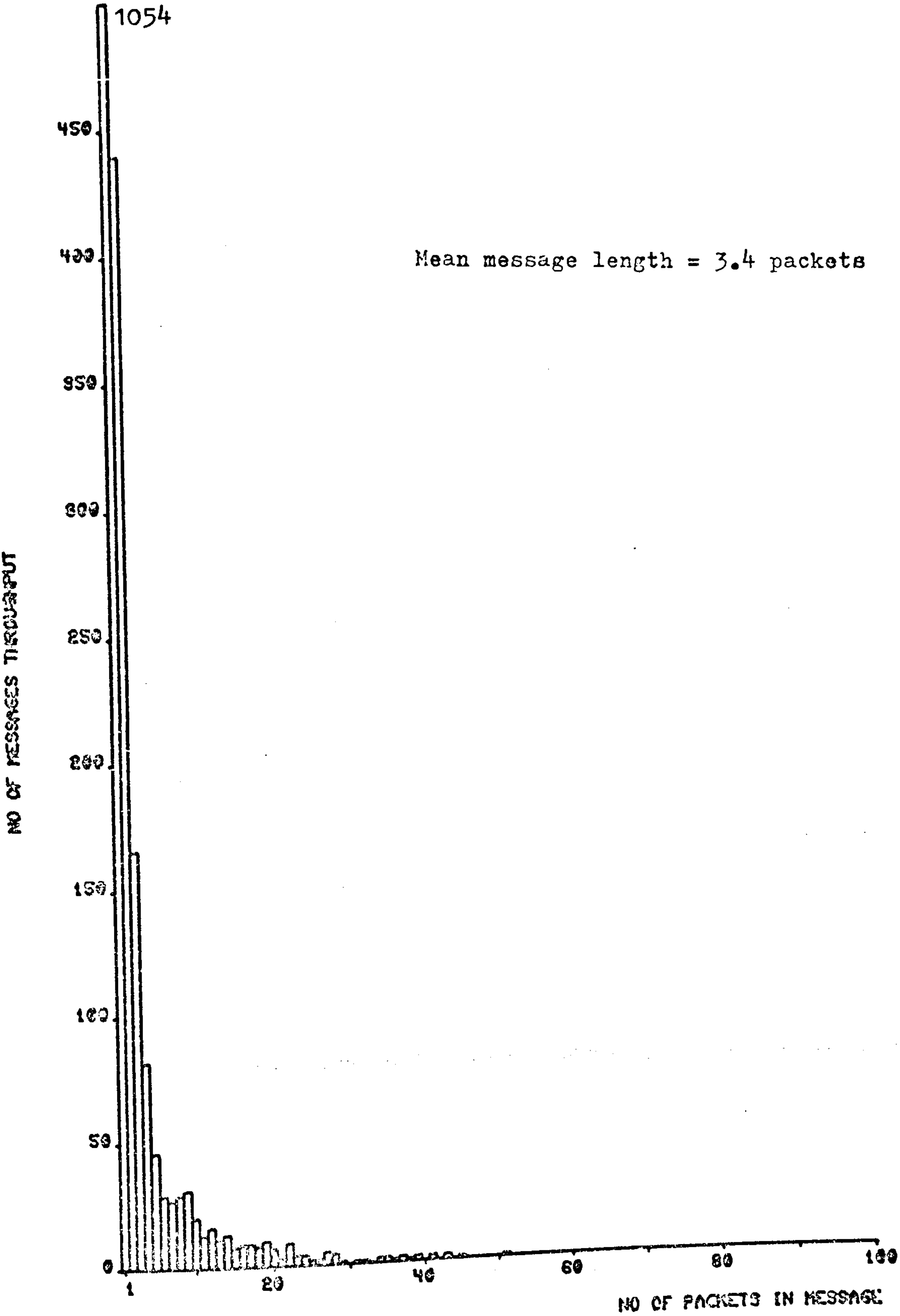


Figure 7.26 Standard Network - Mean number of generating hosts = 2. Distribution of message lengths sent through network.

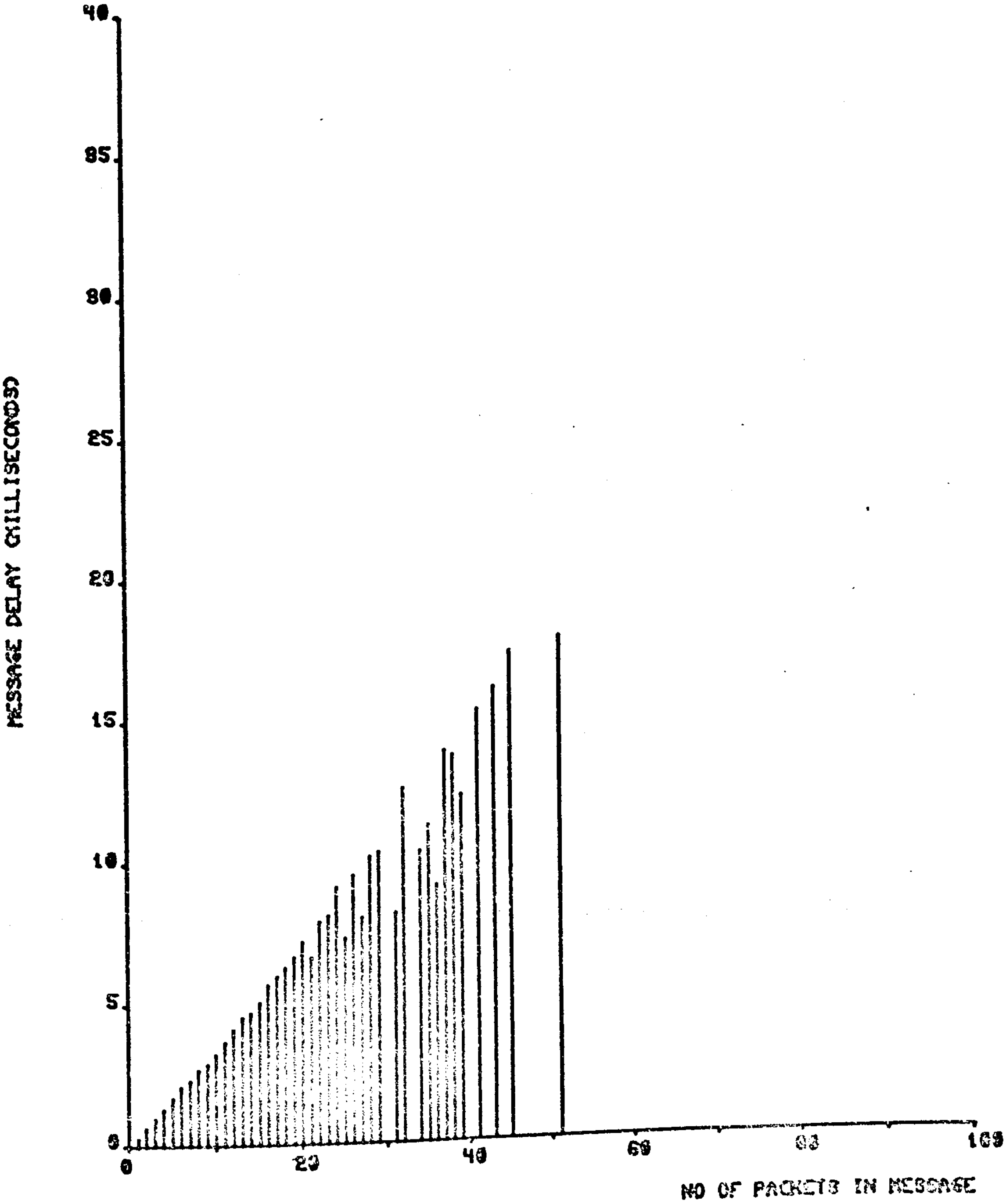


Figure 7.27 Standard Network - Mean number of generating hosts = 2. Effect of message length on message throughput time.

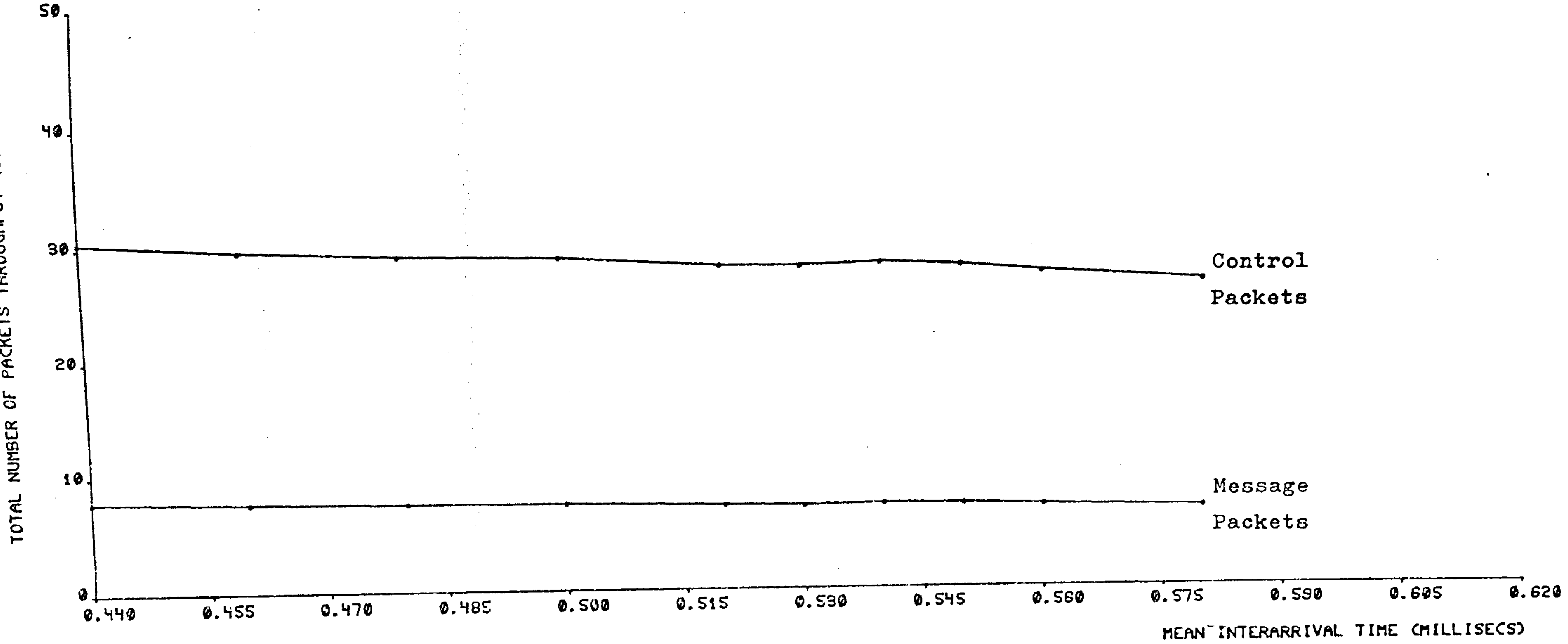


Figure 7.28 Standard Network - Mean number of generating hosts = 2.
Effect of varying message mean
interarrival rate on network traffic.

CHAPTER 8

General Conclusions and Suggestions for further work

This chapter makes some general conclusions about the investigation and makes some suggestions for future work. The use of a simulation study to aid this investigation has proved satisfactory for the results given. However, it must be noted that the computational cost was high. It was assumed at the beginning of the investigation that for a simple network of the kind considered the cost would be quite low. The detail simulated at the level of a processor/memory module handling individual packets through input/output buffers resulted in a small step size which increased the length and complexity of the simulation runs. This resulted in having a limited number of runs which did not enable the full scope of the model to be utilised. Had a simpler model been chosen, more runs could have been obtained. For example, given a symmetric traffic pattern it may have been wiser to simulate a node. It is felt that although the results were very useful in pinpointing deadlocks and quantifying the traffic that the network could sustain, the process of developing the simulation model was an important one in obtaining a deeper understanding of the overall system.

The experiments carried out in this investigation have tried to quantify the effects of parameters on throughput and message delay. On the basis of these results some general conclusions may be made regarding certain features of the network. As is to be expected the node itself is the limiting factor together with the line speeds. To ensure that these bottlenecks are reduced as far as possible the use of balanced processors and memory modules operating at say 100 nsecs per machine cycle are desirable. The use of two processors together with at least the same number of memory modules should ensure that in the event of a breakdown, the network will continue to function adequatelv and throughput should not diminish significantly. As has been shown, the higher the line speed the better the service. Line speeds should be as fast as possible, preferably in the 10 megabit region.

The last chapter discussed how doubling the standard packet length of 128 bytes (8 bits) gave throughput gains of only single percentage figures at a considerable increase in cost for larger memory modules and input/output buffers. There were also no visible advantages in having memory modules capable of storing more than one packet each - in fact it could slow the node down due to memory contention problems. Since 1024 bit memory modules are a standard product, it would seem logical to restrict packets to this length. In order to limit bottlenecks at the node as far as possible it would also be desirable to use microprogrammable microprocessors as opposed to MOS microprocessors since they are both faster and more flexible. The investigation has shown that a multiprocessor node is viable and gives distinct advantages by the provision of parallel processing and fail-soft capability.

During the period of this investigation new networks have come into existence and there is every reason to suppose that the state of the art is in the infant stage. Significant developments are continually emerging in microprocessor and memory technologies.

As was stated before, the full scope of the model has not been taken advantage of. It is suggested that further work could be carried out with this model to observe the effects of transient conditions e.g. the effects of processor/memory breakdown; the network response time to changes in traffic intensity; the effect of non-symmetric host traffic and the effect of passing large files over the network. The amount of traffic that the hosts might generate also needs to be investigated.

At this present time, a prototype node is in the process of being constructed and it is hoped that the results of this investigation will be of use in defining the node and network architecture.

REFERENCES

1. Hayes J. F., Sherman D. N. "Traffic and Delay in a Circular Data Network." 2nd ACM IEEE Symposium on Problems in the optimization of Data Communication Systems, Palo Alto, California, 1971, pp 102-107.

2. Peterson J. J., Veit S. A. "Survey of Computer Networks." MITRE Corporation Report MTP-357, September 1971.

3. Roberts L. G., Wessler B. D. "Computer Network development to achieve resource sharing." AFIPS, Vol 36, pp 543-549, May 1970.

4. Heart F. E., Kahn R. E., Ornstein S. M., Crowther W. R., Walden D. C. "The interface message processor for the ARPA computer network." AFIPS, Vol 36, May 1970, pp 551-567.

5. Frank H., Frisch I. T., Chou W. "Topological considerations in the design of ARPA computer network." AFIPS, Vol 36, May 1970, pp 581-587.

6. Carr C. S., Crocker S. D., Cerf V. G. "Host-Host communication protocol in the ARPA network." AFIPS, Vol 36, May 1970, pp 589-597.

7. Ornstein S. M., Heart F. E., Crowther W. R., Rising H. K., Russell S. B., Michel A. "The terminal IMP for the ARPA computer network." AFIPS, Vol 40, May 1972, pp 243-254.

8. Crocker S. D., Heafner J. F., Metcalfe R. M., Posbel J. B. "Function-oriented protocol for the ARPA computer network." AFIPS, 1972, Vol 40, pp 271-279.

9. McQuillan J. M., Crowther W. R., Cosell B. P., Walden D. C., Heart F. E. "Improvements in the design and performance of the ARPA network." AFIPS 1972, Vol 41 II, pp 741-754.

10. Frank H., Kahn R. E., Kleinrock L. "Computer communication network design - Experience with theory and practice."
(a) AFIPS 1972, Vol 40, pp 255-270.
(b) Networks - Vol 2. No 2. 1972, pp 135-166.

11. Kahn R. "Terminal Access to the ARPA computer network." Computer Networks. R. Rustin (Ed). Prentice Hall. Englewood Cliffs N. J., 1972, pp 147-166.

12. Kleinrock L., Naylor W. E. "On measured behavior of the ARPA Network." AFIPS 1974, Vol 43, pp 767-780.

13. Thomas R. H. "A resource sharing executive for the ARPANET." AFIPS, Vol 42, 1973, pp 155-163.

14. Cole G. D. "Performance measurement on the ARPA computer network." 2nd ACM IEEE Symposium on Problems in the optimization of Data Communication Systems, Palo Alto, California, 1971, pp 39-45.

15. Roberts L. G. "Extensions of packet communications technology to a hand-held personal terminal." AFIPS 1972, Vol 40, pp 295-298.

16. Luther W. J. "Conceptual Bases of Cybernet." Computer Networks. R. Rustin (Ed). Prentice Hall. Englewood Cliffs N. J. 1972, pp 111-146.

17. Farber D. J., Larson K. C. "The System Architecture of the Distributed Computer System - An Informal Description." University of California, Irvine, Technical Report, No. 11, September 1971.

18. Farber D. "Data Ring Orientated Computer Networks." Computer Networks. R. Rustin (Ed). Prentice Hall. Englewood Cliffs N. J. 1972, pp 79-93.

19. Herzog B. "Merit Computer Network." Computer Networks. R. Rustin (Ed). Prentice Hall. Englewood Cliffs N. J. 1972, pp 45-48.

20. Aupperle E. "Merit Computer Network: Hardware Consideration." Computer Networks. R. Rustin (Ed). Prentice Hall. Englewood Cliffs N. J. 1972, pp 49-63.

21. Cocanower A. "Merit Computer Network: Software Considerations." Computer Networks. R. Rustin (Ed). Prentice Hall. Englewood Cliffs N. J. 1972, pp 65-77.

22. Mendicino S. F. "Octopus: The Lawrence Radiation Laboratory Network." Computer Networks. R. Rustin (Ed). Prentice Hall. Englewood Cliffs N. J. 1972, pp 95-110.

23. Weis A. H. "Distributed Network Activity at IBM." Computer Networks. R. Rustin (Ed). Prentice Hall. Englewood Cliffs N. J. 1972, pp 1-25.

24. Williams L. H. "A functioning computer network for higher Education in North Carolina," AFIPS Vol 41 II, 1972, pp 899-904.

25. Coleman M. L. "Accnet - A corporate computer network." AFIPS, Vol 42, 1973, pp 133-140.

26. Fisher C. R., Sligh R. L. "The Datran Network." 2nd ACM IEEE Symposium on problems in the Optimization of Data Communications Systems, Palo Alto, California, 1971, pp 65-72.

27. Scantlebury R. A., Wilkinson P. T. "The design of a switching system to allow remote access to computer services by other computers and terminal devices." 2nd ACM IEEE Symposium on problems in the Optimization of Data Communications Syatems, Palo Alto, California, 1971, pp 160-167.

28. Belton R. C., Smith M. A. "Introduction to the British Post Office Experimental Packet-Switching Service (E.P.S.S.)" Post Office Elec. Eng. Journal, Vol 66, January 1974, pp 216-218.

29. Kleinrock L. "Survey of Analytical methods in queuing networks." Computer Networks, R. Rustin (Ed). Prentice Hall. Englewood Cliffs N. J. 1972, pp 185-205.

30. Kleinrock L. "Analytic and simulation methods in computer design." AFIPS, Vol 36, 1970, pp 569-579.

31. Slyke R. V., Chou W., Frank H. "Avoiding simulation in simulating computer communication networks." AFIPS, Vol 42, 1973, pp 165-169.

32. Bowden E. K., Mamrak S. A., Salz F. R. "Simulation - A tool for performance evaluation in network computers." AFIPS, Vol 42, 1973, pp 121-131.

33. White G. W. "Message Format Principles." 2nd ACM IEEE Symposium on problems in the Optimization of Data Communications Systems, Palo Alto, California, 1971,pp 192-198.

34. Karp D., Seroussi S. "A communication Interface for computer networks." 2nd ACM IEEE Symposium on problems in the Optimization of Data Communications Systems, Palo Alto, California, 1971, pp 117-123.

35. Davis D. W. "The Control of Congestion in packet switching Networks." 2nd ACM IEEE Symposium on problems in the Optimization of Data Communications Systems, Palo Alto, California, 1971, pp 46-49.

36. Gardner A. J., Sandum K. N. "Experimental Packet-Switched Service: Routing of Packets." Post Office Elec. Eng. Journal, Vol. 68, 1975, pp 235-239.

37. Esau C. R., Williams K. C. "A method for approximating the optimal network." IBM System Journal, Vol 5, No 3, 1966, pp 142-147.

38. Frank H. "Optimal design of Computer networks." Computer Networks, R. Rustin (Ed), Prentice Hall, Englewood Cliffs , N. J., 1972, pp 167-183.

39. Hamer M. "Reliability Modelling considerations for a real-time control system." IEEE, 1974, Fault Tolerant Computing Symposium, pp 2-2 to 2-7.

40. Avižienis A. "Architecture of fault-tolerant Computing Systems." International Symposium on Fault-Tolerant Computing, FTC-5 , 1975, Paris, France, pp 3-16.

41. Borgeson B. R. "A fail-softly system for time-sharing use". IEEE, 1972, Fault-Tolerant Computing Symposium, pp 89-93.

42. Hopkins A. L., Smith T. B. "The architectural Elements of a symmetric Fault-Tolerant multiprocessor". Fault-Tolerant Computing Symposium, FTC/3, pp 4-2 to 4-6.

43. Cox D. R., Smith W. L. "Queues". Methuen, 1961.

44. Saaty T. L. "Elements of queueing theory". McGrawHill, 1961.

45. Coffman E. G., Wood R. C. "Interarrival Statistics for TSS". System Development Corporation, SP-2161, August 1965.

46. Tocher K. D. "The Art of Simulation". Hodder and Stoughton, 1975.

47. Green D. H. "Data Monitoring and Smoothing". Digital Simulation Methods, IEE Monogram Series No. 15,(Ed) M.G. Hartley, 1975, pp 184-201.

48. Cox D. R. "Prediction by Exponentially Weighted Moving Averages and Related Methods". J. Royal Statistical Society, Series B, Vols 23-24, 1961.

49. Blackman R. B., Tukey J. W. "The measurement of Power Spectra".. Dover N. J., 1958.

50. Ratcliffe J. F. "Elements of Mathematical Statistics". Oxford Mathematical Handbooks.

51. Southworth R. W. "Autocorrelation and spectral Analysis", Mathematical Methods for Digital Computers. (Ed) A. Ralston & H. S. Wilf, Wiley. Vol 1, 1967.

52. GreenD. H., Hartley M. G. "Simple Pseudo-random Generator". Digital Simulation Methods, IEE Monogram Series No. 15, (Ed) M. G. Hartley, 1975, pp 35-62.

53. Hartley M. G. "Modelling technique for traffic studies". Ph.D. Thesis, Manchester, 1968.

54. Heath F. G. "Digital Codes and Converters". Ph.D. Thesis, Manchester, 1961.

55. Redshaw S. "A repeatable random pulse generator using chain-codes". M.Sc. Tech. Thesis, Manchester, 1961.

APPENDIX I

THE PSEUDO-RANDOM GENERATOR

The basis of the pseudo-random generator used are binary Chaincodes which may be defined as a sequence of 2^n or fewer binary digits arranged so that any n adjacent digits locate the position of those digits uniquely. Figure I.1 illustrates the uniqueness of each set of four adjacent digits in the sequence.

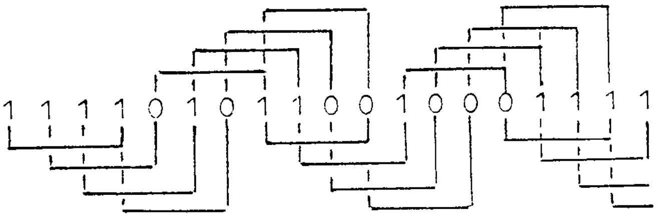


Figure I.1 Unique sets of digits in sequence

As the number of digits in the pattern increases, so the number of different patterns obtainable increases. The particular usefulness of these patterns is that they may be generated using a shift register. The particular chaincode illustrated above may be generated from a four-stage shift register as shown in figure I.2.

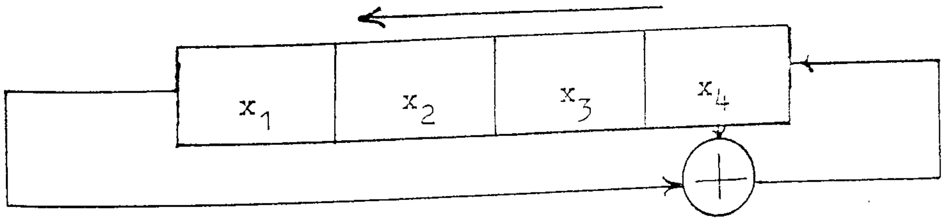


Figure I.2 Chaincode produced by four stage shift register

The new digit is produced by modulo-2 addition of the digits in stages x_1 and x_2. It can be seen that this shift register will produce 15 unique four-digit patterns before repeating the sequence. This is an example af a Linear chaincode.

Heath [54] categorised chaincodes into several classes; Prime, Deficient Prime, Product and Skew Symmetric. Chaincode sequences have the property that they contain approximately equal numbers of ones and zeros, 2^{n-1} and $2^{n-1}-1$ respectively which may be employed in generating uniformly distributed numbers.

Redshaw [55] employed Prime chaincodes for the generation of pseudo-random numbers using two independent chaincodes as shown in figure I.3 to overcome the effects of correlation i.e. the state of the stage becomes the state of the following stage after the next shift pulse.

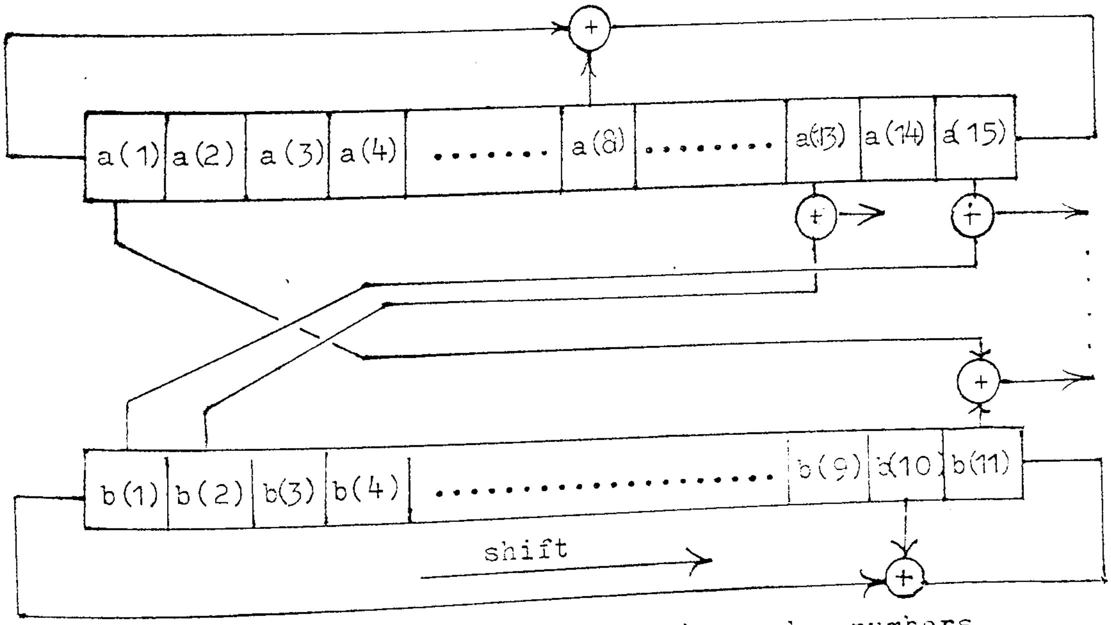


Figure I.3 Generation of pseudo-random numbers using two independent chaincodes.

Pairs of pick-offs, one from each chaincode are connected to modulo-2 adders which in turn are connected to an 'AND' gate. Given chaincodes of m and n stages, then the total sequence length is given by the product $(2^m-1)(2^n-1)$. With an eleven and fifteen stage generator sequences of length greater than 67 million were produced.

The probability of a stage containing a one or zero at any time is $\frac{1}{2}$ as is the probability of the output from a modulo-2 adder. For an 'AND' gate having k inputs from k modulo-2 adders the probability of a one is given by $(\frac{1}{2})^k$. The number of ones obtained from the 'AND' gate in one sequence is given by $(\frac{1}{2})^k(2^m-1)(2^n-1)$. A coarse control of the output rate may be obtained by varying the value of k. The output rate is doubled by reducing the value of k by one.

A finer control may be used by forming a binary number from the modulo-2 adder outputs instead of the 'AND' gate and the control of the number of inputs. This number is a uniformly distributed random number. This integer is compared to a control word and only output if it exceeds the control word. The fineness of control of the output frequency is now determined by the maximum value of the picked-off word, e.g. a ten-bit word picked-off will permit uniformly distributed numbers in the range 0 - 1023 to be obtained. A change in the control word by one is approximately equal to a change in the output rate of 0.1%.

This type of pseudo-random generator is easily implemented in software form. Redshaw[55] and Hartley[53] have carried out extensive tests upon the random properties of this type of generator finding them acceptable means of producing the required random numbers.

APPENDIX II

This section contains the complete simulation program.

```
C      SIMULATION OF ASTONET
       INTEGER NET(5,5),NETB(5,5),REJ(5,5),PKT(500,5),NODE(5),
     XLK(5),H(5),NBUFI(5,3),NBUFF(5,50),CBUFI(5,3),CBUFF(5,50),
     XHPBI(5,5,3),HCBI(5,5,3),HPB(5,5,50),HCB(5,5,50),CMEM(5,50),
     XNODEI(5,5),NIB(5,5),NOB(5,5),PFL(5,10),MFL(5,10),AVAIL(5),
     XHOSTS(5,5),HOSTC(5,5),HQL(5,5),HMD(5,5),CHQ(25),MG(5,5),
     XHPSN(5,5),HPM(5,5),HIB(5,5),HOB(5,5),NMEM(5,50),A(16),B(14),
     XSNET(5,5),MEMS(5,10),PTY(5),ML(200),SNC(5),RNC(5),BFL(5),
     XRNIB(5,5),RNOB(5,5),RHIB(5,5),RHOB(5,5),PORD(5,5),UPRP(5),
     XCUM(600),HCF(25),CFQ(600),VEC(9),CH1(21),CH2(21),CH3(21),
     XCH4(21),CA1(21),CA2(21),CA3(21),CA4(21)
       INTEGER CP1(21),CP2(21),CP3(21),CP4(21)
       REAL NPLR(5,5),NCLR(5,5),HPLR(5,5),HCLR(5,5),
     XNIE(5,5),NOE(5,5),BRN(5),BRP(5,10),BRM(5,10),STI(10,2),
     XRHIE(5,5),RHOE(5,5),RNIE(5,5),RNOE(5,5),TPROC(5,10),
     XPUSE(5,10),HNIE(5,5),HNOE(5,5),PNE(5,10),MNE(5,10),TPT(5),
     XTIN(25,25),TMT(200),NSP(5,3),TPIN(500,2),HNU(5,5)
       INTEGER NH,HH,PN,PACK,P,M,BR,RUNNO,PNO,HC,SR,MSM,MLM,
     XMNGN,SUM,TOTAL,CLASS,TYPE,PACK,HHH,MINQL,MAXQL,TQL,AVQL,
     XPACKD,PD,RP,DEST,AA,PACKS,SOUR
       REAL SMALL,TIME,MESS,NCHECK,CHECK,LGSIM,HARD,PKLGH,
     XHN,MMR,OCC,CNTP,R,BITS,NHARD,TI,FIA,FSM,FLM,RC,EXEC,
     XQL,AAA,BBB,NUM,THRU,TNREC,XNREC,F,FX,AAS,E1,E2,TT,
     XPU1,PU2,MUSE,NOPR,NOMEM


C      CLEAR ARRAYS

       DO 2030 I=1,N
       DO 2040 J=1,5
       REJ(I,J)=0
       NIB(I,J)=0
       NOB(I,J)=0
       NIE(I,J)=0.0
       NOE(I,J)=0.0
       RNIB(I,J)=0
       RNOB(I,J)=0
       RHIB(I,J)=0
       RHOB(I,J)=0
       HNIE(I,J)=0.0
       HNOE(I,J)=0.0
       RHIE(I,J)=0.0
       RHOE(I,J)=0.0
       RNIE(I,J)=0.0
       RNOE(I,J)=0.0
       HQL(I,J)=0
       HMD(I,J)=0
       HPSN(I,J)=0
       HPM(I,J)=0
       HIB(I,J)=0
       HOB(I,J)=0
 2040  CONTINUE
       DO 2080 J=1,10
       PNE(I,J)=0.0
```

```
      MNE(I,J)=0.0
      PFL(I,J)=0
      MFL(I,J)=0
      MEMS(I,J)=0
      TPROC(I,J)=0.0
      PUSE(I,J)=0.0
 2080 CONTINUE
      PTY(I)=0
      SNC(I)=0
      RNC(I)=0
      TPT(I)=0.0
      BRN(I)=0.0
 2030 CONTINUE
      DO 2050 I=1,500
      DO 2051 J=1,5
 2051 PKT(I,J)=0
      TPIN(I,1)=0.0
      TPIN(I,2)=0.0
 2050 CONTINUE
      DO 2060 I=1,5
      DO 2060 J=1,50
      NBUFF(I,J)=0
      NMEM(I,J)=0
      CBUFF(I,J)=0
      CMEM(I,J)=0
 2060 CONTINUE
      DO 2085 I=1,N
      DO 2085 J=1,5
      DO 2075 K=1,50
      HPB(I,J,K)=0
 2075 HCB(I,J,K)=0
 2085 CONTINUE
      DO 2095 I=1,16
 2095 A(I)=0
      DO 2096 I=1,14
 2096 B(I)=0
      DO 2097 I=1,25
      DO 2097 J=1,25
 2097 TIN(I,J)=0.0
      DO 2098 I=1,200
      TMT(I)=0.0
 2098 ML(I)=0
      DO 2063 I=1,21
      CH1(I)=0
      CH2(I)=0
      CH3(I)=0
      CH4(I)=0
      CA1(I)=0
      CA2(I)=0
      CA3(I)=0
      CA4(I)=0
      CP1(I)=0
      CP2(I)=0
      CP3(I)=0
 2063 CP4(4)=0
```

```
C      INPUT NETWORK PARAMETERS
C      ************************

       CALL DINPUT(N,LK,NET,NETB,NPLR,NCLR,NODE,NODEI,
     XNBUFI,CBUFI,H,HPLR,HCLR,HOSTS,HOSTC,HNU,NH,LGSIM,PKLGH,
     XCNTP,CHECK,HPBI,HCBI,MMR,SNET,NSP,BR,BRN,BRP,BRM,RUNNO,
     XMSM,MLM,MNGN,HCF,BITS,HARD,TI,STI)

C      INITIALISE RANDOM NUMBER ROUTINES
C      *********************************

       DO 4507 I=2,14,2
       A(I)=0
       B(I)=1
       A(I+1)=1
 4507  B(I+1)=1
       AAS=32767.0
       E1=0.0
       DO 4504 I=1,600
 4504  CFQ(I)=0
       TT=0.0
       DO 4503 I=1,600
       TT=TT+0.01
       E2=1.0-EXP(-TT)
       CFQ(I)=INT((E2-E1)*AAS)
       E1=E2
 4503  CONTINUE
 4505  ISUM=0
       DO 4501 I=1,600
       ISUM=ISUM+CFQ(I)
 4501  CUM(I)=ISUM
       WRITE (2,9010)
       DO 4506 I=1,600,10
 4506  WRITE (2,8100) I,CUM(I),CUM(I+1),CUM(I+2),CUM(I+3),
     XCUM(I+4),CUM(I+5),CUM(I+6),CUM(I+7),CUM(I+8),CUM(I+9)
       DO 3075 I=2,16,2
       CH1(I)=1
       CH2(I)=0
       CH3(I)=0
       CH4(I)=0
       CH1(I+1)=1
       CH2(I+1)=1
       CH3(I+1)=1
       CH4(I+1)=1
       CA1(I)=1
       CA2(I)=0
       CA3(I)=0
       CA4(I)=0
       CA1(I+1)=1
       CA2(I+1)=1
       CA3(I+1)=1
       CA4(I+1)=1
       CP1(I)=1
       CP2(I)=0
       CP3(I)=0
```

```
      CP4(I)=0
      CP1(I+1)=1
      CP2(I+1)=1
      CP3(I+1)=1
3075  CP4(I+1)=1
      FIA=TI
      WRITE (2,9020) RUNNO,FIA
      FIA=FIA*0.01
      R=MSM
      FSM=R*0.01
      R=MLM
      FLM=R*0.01
      MAXH=HCF(NH)

      DO 3071 I=1,N
      K=NODEI(I,1)
C     NO OF PROCESSORS IN NODE I
      DO 3071 J=1,K
3071  PFL(I,J)=J
      DO 3072 I=1,N
      K=NODEI(I,2)
C     NO OF MEMORY UNITS IN NODE I
      DO 3072 J=1,K
3072  MFL(I,J)=1
      DO 3073 I=1,N
      PORD(I,1)=1
      PORD(I,2)=2
      PORD(I,3)=3
3073  PORD(I,4)=4
      MESS=0.0
      TIME=0.0
      NREC=0
      TNREC=0.0
      NPKT=0
      HN=NH
      NO=0
      NCHECK=CHECK
      NHARD=HARD
      EXEC=0.0
      MUSE=0.0
      PU1=0.0
      PU2=0.0
      I=NODEI(1,1)+NODEI(2,1)+NODEI(3,1)+NODEI(4,1)+NODEI(5,1)
      NOPR=I
      I=NODEI(1,2)+NODEI(2,2)+NODEI(3,2)+NODEI(4,2)+NODEI(5,2)
      NOMEM=I
      GOTO 3050
```

```
C     FIND SMALLEST EPOCH IN NIE,NOE,HNIE,HNOE,PNE,MNE,MESS,NCHECK,HARD
C     *****************************************************************
5000  SMALL=32767.0
      EXEC=EXEC+1.0
      DO 1400 I=1,N
      DO 1410 J=1,5
      IF ((HNIE(I,J).LT.SMALL).AND.(HNIE(I,J).NE.0.0)) SMALL=HNIE(I,J)
      IF ((HNOE(I,J).LT.SMALL).AND.(HNOE(I,J).NE.0.0)) SMALL=HNOE(I,J)
      IF ((NIE(I,J).LT.SMALL).AND.(NIE(I,J).NE.0.0)) SMALL=NIE(I,J)
      IF ((NOE(I,J).LT.SMALL).AND.(NOE(I,J).NE.0.0)) SMALL=NOE(I,J)
      IF ((RHIE(I,J).LT.SMALL).AND.(RHIE(I,J).NE.0.0)) SMALL=RHIE(I,J)
      IF ((RHOE(I,J).LT.SMALL).AND.(RHOE(I,J).NE.0.0)) SMALL=RHOE(I,J)
      IF ((RNIE(I,J).LT.SMALL).AND.(RNIE(I,J).NE.0.0)) SMALL=RNIE(I,J)
      IF ((RNOE(I,J).LT.SMALL).AND.(RNOE(I,J).NE.0.0)) SMALL=RNOE(I,J)
1410  CONTINUE
      KK=NODEI(I,1)
      DO 1420 J=1,KK
      IF ((PNE(I,J).LT.SMALL).AND.(PNE(I,J).NE.0.0)) SMALL=PNE(I,J)
1420  CONTINUE
      DO 1440 J=1,KK
      IF ((BRP(I,J).LT.SMALL).AND.(BRP(I,J).NE.0.0)) SMALL=BRP(I,J)
1440  CONTINUE
      KK=NODEI(I,2)
      DO 1430 J=1,KK
      IF ((MNE(I,J).LT.SMALL).AND.(MNE(I,J).NE.0.0)) SMALL=MNE(I,J)
1430  CONTINUE
      DO 1450 J=1,KK
      IF ((BRM(I,J).LT.SMALL).AND.(BRM(I,J).NE.0.0)) SMALL=BRM(I,J)
1450  CONTINUE
1400  CONTINUE
      IF ((MESS.LT.SMALL).AND.(MESS.NE.0.0)) SMALL=MESS
      IF ((NCHECK.LT.SMALL).AND.(NCHECK.NE.0.0)) SMALL=NCHECK
      IF ((NHARD.LT.SMALL).AND.(NHARD.NE.0.0)) SMALL=NHARD
      IF (SMALL.EQ.32767.0) GOTO 8000
      TIME=TIME+SMALL
      IF (NPKT.GE.490) GOTO 8005
```

```
C     RELEASE PROCESSORS FINISHED WITH

3060  DO 4020 I=1,N
      L=NODEI(I,1)
      DO 4020 J=1,L
      IF (PNE(I,J).EQ.0.0) GOTO 4020
      PNE(I,J)=PNE(I,J)-SMALL
      IF (PNE(I,J).NE.0.0) GOTO 4020
      DO 4030 K=1,10
      IF (PFL(I,K).EQ.0) GOTO 4040
4030  CONTINUE
      GOTO 4020
4040  PFL(I,K)=J
4020  CONTINUE

C     RELEASE MEMORY UNITS FINISHED WITH

      DO 4010 I=1,N
      L=NODEI(I,2)
      DO 4010 J=1,L
      IF (MNE(I,J).EQ.0.0) GOTO 4010
      MNE(I,J)=MNE(I,J)-SMALL
      MUSE=MUSE+SMALL
      IF (MNE(I,J).NE.0.0) GOTO 4010
      MFL(I,J)=1
C     MEMORY UNIT FREE AND JOINS FREE LIST
4010  CONTINUE
      LL=0
      DO 4050 I=1,N
      IF (PFL(I,1).EQ.0) GOTO 4050
      LL=1
4050  CONTINUE

C     PROCESSOR FAILURE

      KK=0
      DO 4031 I=1,N
      L=NODEI(I,1)
      DO 4031 J=1,L
      IF (BRP(I,J).EQ.0.0) GOTO 4031
      BRP(I,J)=BRP(I,J)-SMALL
      IF (BRP(I,J).NE.0.0) GOTO 4031
      WRITE (2,1212) I,J
1212  FORMAT(' NODE ',I3,' PROCESSOR ',I3,' FAILED')
      KK=1
      IF (PNE(I,J).NE.0.0) GOTO 4035
      DO 4032 K=1,L
      IF (PFL(I,K).EQ.J) GOTO 4033
4032  CONTINUE
4033  DO 4034 K=K,9
4034  PFL(I,K)=PFL(I,K+1)
4035  PNE(I,J)=100.0
4037  NOPR=NOPR-1.0
4031  CONTINUE
      IF (KK.EQ.0) GOTO 4038
      CALL R10(10,TPROC)
```

```
C     MEMORY MODULE FAILURE
4038  DO 4101 I=1,N
      L=NODEI(I,2)
      DO 4101 J=1,L
      IF (BRM(I,J).EQ.0.0) GOTO 4101
      BRM(I,J)=BRM(I,J)-SMALL
      IF (BRM(I,J).NE.0.0) GOTO 4101
      WRITE (2,1213) I,J
1213  FORMAT(' NODE ',I3,' MEMORY MODULE ',I3,' FAILED')
      MFL(I,J)=0
      MNE(I,J)=100.0
      LLL=MEMS(I,J)
      MEMS(I,J)=NODEI(I,3)
4106  IF (LLL.EQ.0) GOTO 4102
      KKK=NBUFI(I,2)
      DO 4103 KK=1,KKK
      IF (NMEM(I,KK).EQ.J) GOTO 4104
4103  CONTINUE
      GOTO 4107
4104  DO 4105 KK=KK,KKK-1
      NMEM(I,KK)=NMEM(I,KK+1)
4105  NBUFF(I,KK)=NBUFF(I,KK+1)
      LLL=LLL-1
      NBUFI(I,2)=NBUFI(I,2)-1
      GOTO 4106
4107  KKK=NBUFI(I,3)
      IF (LLL.EQ.0) GOTO 4102
      DO 4108 KK=1,KKK
      KJ=51-KK
      IF (NMEM(I,KJ).EQ.J) GOTO 4109
4108  CONTINUE
4109  DO 4111 KK=KK,KKK-1
      KJ=51-KK
      NMEM(I,KJ)=NMEM(I,KJ-1)
4111  NBUFF(I,KJ)=NBUFF(I,KJ-1)
      LLL=LLL-1
      NBUFI(I,3)=NBUFI(I,3)-1
      GOTO 4107
4102  NOMEM=NOMEM-1
4101  CONTINUE
      MESS=MESS-SMALL
      IF (MESS.GT.0.0) GOTO 4115
```

```
C     GENERATE MESSAGES
C     *****************
C     GENERATE NO OF HOSTS WITH NEW MESSAGES
3050  CALL GEN(CH1,CH2,CH3,CH4,TOTAL)
      IF (TOTAL.GT.MAXH) GOTO 3050
      DO 5250 NN=1,NH
      IF (TOTAL.LE.HCF(NN)) GOTO 5100
5250  CONTINUE
C     FEW HOSTS GENERATING MESSAGES
5100  CALL GEN(CA1,CA2,CA3,CA4,TOTAL)
      IF (TOTAL.GT.32383) GOTO 5100
      CALL CHOP(CUM,TOTAL,CLASS)
      RC=CLASS
      MESS=RC*FIA
C     MESS HOLDS TIME OF NEXT MESSAGE INPUT
      IF (NN.EQ.0) GOTO 3935
C     GENERATE NN HOST NOS WITH MESSAGES
      DO 3920 I=1,NH
3920  CHQ(I)=I
      KH=NH
      RH=NH
      DO 3900 I=1,5
      DO 3900 J=1,5
3900  MG(I,J)=0
      JJJ=0
3910  CALL RAND(A,B,SUM)
      R=SUM
      R=R*0.0009765625
      HC=IDINT(R*(RH-1.0)+1.5)
      NMG=CHQ(HC)
      CALL SUBS(NMG,H,II,JJ)
      KKK=KH-1
      DO 3936 I=HC,KKK
3936  CHQ(I)=CHQ(I+1)
      KH=KH-1
      RH=KH
      JJJ=JJJ+1
      CALL RAND(A,B,SUM)
      R=SUM
      R=R*0.0009765625
      IF (R.LT.HNU(II,JJ)) GOTO 3945
      MG(II,JJ)=1
3945  IF (JJJ.LT.NN) GOTO 3910
      DO 3930 I=1,N
      HH=H(I)
      DO 3930 J=1,HH
      IF (MG(I,J).NE.1) GOTO 3930
      IF (HQL(I,J).LT.32767) GOTO 3940
      REJ(I,J)=REJ(I,J)+1
      GOTO 3930
3940  HQL(I,J)=HQL(I,J)+1
C     ONE MORE MESSAGE IN QUEUE
3930  CONTINUE

3935  IF (TIME.EQ.0.0) GOTO 4065

C     IF NO PROC FREE HANDLE OBS,HI,UHO - NO PROC REQUIRED
```

```
C     HANDLE NODE OUTPUT BUFFER
C     *************************

4115  DO 2070 I=1,N
      L=LK(I)
      DO 2070 J=1,L
      IF (NOE(I,J).EQ.0.0) GOTO 2070
C     ANY PACKETS TO TRANSMIT
      NOE(I,J)=NOE(I,J)-SMALL
      IF (NOE(I,J).NE.0.0) GOTO 2070
C     THIS PACKET NOW READY FOR TRANSMISSION
      II=NET(I,J)
      JJ=NETB(I,J)
      IF (NIE(II,JJ).EQ.0.0) GOTO 2073
      NOE(I,J)=NIE(II,JJ)
      GOTO 2070
2073  NIB(II,JJ)=NOB(I,J)
      PACK=NOB(I,J)
      TYPE=PKT(PACK,3)
C     ONLY RETRANSMIT MESSAGE PACKET
      IF (TYPE.NE.0) GOTO 2072
      RNOE(I,J)=0.1
      RNOB(I,J)=NOB(I,J)
      NOB(I,J)=0
C     AUTOMATIC RETRANSMISSION OF MESSAGE IF ACKNOWLEDGEMENT
C     NOT RECEIVED WITHIN 0.1 SECONDS
      GOTO 2071
2072  NOB(I,J)=0
C     CLEAR BUFFER WHICH CONTAINS CONTROL MESSAGE
2071  NIE(II,JJ)=SMALL+0.000001
2070  CONTINUE
```

```
C      HANDLE HOST INPUT BUFFERS
C      ***************************

       DO 7010 I=1,N
       LJ=H(I)
       DO 7010 J=1,LJ
       IF (HNIE(I,J).EQ.0.0) GOTO 7010
       HNIE(I,J)=HNIE(I,J)-SMALL
       IF (HNIE(I,J).NE.0.0) GOTO 7010
       PACK=HIB(I,J)
       TYPE=PKT(PACK,3)
       IF (TYPE.EQ.4) GOTO 7015
       IF (TYPE.NE.3) GOTO 7030
 7015  CALL TRANS(PKT,TPIN,PACK,VEC,NREC,TIME,
      XPTY,TPT,TIN,ML,TMT,SNC,RNC)
C      RECORD PKT STATISTICS AND DELETE PKT FROM SYSTEM
       NPKT=NPKT-1
       HIB(I,J)=0
       HNIE(I,J)=0.0
       IF (TYPE.EQ.3) GOTO 7040
       GOTO 7010
 7030  IF (TYPE.NE.0) GOTO 7040
       IF (NPKT.LT.500) GOTO 7005
 7034  HNIE(I,J)=0.0000001
       GOTO 7010
 7005  IF (HCBI(I,J,2).LT.HCBI(I,J,1)) GOTO 7035
C      OUTPUT BUFFER FULL
 7032  HNIE(I,J)=HNOE(I,J)
       IF (HNIE(I,J).EQ.0.0) GOTO 7034
       GOTO 7010
 7035  RHIE(I,J)=0.1
       RHIB(I,J)=HIB(I,J)
       HIB(I,J)=0
C      AUTOMATIC RETRANSMISSION OF MESSAGE PACKET
       DO 7080 II=1,500
       IF (PKT(II,1).EQ.0) GOTO 7095
 7080  CONTINUE

C      ACKNOWLEDGE MESSAGE PACKET

 7095  PKT(II,1)=PKT(PACK,2)+1000
       PKT(II,2)=I+100
       PKT(II,3)=2
       TPIN(II,1)=TIME
       HCBI(I,J,2)=HCBI(I,J,2)+1
       K=HCBI(I,J,2)
       HCB(I,J,K)=II
C      SEARCH PKT FOR VACANT VECTOR
       DO 7090 JJ=1,500
       IF (PKT(JJ,1).EQ.0) GOTO 7091
 7090  CONTINUE
```

```
C     INFORM SOURCE TO SEND NEXT PACKET

7091  PKT(JJ,1)=PKT(PACK,2)
      PKT(JJ,2)=PKT(PACK,1)
      PKT(JJ,3)=3
      TPIN(JJ,1)=TIME
      HCBI(I,J,2)=HCBI(I,J,2)+1
      K=HCBI(I,J,2)
      HCB(I,J,K)=JJ
      NPKT=NPKT+2
C     TWO MORE PACKETS IN SYSTEM
      TPIN(PACK,2)=TIME
C     REAL THRUPUT TIME OF MESSAGE PACKET
      GOTO 7010

C     SEND NEXT PACKET

7040  PACK=RHOB(I,J)
      RHOB(I,J)=0
      RHOE(I,J)=0.0
      IF (HPBI(I,J,2).GE.(HPBI(I,J,1)-1)) GOTO 7032
      CALL TRANS(PKT,TPIN,PACK,VEC,NREC,TIME,
     XPTY,TPT,TIN,ML,TMT,SNC,RNC)
      NPKT=NPKT-1
      CALL ISUBS(NMG,H,I,J)
      HPSN(I,J)=HPSN(I,J)+1
      IF (HPSN(I,J).LE.HPM(I,J)) GOTO 7085
      IF (HQL(I,J).EQ.0) GOTO 7010
      HQL(I,J)=HQL(I,J)-1
C     GENERATE NEW MESSAGE INFORMATION
      HPSN(I,J)=1
C     GENERATE MESSAGE DESTINATION
7096  CALL RAND(A,B,SUM)
      R=SUM
      R=R*0.0009765625
      L=IDINT(R*(HN-1.0)+1.5)
      IF (L.EQ.NMG) GOTO 7096
      HMD(I,J)=L
C     GENERATE MESS OF MEAN LENGTH 1
7098  CALL GEN(CP1,CP2,CP3,CP4,TOTAL)
      IF (TOTAL.GT.32383) GOTO 7098
      CALL CHOP(CUM,TOTAL,CLASS)
      RC=CLASS
C     IF R <= MMR GENERATE SHORT MESSAGE
C     ELSE GENERATE LONG MESSAGE
      CALL RAND(A,B,SUM)
      R=SUM
      R=R*0.0009765625
      IF (R.GT.MMR) GOTO 7088
C     SCALE MESS OF MEAN 1 TO SHORT MESS MEAN MSM
      L=IDINT(RC*FSM+0.999)
      GOTO 7089
C     SCALE MESS OF MEAN 1 TO LONG MESS MEAN MLM
7088  L=IDINT(RC*FLM+0.999)
```

```
7089  HPM(I,J)=L
7085  PKT(PACK,1)=NMG
      PKT(PACK,2)=HMD(I,J)
      PKT(PACK,3)=0
      PKT(PACK,4)=HPSN(I,J)
      PKT(PACK,5)=HPM(I,J)
      TPIN(PACK,1)=TIME
      NPKT=NPKT+1
      HPBI(I,J,2)=HPBI(I,J,2)+1
      K=HPBI(I,J,2)
      HPB(I,J,K)=PACK
7010  CONTINUE
      IF (LL.EQ.1) GOTO 4055

C     UPDATE EVENT TABLES WHICH REQUIRE PROCESSORS

      DO 4210 I=1,N
      L=H(I)
      DO 4210 J=1,L
      IF (HNOE(I,J).EQ.0.0) GOTO 4210
      IF (HNOE(I,J).GT.SMALL) GOTO 4215
C     TRY AGAIN WHEN NEXT PROCESSOR RELEASED
      CALL NPR(I,NODEI(I,1),PNE,HNOE(I,J))
      GOTO 4210
4215  HNOE(I,J)=HNOE(I,J)-SMALL
4210  CONTINUE
      DO 4220 I=1,N
      L=LK(I)
      DO 4220 J=1,L
      IF (NIE(I,J).EQ.0.0) GOTO 4220
      IF (NIE(I,J).GT.SMALL) GOTO 4225
      CALL NPR(I,NODEI(I,1),PNE,NIE(I,J))
      GOTO 4220
4225  NIE(I,J)=NIE(I,J)-SMALL
4220  CONTINUE
      GOTO 4065
4055  DO 5999 I=1,N
      UPRP(1)=PORD(I,1)
      UPRP(2)=PORD(I,2)
      UPRP(3)=PORD(I,3)
      UPRP(4)=PORD(I,4)
      NPROC=0
      PNO=0
C     NPROC IS THE NEXT PROCESS NUMBER
      CALL PROC(I,PORD,UPRP,NPROC,PFL(I,1),PNO)
      GOTO (1999,2999,3999,4999,5999) , NPROC
```

```
C     INPUT PACKETS FROM HOSTS INTO NETWORK
C     *************************************
1999  OCC=0.0
      LJ=H(I)
      IF (PFL(I,1).NE.0) GOTO 4135
      DO 4136 J=1,LJ
      IF (HNOE(I,J).EQ.0.0) GOTO 4136
      IF (HNOE(I,J).GT.SMALL) GOTO 4138
      CALL NPR(I,NODEI(I,1),PNE,HNOE(I,J))
      GOTO 4136
4138  HNOE(I,J)=HNOE(I,J)-SMALL
4136  CONTINUE
      GOTO 4131
4135  DO 7500 J=1,LJ
      IF (HNOE(I,J).EQ.0.0) GOTO 7500
      HNOE(I,J)=HNOE(I,J)-SMALL
      IF (HNOE(I,J).NE.0.0) GOTO 7500
      K=HOB(I,J)
      TYPE=PKT(K,3)
      DEST=PKT(K,2)
      IF (TYPE.NE.2) GOTO 7510
C     NODE - HOST ACK
      CALL TRANS(PKT,TPIN,K,VEC,NREC,TIME,
     XPTY,TPT,TIN,ML,TMT,SNC,RNC)
      NPKT=NPKT-1
      OCC=OCC+2.0*NSP(I,3)
      HOB(I,J)=0
      RHIB(I,J)=0
      RHIE(I,J)=0.0
C     RETRANSMISSION NOT REQUIRED
      GOTO 7500
C     CAN NOW TRANSFER PKT READY FOR NETWORK TRANSMISSION
7510  IF ((CBUFI(I,2)+CBUFI(I,3)).LT.CBUFI(I,1)) GOTO 7520
C     SYSTEM FULL TRY AGAIN IN 0.0000001 SECS
7540  HNOE(I,J)=0.0000001
      GOTO 7500
C     HANDLE SNM PKT
7520  IF (TYPE.NE.3) GOTO 7530
      CALL SUBS(DEST,H,ILK,JLK)
      IF (ILK.NE.I) GOTO 7521
C     PKT DESTINATION AT SAME NODE
      CBUFI(I,2)=CBUFI(I,2)+1
      K=CBUFI(I,2)
      GOTO 7522
7521  CBUFI(I,3)=CBUFI(I,3)+1
      K=51-CBUFI(I,3)
7522  CBUFF(I,K)=HOB(I,J)
      HOB(I,J)=0
      OCC=OCC+2.0*NSP(I,3)
      GOTO 7500
7530  IF ((NBUFI(I,2)+NBUFI(I,3)).GE.NBUFI(I,1)) GOTO 7540
      IF (NPKT.GE.499) GOTO 7540
      KK=NODEI(I,2)
      DO 7550 M=1,KK
C     IS MEMORY UNIT M FREE
      IF (MFL(I,M).EQ.0) GOTO 7550
C     IS THERE ROOM IN M TO STORE ONE MORE PACKET
      IF (MEMS(I,M).LT.NODEI(I,3)) GOTO 7560
7550  CONTINUE
C     TRY AGAIN WHEN NEXT MEMORY UNIT FREE
      CALL NPR(I,NODEI(I,2),MNE,HNOE(I,J))
      WRITE (2,7551)
7551  FORMAT(' INPUT PKTS INTO NETWORK - NO MEMORY AVAILABLE')
      GOTO 7500
C     NO MEMORY UNIT CAN STORE PACKET
C     SET MEMORY M BUSY
7560  MFL(I,M)=0
C     NEXT MEMORY EVENT IN TIME TO STORE A PACKET
      MNE(I,M)=NSP(I,2)
C     INCREMENT NO OF PKTS IN MEM UNIT M BY 1
      MEMS(I,M)=MEMS(I,M)+1
      CALL SUBS(DEST,H,ILK,JLK)
      IF (ILK.NE.I) GOTO 7581
C     PKT DEST AT SAME NODE
      NBUFI(I,2)=NBUFI(I,2)+1
      L=NBUFI(I,2)
      GOTO 7582
7581  NBUFI(I,3)=NBUFI(I,3)+1
C     PUT PACKET INTO NODE OUTPUT BUFFER
      L=51-NBUFI(I,3)
7582  NBUFF(I,L)=HOB(I,J)
      NMEM(I,L)=M
C     MEMORY UNIT PACKET STORED IN
      RHOE(I,J)=0.1
      RHOB(I,J)=HOB(I,J)
      HOB(I,J)=0
C     AUTOMATIC RETRANSMISSION WITHIN 0.5 SECS
C     ACK HOST - NODE PACKET TRANSFER
      DO 7570 II=1,500
      IF (PKT(II,1).EQ.0) GOTO 7580
7570  CONTINUE
7580  PKT(II,1)=I+100
      PKT(II,2)=PKT(K,1)
      PKT(II,3)=4
      TPIN(II,1)=TIME
      NPKT=NPKT+1
      CBUFI(I,2)=CBUFI(I,2)+1
      K=CBUFI(I,2)
      CBUFF(I,K)=II
      OCC=OCC+2.0*NSP(I,2)+2.0*NSP(I,3)
7500  CONTINUE
      IF (OCC.EQ.0.0) GOTO 4131
      CALL ALLOC(I,PFL,PNE,P,OCC,TPROC)
4131  CONTINUE
      CALL PROC(I,PORD,UPRP,NPROC,PFL(I,1),PNO)
      GOTO (1999,2999,3999,4999,5999) , NPROC
```

```
C     HANDLE NODE INPUT BUFFERS
C     *************************
2999  OCC=0.0
      L=LK(I)
      IF (PFL(I,1).NE.0) GOTO 4145
      DO 4146 J=1,L
      IF (NIE(I,J).EQ.0.0) GOTO 4146
      IF (NIE(I,J).GT.SMALL) GOTO 4148
C     TRY AGAIN WHEN NEXT PROCESSOR RELEASED
      CALL NPR(I,NODEI(I,1),PNE,NIE(I,J))
      GOTO 4146
4148  NIE(I,J)=NIE(I,J)-SMALL
4146  CONTINUE
      GOTO 4141
4145  DO 6020 J=1,L
      IF (NIE(I,J).EQ.0.0) GOTO 6020
C     ANY O/P - I/P
      NIE(I,J)=NIE(I,J)-SMALL
      IF (NIE(I,J).NE.0.0) GOTO 6020
      PACK=NIB(I,J)
C     PKT NO BEING HANDLED
      PD=PKT(PACK,2)
C     PACKET DESTINATION
      TYPE=PKT(PACK,3)
C     TYPE OF PACKET
      IB=PKT(PACK,4)
C     PKT NO WHICH IS BEING ACK
      IF (TYPE.NE.1) GOTO 6030
C     LOCAL CNT - NODE TO NODE TRANSFER
      CALL TRANS(PKT,TPIN,PACK,VEC,NREC,TIME,
     XPTY,TPT,TIN,ML,TMT,SNC,RNC)
C     RECORD PKT STATISTICS AND DELETE PKT FROM SYSTEM
      DO 6055 JJ=1,5
      IF (RNOB(I,JJ).EQ.IB) GOTO 6045
6055  CONTINUE
6045  RNOB(I,JJ)=0
      RNOE(I,JJ)=0.0
      NIB(I,J)=0
      NPKT=NPKT-1
C     RECORD TIME TO HANDLE PROCESS
      OCC=OCC+2.0*NSP(I,2)
      GOTO 6020
6030  IF (NPKT.LT.500) GOTO 6135
6130  NIE(I,J)=0.0000001
      GOTO 6020
6135  IF ((CBUFI(I,2)+CBUFI(I,3)).GE.(CBUFI(I,1)-1)) GOTO 6130
      IF (TYPE.NE.3) GOTO 6140
      OCC=OCC+2.0*NSP(I,3)
      NIB(I,J)=0
      CALL SUBS(PD,H,II,JJ)
      IF (II.EQ.I) GOTO 6025
```

```
C     CNT PACKET NOT AT DESTINATION

      CBUFI(I,3)=CBUFI(I,3)+1
      K=51-CBUFI(I,3)
      CBUFF(I,K)=PACK
      GOTO 6020

C     PACKET LEFT IS ONE AT DESTINATION

6025  CBUFI(I,2)=CBUFI(I,2)+1
      K=CBUFI(I,2)
      CBUFF(I,K)=PACK
      GOTO 6020
6140  IF ((NBUFI(I,2)+NBUFI(I,3)).GE.(NBUFI(I,1))) GOTO 6130
      KK=NODEI(I,2)
      DO 6145 M=1,KK
C     IS MEMORY UNIT FREE
      IF (MFL(I,M).EQ.0) GOTO 6145
C     IS THERE ROOM TO STORE ANOTHER PACKET
      IF (MEMS(I,M).LT.NODEI(I,3)) GOTO 6165
6145  CONTINUE
C     TRY AGAIN WHEN NEXT MEMORY UNIT RELEASED
      CALL NPR(I,NODEI(I,2),MNE,NIE(I,J))
      WRITE (2,6113)
6113  FORMAT(' NODE INPUT - NO MEMORY AVAILABLE')
      GOTO 6020
C     NO MEMORY UNIT CAN STORE A PACKET
C     SET MEMORY UNIT M BUSY
6165  MFL(I,M)=0
C     NEXT MEMORY EVENT IN TIME IT TAKES
C     TO STORE A MESSAGE PACKET
      MNE(I,M)=NSP(I,2)
      MEMS(I,M)=MEMS(I,M)+1
C     INCREMENT NO OF PKTS HELD IN MEMORY UNIT M
C     FIND FREE SLOT FOR ANOTHER PKT
      DO 6040 II=1,500
      IF (PKT(II,1).EQ.0) GOTO 6050
6040  CONTINUE

C     ACK PKT RECEIVED

6050  PKT(II,1)=I+100
      KK=SNET(I,J)
      PKT(II,2)=KK+100
      PKT(II,3)=1
      PKT(II,4)=PACK
C     KNOW WHICH PACKET WE ARE ACK
      TPIN(II,1)=TIME
      CBUFI(I,3)=CBUFI(I,3)+1
      K=51-CBUFI(I,3)
      CBUFF(I,K)=II
      NPKT=NPKT+1
      NIB(I,J)=0
      CALL SUBS(PD,H,II,JJ)
      IF (II.EQ.I) GOTO 6060
```

```
C     PACKET NOT AT DESTINATION
      NBUFI(I,3)=NBUFI(I,3)+1
      K=51-NBUFI(I,3)
      NBUFF(I,K)=PACK
      NMEM(I,K)=M
      GOTO 6020
C     PACKET LEFT IS ONE AT DESTINATION
6060  NBUFI(I,2)=NBUFI(I,2)+1
      K=NBUFI(I,2)
      NBUFF(I,K)=PACK
      NMEM(I,K)=M
C     RECORD TIME TO HANDLE A PACKET AND CONTROL PACKET
      OCC=OCC+2.0*NSP(I,2)+2.0*NSP(I,3)
6020  CONTINUE
      IF (OCC.EQ.0.0) GOTO 4141
      CALL ALLOC(I,PFL,PNE,P,OCC,TPROC)
4141  CONTINUE
      CALL PROC(I,PORD,UPRP,NPROC,PFL(I,1),PNO)
      GOTO (1999,2999,3999,4999,5999) , NPROC
```

```
C     UPDATE HOST INPUT BUFFERS
C     *************************
3999  OCC=0.0
      LJ=H(I)
      IF (PFL(I,1).NE.0) GOTO 4165
      DO 4166 J=1,LJ
      IF (HNIE(I,J).EQ.0.0) GOTO 4166
      IF (HNIE(I,J).GT.SMALL) GOTO 4168
      CALL NPR(I,NODEI(I,1),PNE,HNIE(I,J))
      GOTO 4166
4168  HNIE(I,J)=HNIE(I,J)-SMALL
4166  CONTINUE
      DO 5600 J=1,LJ
      IF (RHIE(I,J).EQ.0.0) GOTO 5600
      IF (RHIE(I,J).GT.SMALL) GOTO 5610
      CALL NPR(I,NODEI(I,1),PNE,RHIE(I,J))
      GOTO 5600
5610  RHIE(I,J)=RHIE(I,J)-SMALL
5600  CONTINUE
      GOTO 4161
4165  DO 7780 J=1,LJ
C     UPDATE RETRANSMISSION TIME
      RP=0
      IF (RHIB(I,J).EQ.0) GOTO 7700
      RHIE(I,J)=RHIE(I,J)-SMALL
      IF (RHIE(I,J).GT.0.0) GOTO 7700
      RP=1
7700  IF (HNIE(I,J).NE.0.0) GOTO 7710
C     PACKET STILL WAITING FOR TRANSMISSION
      IF (HIB(I,J).NE.0) GOTO 7710
C     IF ZERO BUFFER FREE,ELSE WAITING FOR ACKNOWLEDGEMENT
      CALL ISUBS(NMG,H,I,J)
C     CHECK IF ANY CNT MESSAGES FOR HOST(I,J)
      IF (CBUFI(I,2).EQ.0) GOTO 7720
C     NO CONTROL PACKETS TO TRANSMIT
C     SEARCH CBUFF I/P FOR PKT WHOSE DEST IS HOST(I,J)
      LLL=CBUFI(I,2)
      DO 7740 L=1,LLL
      PACK=CBUFF(I,L)
      DEST=PKT(PACK,2)
      IF (DEST.EQ.NMG) GOTO 7730
7740  CONTINUE
      GOTO 7720
7730  HIB(I,J)=CBUFF(I,L)
      HNIE(I,J)=HCLR(I,J)
      CBUFI(I,2)=CBUFI(I,2)-1
C     ONE LESS PACKET IN BUFFER
C     NOW SHIFT QUEUE UP
      LLL=CBUFI(I,2)
      DO 7750 L=L,LLL
7750  CBUFF(I,L)=CBUFF(I,L+1)
      OCC=OCC+2.0*NSP(I,3)
7710  IF (RP.EQ.0) GOTO 7780
      IF (HNIE(I,J).EQ.0.0) GOTO 7760
      RHIE(I,J)=HNIE(I,J)
      GOTO 7780
7760  RHIE(I,J)=0.0000001
      GOTO 7780
```

```
C       NOW DEAL WITH FULL LENGTH PACKETS
7720    IF (RP.EQ.0) GOTO 7770
C       RETRANSMISSION OF PACKET REQUIRED
        HIB(I,J)=RHIB(I,J)
        HNIE(I,J)=HPLR(I,J)
        RHIE(I,J)=0.1
        OCC=OCC+2.0*NSP(I,2)
        GOTO 7780
7770    IF (NBUFI(I,2).EQ.0) GOTO 7780
C       NO MORE PACKETS TO TRANSMIT
C       SEARCH NBUFF I/P FOR PACKET WHOSE DEST IS HOST(I,J)
        LLL=NBUFI(I,2)
        DO 7785 L=1,LLL
        PACK=NBUFF(I,L)
        DEST=PKT(PACK,2)
        IF (DEST.NE.NMG) GOTO 7785
        MEM=NMEM(I,L)
C       CHECK NOW IF MEMORY UNIT FREE
        IF (MFL(I,MEM).EQ.1) GOTO 7790
7785    CONTINUE
        GOTO 7780
7790    HIB(I,J)=NBUFF(I,L)
        HNIE(I,J)=HPLR(I,J)
        NBUFI(I,2)=NBUFI(I,2)-1
C       ONE LESS PACKET IN BUFFER
C       NOW SHIFT BUFFER QUEUE UP
        LLL=NBUFI(I,2)
        DO 7795 L=L,LLL
        NBUFF(I,L)=NBUFF(I,L+1)
7795    NMEM(I,L)=NMEM(I,L+1)
        MFL(I,MEM)=0
C       MEMORY UNIT SET BUSY
        MNE(I,MEM)=NSP(I,2)
C       MEMORY UNIT OCCUPIED FOR TIME REQD TO EXTRACT PKT
        MEMS(I,MEM)=MEMS(I,MEM)-1
C       ONE LESS PKT IN MEMORY UNIT
        OCC=OCC+2.0*NSP(I,2)
7780    CONTINUE
        IF (OCC.EQ.0.0) GOTO 4161
        CALL ALLOC(I,PFL,PNE,P,OCC,TPROC)
4161    CONTINUE
        CALL PROC(I,PORD,UPRP,NPROC,PFL(I,1),PNO)
        GOTO (1999,2999,3999,4999,5999) , NPROC
```

```
C     UPDATE NODE OUTPUT BUFFERS
C     **************************
4999  OCC=0.0
      LJ=LK(I)
      IF (PFL(I,1).NE.0) GOTO 4175
      DO 4176 J=1,LJ
      IF (NOE(I,J).EQ.0.0) GOTO 4176
      IF (NOE(I,J).GT.SMALL) GOTO 4178
      CALL NPR(I,NODEI(I,1),PNE,NOE(I,J))
      GOTO 4176
4178  NOE(I,J)=NOE(I,J)-SMALL
4176  CONTINUE
      DO 5650 J=1,LJ
      IF (RNOE(I,J).EQ.0.0) GOTO 5650
      IF (RNOE(I,J).GT.SMALL) GOTO 5660
      CALL NPR(I,NODEI(I,1),PNE,RNOE(I,J))
      GOTO 5650
5660  RNOE(I,J)=RNOE(I,J)-SMALL
5650  CONTINUE
      GOTO 4171
4175  DO 6051 J=1,5
6051  AVAIL(J)=0
      AA=0
      K=LK(I)
      DO 6110 J=1,K
      IF (NOE(I,J).NE.0.0) GOTO 6110
C     PACKET STILL WAITING FOR TRANSMISSION
      IF (NOB(I,J).NE.0) GOTO 6110
C     IF ZERO BUFFER FREE, ELSE PACKET HELD AWAITING ACK
      AVAIL(J)=1
      AA=AA+1
6110  CONTINUE
C     AVAIL HOLDS BUFFERS AVAILABLE FOR USE
C     UPDATE TRANSMISSION TIME
      DO 6000 J=1,K
      IF (RNOB(I,J).EQ.0) GOTO 6000
      RNOE(I,J)=RNOE(I,J)-SMALL
      IF (RNOE(I,J).GT.0.0) GOTO 6000
      IF (AVAIL(J).NE.0) GOTO 6100
      RNOE(I,J)=NOE(I,J)
      GOTO 6000
6100  AVAIL(J)=0
      AA=AA-1
      NOB(I,J)=RNOB(I,J)
      NOE(I,J)=NPLR(I,J)
      RNOE(I,J)=0.1
      OCC=OCC+2.0*NSP(I,2)
6000  CONTINUE
      IF (CBUFI(I,3).EQ.0) GOTO 6410
C     ANY CONTROL PACKETS TO OUTPUT
      BBB=AA
      QL=CBUFI(I,3)
      NOP=50
```

```
6326  NTON=0
      IF ((51-CBUFI(I,3)).GT.NOP) GOTO 6400
      IF (AA.EQ.0) GOTO 6400
C     NO BUFFER FREE
      PACK=CBUFF(I,NOP)
      PACKS=PKT(PACK,1)
      IF (PACKS.LT.100) GOTO 6080
      SOUR=PACKS-100
C     NTON=1 INDICATES THAT PACKET REQUIRES DIRECT ROUTE
      NTON=1
      DEST=PKT(PACK,2)-100
      GOTO 6090
6080  IF (PACKS.LT.100) GOTO 6700
      SOUR=PACKS-100
      GOTO 6710
6700  CALL SUBS(PACKS,H,SOUR,JJ)
6710  PACKD=PKT(PACK,2)
      IF (PACKD.LT.100) GOTO 6720
      DEST=PACKD-100
      GOTO 6090
6720  CALL SUBS(PACKD,H,DEST,JJ)
C     SOUR=PKT NODE SOURCE,DEST=PACKET DEST NODE
6090  J=1
6093  IF (AVAIL(J).EQ.0) GOTO 6300
      KK=NET(I,J)
C     KK=DEST NODE
      IF (DEST.NE.KK) GOTO 6300
C     PKT FOUND FOR NODE KK
6350  AA=AA-1
      AVAIL(J)=0
      NOB(I,J)=PACK
      NOE(I,J)=NCLR(I,J)
      CBUFI(I,3)=CBUFI(I,3)-1
C     ONE LESS PACKET IN O/P BUFFER
C     NOW SHIFT QUEUE UP
      KK=CBUFI(I,3)-(50-NOP)
      LLL=NOP
      DO 6270 II=1,KK
      CBUFF(I,LLL)=CBUFF(I,LLL-1)
6270  LLL=LLL-1
      GOTO 6326
6300  J=J+1
      IF (J.LE.K) GOTO 6093
C     NO DIRECT NODE FOUND
      IF (NTON.EQ.1) GOTO 6345
C     NODE TO NODE ACK REQUIRES DIRECT ROUTE
      DO 6340 J=1,K
6340  BFL(J)=0
      IN=0
      DO 6342 J=1,K
      IF (AVAIL(J).EQ.0) GOTO 6342
      IF (NET(I,J).EQ.SOUR) GOTO 6342
C     PKT NOT REROUTED BACK THRU SOURCE NODE
      IN=IN+1
      BFL(IN)=J
6342  CONTINUE
```

```
C     NO BUFFER AVAILABLE FOR CURRENT PKT
      IF (IN.EQ.0) GOTO 6345
      IF (IN.GT.1) GOTO 6343
C     ONLY ONE BUFFER AVAILABLE
      J=BFL(1)
      GOTO 6350
C     SELECT RANDOM OUTPUT BUFFER
6343  CALL RAND(A,B,SUM)
      R=SUM
      R=R*0.0009765625
      IF (R.LE.0.0001) GOTO 6343
      RIN=IN
      JJJ=INT(R*RIN+0.9999)
      J=BFL(JJJ)
C     EXTRACT REQUIRED BUFFER NO
      GOTO 6350
6345  NOP=NOP-1
      GOTO 6326
6400  AAA=AA
      BBB=BBB-AAA
      IF (BBB.EQ.0.0) GOTO 6410
      OCC=OCC+QL*NSP(I,1)+2.0*BBB*NSP(I,3)
6410  IF (NBUFI(I,3).EQ.0) GOTO 6210
C     THIS NODE HAS NO PACKETS TO TRANSMIT
      QL=NBUFI(I,3)
      BBB=AA
6320  NOP=50
6325  NTON=0
      IF ((51-NBUFI(I,3)).GT.NOP) GOTO 6200
      IF (AA.EQ.0) GOTO 6200
C     NO BUFFER FREE
      MEM=NMEM(I,NOP)
      IF (MFL(I,MEM).NE.1) GOTO 6346
C     IS MEMORY FREE
      PACK=NBUFF(I,NOP)
      PACKS=PKT(PACK,1)
      IF (PACKS.LT.100) GOTO 6081
      SOUR=PACKS-100
C     NTON=1 INDICATES THAT PACKET REQUIRES DIRECT ROUTE
      NTON=1
      DEST=PKT(PACK,2)-100
      GOTO 6091
6081  CALL SUBS(PACKS,H,SOUR,JJ)
      PACKD=PKT(PACK,2)
      CALL SUBS(PACKD,H,DEST,JJ)
C     SOUR=PACKET NODE SOURCE, DEST=PACKET DEST NODE
6091  J=1
6092  IF (AVAIL(J).EQ.0) GOTO 6301
      KK=NET(I,J)
C     KK=DEST NODE
      IF (DEST.NE.KK) GOTO 6301
C     PACKET FOUND FOR NODE KK
6351  AA=AA-1
      AVAIL(J)=0
      NOB(I,J)=PACK
      NOE(I,J)=NPLR(I,J)
      NBUFI(I,3)=NBUFI(I,3)-1
C     ONE LESS PKT IN OUTPUT BUFFER
```

```
C     NOW SHIFT QUEUE UP
      KK=NBUFI(I,3)-(50-NOP)
      LLL=NOP
      DO 6271 II=1,KK
      NBUFF(I,LLL)=NBUFF(I,LLL-1)
      NMEM(I,LLL)=NMEM(I,LLL-1)
6271  LLL=LLL-1
      MFL(I,MEM)=0
C     MEMORY UNIT SET BUSY
      MNE(I,MEM)=NSP(I,2)
C     MEMORY UNIT OCCUPIED FOR TIME REQD TO EXTRACT PACKET
      MEMS(I,MEM)=MEMS(I,MEM)-1
C     ONE LESS PACKET IN MEMORY UNIT
      GOTO 6325
6301  J=J+1
      IF (J.LE.K) GOTO 6092
C     NO DIRECT NODE FOUND
      IF (NTON.EQ.1) GOTO 6346
C     NODE TO NODE ACK REQUIRES DIRECT ROUTE
      DO 6341 J=1,K
6341  BFL(J)=0
      IN=0
      DO 6348 J=1,K
      IF (AVAIL(J).EQ.0) GOTO 6348
      IF (NET(I,J).EQ.SOUR) GOTO 6348
C     PKT NOT REROUTED BACK THRU SOURCE NODE
      IN=IN+1
      BFL(IN)=J
6348  CONTINUE
C     NO BUFFER AVAILABLE FOR CURRENT PKT
      IF (IN.EQ.0) GOTO 6346
      IF (IN.GT.1) GOTO 6349
C     ONLY ONE BUFFER AVAILABLE
      J=BFL(1)
      GOTO 6351
C     SELECT RANDOM OUTPUT BUFFER
6349  CALL RAND(A,B,SUM)
      R=SUM
      R=R*0.0009765625
      IF (R.LE.0.0001) GOTO 6349
      RIN=IN
      JJJ=INT(R*RIN+0.9999)
      J=BFL(JJJ)
C     EXTRACT REQUIRED BUFFER NO
      GOTO 6351
6346  NOP=NOP-1
      GOTO 6325
6200  AAA=AA
      BBB=BBB-AAA
      IF (BBB.EQ.0.0) GOTO 6210
      OCC=OCC+QL*NSP(I,1)+2.0*BBB*NSP(I,2)
6210  CONTINUE
      IF (OCC.EQ.0.0) GOTO 4171
      CALL ALLOC(I,PFL,PNE,P,OCC,TPROC)
4171  CONTINUE
      CALL PROC(I,PORD,UPRP,NPROC,PFL(I,1),PNO)
      GOTO (1999,2999,3999,4999,5999) , NPROC
5999  CONTINUE
```

```
C     BRING PACKETS INTO HOST OUTPUT BUFFERS
C     ************************************
4065  DO 7000 I=1,N
      K=H(I)
      DO 7000 J=1,K
C     UPDATE RETRANSMISSION TIME
      RP=0
      IF (RHOB(I,J).EQ.0) GOTO 7250
      RHOE(I,J)=RHOE(I,J)-SMALL
      IF (RHOE(I,J).GT.0.0) GOTO 7250
      RP=1
7250  IF (HNOE(I,J).NE.0.0) GOTO 7300
C     PACKET STILL WAITING FOR TRANSMISSION
      IF (HOB(I,J).NE.0) GOTO 7300
C     IF ZERO BUFFER FREE, ELSE AWAITING ACK
      IF (HCBI(I,J,2).EQ.0) GOTO 7200
C     ANY CONTROL MESSAGES TO OUTPUT
      HOB(I,J)=HCB(I,J,1)
      HNOE(I,J)=HCLR(I,J)
      HCBI(I,J,2)=HCBI(I,J,2)-1
C     ONE PKT LESS IN BUFFER
C     SHIFT QUEUE UP
      L=HCBI(I,J,2)
      DO 7240 LJ=1,L
7240  HCB(I,J,LJ)=HCB(I,J,LJ+1)
7300  IF (RP.EQ.0) GOTO 7000
      IF (HNOE(I,J).EQ.0.0) GOTO 7140
      RHOE(I,J)=HNOE(I,J)
      GOTO 7000
7140  RHOE(I,J)=0.0000001
      GOTO 7000
C     NO CONTROL PACKETS TO OUTPUT
7200  IF (RP.EQ.0) GOTO 7210
      HOB(I,J)=RHOB(I,J)
      HNOE(I,J)=HPLR(I,J)
      RHOE(I,J)=0.1
C     RETRANSMISSION OF PACKET REQUIRED
      GOTO 7000
7210  IF (RHOE(I,J).NE.0.0) GOTO 7000
      IF (HPBI(I,J,2).NE.0) GOTO 7100
      IF (HQL(I,J).EQ.0) GOTO 7000
C     NO MORE MESSAGES
      IF (NPKT.EQ.500) GOTO 7000
      HQL(I,J)=HQL(I,J)-1
C     INPUT NEW MESSAGE INTO SYSTEM
      DO 7110 II=1,500
      IF (PKT(II,1).EQ.0) GOTO 7120
7110  CONTINUE
7120  CALL ISUBS(NMG,H,I,J)
      PKT(II,1)=NMG
```

```
C     GENERATE MESSAGE DESTINATION
7130  CALL RAND(A,B,SUM)
      R=SUM
      R=R*0.0009765625
      NNN=IDINT(R*(HN-1.0)+1.5)
      IF (NNN.EQ.NMG) GOTO 7130
      HMD(I,J)=NNN
      PKT(II,2)=NNN
      PKT(II,3)=0
      HPSN(I,J)=1
      PKT(II,4)=1
C     GENERATE MESS OF MEAN LENGTH 1
7600  CALL GEN(CP1,CP2,CP3,CP4,TOTAL)
      IF (TOTAL.GT.32383) GOTO 7600
      CALL CHOP(CUM,TOTAL,CLASS)
      RC=CLASS
C     IF R <= MMR GENERATE SHORT MESSAGE
C     ELSE GENERATE LONG MESSAGE
      CALL RAND(A,B,SUM)
      R=SUM
      R=R*0.0009765625
      IF (R.GT.MMR) GOTO 7078
C     SCALE MESS OF MEAN 1 TO SHORT MESS MEAN MSM
      NNN=IDINT(RC*FSM+0.999)
      GOTO 7079
C     SCALE MESS OF MEAN 1 TO LONG MESS MEAN MLM
7078  NNN=IDINT(RC*FLM+0.999)
7079  PKT(II,5)=NNN
      HPM(I,J)=NNN
      TPIN(II,1)=TIME
      NPKT=NPKT+1
C     ONE MORE PACKET IN SYSTEM
      HPBI(I,J,2)=1
      HPB(I,J,1)=II
7100  HOB(I,J)=HPB(I,J,1)
      HNOE(I,J)=HPLR(I,J)
      HPBI(I,J,2)=HPBI(I,J,2)-1
C     ONE LESS PACKET IN BUFFER
C     NOW SHIFT BUFFER QUEUE UP
      L=HPBI(I,J,2)
      DO 7041 LJ=1,L
7041  HPB(I,J,LJ)=HPB(I,J,LJ+1)
7000  CONTINUE
      NCHECK=NCHECK-SMALL
      NHARD=NHARD-SMALL
      IF (NCHECK.NE.0.0) GOTO 3305
```

```
C     OUTPUT CURRENT SYSTEM STATE DATA
      XNREC=NREC
      TNREC=TNREC+XNREC
      MNREC=NREC
      NREC=0
      NCHECK=CHECK
3305  IF (NHARD.NE.0.0) GOTO 3300
      TQL=0
      MINQL=32767
      MAXQL=0
      AVQL=0
      K=0
      DO 3306 I=1,N
      L=H(I)
      DO 3306 J=1,L
      HHH=HQL(I,J)
      IF (HHH.EQ.0) GOTO 3306
      TQL=TQL+HHH
      IF (HHH.LT.MINQL) MINQL=HHH
      IF (HHH.GT.MAXQL) MAXQL=HHH
      K=K+1
3306  CONTINUE
      IF (MINQL.EQ.32767) MINQL=0
      IF (K.EQ.0) GOTO 3307
      TTQL=TQL
      TK=K
      AVQL=INT(TTQL/TK+0.5)
      PU2=0.0
      DO 3308 I=1,N
      II=NODEI(I,1)
      DO 3308 J=1,II
      IF (PNE(I,J).GT.1.0) GOTO 3308
      PU2=PU2+TPROC(I,J)
3308  CONTINUE
      PU1=((PU2-PU1)/(NOPR*HARD))*100.0
      III=INT(PU1+0.5)
      MUSE=(MUSE/(NOMEM*HARD))*100.0
      JJJ=INT(MUSE+0.5)
3307  WRITE (2,9105) TIME,NPKT,MNREC,TNREC,NBUFI(1,2),HQL(1,1),
     XHQL(1,2),HQL(1,3),HQL(2,1),HQL(2,2),HQL(2,3),HQL(3,1),
     XHQL(3,2),HQL(3,3),TQL,MINQL,MAXQL,K,AVQL,III,JJJ

      NHARD=HARD
      PU1=PU2
      MUSE=0.0
3300  IF (TIME.LT.LGSIM) GOTO 5000
      WRITE (2,9700) TIME
      WRITE (2,9702) EXEC
      WRITE (2,9600) PTY(1),PTY(2),PTY(3),PTY(4),PTY(5)
      WRITE (2,9610) TPT(1),TPT(2),TPT(3),TPT(4),TPT(5)
      WRITE (2,9620) SNC(1),SNC(2),SNC(3),SNC(4),SNC(5)
      WRITE (2,9630) RNC(1),RNC(2),RNC(3),RNC(4),RNC(5)
      F=0.0
      FX=0.0
      DO 5025 I=1,200
      IF (ML(I).EQ.0) GOTO 5025
      NUM=ML(I)
      THRU=TMT(I)/NUM
      WRITE (2,8200) I,ML(I),TMT(I),THRU
      TK=I
      F=F+NUM
      FX=FX+NUM*TK
5025  CONTINUE
      XMEAN=FX/F
      WRITE (2,8300) F,FX,XMEAN
      DO 5026 I=1,5
      DO 5026 J=1,10
5026  PUSE(I,J)=(TPROC(I,J)/LGSIM)*100.0
      WRITE (2,8210)
      CALL R10(10,TPROC)
      WRITE (2,8220)
      CALL R10(10,PUSE)
      STOP
8000  WRITE (2,9910)
      STOP
8005  WRITE (2,9800)
      STOP
8010  FORMAT(I1)
8020  FORMAT(2I6)
8030  FORMAT(2I6,F20.9)
8050  FORMAT(5I10)
8060  FORMAT(42X,6I6,2F14.9)
8070  FORMAT(5F20.9)
8075  FORMAT(F8.2,I3,I3,I3)
8080  FORMAT(I5,F20.9)
8085  FORMAT(F14.8)
8090  FORMAT(5X,3I12)
8100  FORMAT(11I8)
8200  FORMAT(2I6,2F14.9)
8210  FORMAT(1H0,5X,'TOTAL PROCESSOR TIME USED')
8220  FORMAT(1H0,5X,'TOTAL PROCESSOR TIME USED AS %' ,
     X'OF TIME AVAILABLE')
8300  FORMAT(1H0,5X,'   F = ',F8.1,'   FX = ',F8.1,
     X'   MEAN PACKET LENGTH = ',F8.1)
9010  FORMAT(1H1,5X,'CUMULATIVE FREQUENCY CURVE MEAN 1',/)
9020  FORMAT(1H1,5X,'RUN NUMBER =',I6,5X,'IAR =',F10.6,//)
9105  FORMAT(1H ,F5.3,I3,I4,F8.0,10I4,I6,4I4,2I6)
9110  FORMAT(//,5X,'NODES OPERATIVE =',5I5,//)
9111  FORMAT(//,5X,'DIST OF TYPES OF PACKETS PASSED',//)
9112  FORMAT(//,5X,'TIME TO THRUPUT TYPE OF PACKET',//)
9114  FORMAT(//,5X,'DIST OF MESS PASSED OF LENGTH 1 - 100')
9115  FORMAT(5X,'AND MEAN TIME TO THRUPUT',//)
9116  FORMAT(//,5X,'DIST OF SENDING NODES',//)
9117  FORMAT(//,5X,'DIST OF RECEIVING NODES',//)
9600  FORMAT(//,'   PTY =',5I6)
9610  FORMAT(//,'   TPT =',5F12.9)
9620  FORMAT(//,'   SNC =',5I6)
9630  FORMAT(//,'   RNC =',5I6,//)
9700  FORMAT(//,5X,'END OF SIMULATION TIME =',F14.9)
9702  FORMAT(5X,'NEXT LOOP EXECUTED ',F8.0,'   TIMES')
9800  FORMAT(//,5X,'DANGER - TOO MANY PACKETS IN SYSTEM')
9910  FORMAT(//,5X,'DANGER - EVENT TABLES GONE HAYWIRE')
      END
      SUBROUTINE DINPUT(N,LK,NET,NETB,NPLR,NCLR,NODE,NODEI,
     XNBUFI,CBUFI,H,HPLR,HCLR,HOSTS,HOSTC,HNU,NH,LGSIM,PKLGH,
     XCNTP,CHECK,HPBI,HCBI,MMR,SNET,NSP,BR,BRN,BRP,BRM,RUNNO,
     XMSM,MLM,MNGN,HCF,BITS,HARD,TI,STI)
C     INITIALISE SYSTEM

      INTEGER NODE(5),LK(5),H(5),NET(5,5),NETB(5,5),
     XSNET(5,5),NODEI(5,5),NBUFI(5,3),CBUFI(5,3),
     XHOSTC(5,5),HOSTS(5,5),BU(5),HPBI(5,5,3),HCBI(5,5,3),
     XHCF(25)
      REAL NPLR(5,5),NCLR(5,5),NSP(5,3),HPLR(5,5),
     XHCLR(5,5),HNU(5,5),LINEC(5,5),HLR(5,5),BRN(5),
     XBRP(5,10),BRM(5,10),STI(10,2)
      INTEGER NH,RUNNO,BR,MSM,MLM,MNGN,FREQ
      REAL LGSIM,PKLGH,CHECK,XLINEC,HHC,P,BITS,CNTP,
     XHARD,MMR,TI,E1,E2,AAS,TT

      READ (1,8040) RUNNO
      READ (1,8010) N
      IF (N.GT.5) GOTO 2100
      DO 1105 I=1,N
 1105 BU(I)=1
C     CLEAR ARRAY
      DO 2030 I=1,N
      DO 2040 J=1,5
      NET(I,J)=0
      SNET(I,J)=0
      NETB(I,J)=0
      NPLR(I,J)=0.0
      NCLR(I,J)=0.0
      HPLR(I,J)=0.0
      HCLR(I,J)=0.0
      HOSTS(I,J)=0
      HOSTC(I,J)=0
      HNU(I,J)=0.0
      NODEI(I,J)=0
 2040 CONTINUE
      DO 2080 J=1,10
      BRP(I,J)=0.0
      BRM(I,J)=0.0
 2080 CONTINUE
      DO 2090 I=1,10
      DO 2090 J=1,2
 2090 STI(I,J)=0.0
      NODE(I)=0
      LK(I)=0
      H(I)=0
      BRN(I)=0.0
 2030 CONTINUE
      DO 2035 I=1,25
 2035 HCF(I)=0
      DO 2085 I=1,N
      DO 2085 J=1,5
      DO 2085 K=1,3
      HPBI(I,J,K)=0
 2085 HCBI(I,J,K)=0
      DO 2070 I=1,5
      DO 2070 J=1,3
      CBUFI(I,J)=0
 2070 NBUFI(I,J)=0
```

```
      DO 2110 I=1,N
      DO 2110 J=1,3
2110  NSP(I,J)=0.0
      IF (N.GT.1) GOTO 1095
      NODE(1)=1
      GOTO 1082
1095  DO 1080 I=1,N
      READ (1,8010) LK(I)
C     LK(I) HOLDS NO OF LINES FROM NODE I
      LKKK=LK(I)
      DO 1090 J=1,LKKK
      READ (1,8090) NET(I,J),LINEC(I,J)
1115  K=NET(I,J)
      NETB(I,J)=BU(K)
      BU(K)=BU(K)+1
1090  CONTINUE
      NODE(I)=1
1080  CONTINUE
      DO 1085 I=1,N
      L=LK(I)
      DO 1085 J=1,L
      II=NET(I,J)
      JJ=NETB(I,J)
1085  SNET(II,JJ)=I
C     INITIALISE NODE ATTRIBUTES
1082  DO 1120 I=1,N
      READ (1,8020) NODEI(I,1)
      READ (1,8020) NODEI(I,2)
      READ (1,8050) NODEI(I,3)
      READ (1,8100) NSP(I,1)
      READ (1,8100) NSP(I,2)
1120  CONTINUE
C     INITIALISE HOST ATTRIBUTES
      NH=0
      DO 1130 I=1,N
      READ (1,8010) H(I)
      IF (H(I).GT.5) GOTO 2200
      NH=NH+H(I)
C     TOTAL NO OF HOSTS
      IH=H(I)
      DO 1150 J=1,IH
      READ (1,8096) HLR(I,J)
C     READ (1,8060) HOSTS(I,J)
      READ (1,8050) HOSTC(I,J)
      READ (1,8080) HNU(I,J)
1150  CONTINUE
1130  CONTINUE
      READ (1,8230) LGSIM
      READ (1,8260) BITS
      READ (1,8270) PKLGH
      READ (1,8260) CNTP
C     STORE TIME REQD TO ACCESS PACKET IN NSP
      DO 1154 I=1,N
      P=NODEI(I,3)
      NODEI(I,3)=P/PKLGH
      HHC=NSP(I,2)
      NSP(I,2)=HHC*PKLGH
1154  NSP(I,3)=HHC*CNTP
```

```
      DO 1155 I=1,N
      K=LK(I)
      DO 1155 J=1,K
      NCLR(I,J)=(CNTP*BITS)/LINEC(I,J)
1155  NPLR(I,J)=(PKLGH*BITS)/LINEC(I,J)
      DO 1165 I=1,N
      K=H(I)
      DO 1165 J=1,K
      HPLR(I,J)=(PKLGH*BITS)/HLR(I,J)
1165  HCLR(I,J)=(CNTP*BITS)/HLR(I,J)
      DO 1135 I=1,N
      K=H(I)
      DO 1135 J=1,K
      HHC=HOSTC(I,J)
      HPBI(I,J,1)=HHC/PKLGH
1135  HCBI(I,J,1)=HHC/PKLGH
      DO 1145 I=1,N
      P=NODEI(I,2)
      HHC=NODEI(I,3)
      CBUFI(I,1)=P*HHC
1145  NBUFI(I,1)=P*HHC
      READ (1,8235) CHECK
      READ (1,8235) HARD
      READ (1,8266) TI
      READ (1,8030) MSM
      READ (1,8030) MLM
      READ (1,8266) MMR
      READ (1,8030) MNGN
C     GENERATE CUM FREQ CURVE FOR GENERATING HOSTS
      ISUM=0
      E1=0.0
      AAS=32767.0
      DO 4400 I=1,NH
      TT=I
      E2=1.0-EXP(-TT/MNGN)
      FREQ=INT((E2-E1)*AAS)
      ISUM=ISUM+FREQ
      HCF(I)=ISUM
4400  E1=E2
      READ (1,8010) BR
      IF (BR.EQ.0) GOTO 9700
9720  READ (1,9001) I,T
      IF (I.EQ.0) GOTO 9740
      BRN(I)=T
      GOTO 9720
9740  READ (1,9002) I,J,T
      IF (I.EQ.0) GOTO 9750
      BRP(I,J)=T
      GOTO 9740
9750  READ (1,9002) I,J,T
      IF (I.EQ.0) GOTO 9700
      BRM(I,J)=T
      GOTO 9750
```

```
9700  I=1
9705  READ (1,8267) STI(I,1)
      IF (STI(I,1).EQ.0.0) GOTO 9760
      READ (1,8266) STI(I,2)
      IF (I.EQ.10) GOTO 9760
      I=I+1
      GOTO 9705

C     OUTPUT NETWORK CONFIGURATION DATA

9760  CALL PAT(N,LK,NET,NETB,NPLR,NCLR,NODE,NODEI,
     XNBUFI,CBUFI,H,HPLR,HCLR,HOSTS,HOSTC,HNU,NH,LGSIM,PKLGH,
     XCNTP,CHECK,HPBI,HCBI,MMR,SNET,NSP,BR,BRN,BRP,BRM,RUNNO,
     XMSM,MLM,MNGN,HCF,BITS,HARD,TI,STI)
      WRITE (2,9340)
      CALL R5(N,LINEC)
      WRITE (2,9390)
      CALL R5(N,HLR)
      RETURN
2100  WRITE (2,9310)
      RETURN
2200  WRITE (2,9320)
      RETURN

8010  FORMAT(I1)
8020  FORMAT(I2)
8030  FORMAT(I3)
8040  FORMAT(I4)
8050  FORMAT(I5)
8060  FORMAT(I6)
8070  FORMAT(5I5)
8080  FORMAT(F4.2)
8090  FORMAT(I1,F9.0)
8095  FORMAT(F6.0)
8096  FORMAT(F9.0)
8100  FORMAT(F10.9)
8220  FORMAT(F7.1)
8230  FORMAT(F7.3)
8235  FORMAT(F9.5)
8240  FORMAT(F6.1)
8250  FORMAT(F4.1)
8260  FORMAT(F3.0)
8265  FORMAT(F4.0)
8266  FORMAT(F8.6)
8267  FORMAT(F6.3)
8270  FORMAT(F5.0)
9000  FORMAT(5F8.2)
9001  FORMAT(I1,F4.2)
9002  FORMAT(I1,I2,F4.2)
9310  FORMAT(//,5X,'ERROR - MAX 5 NODES PERMITTED',//)
9320  FORMAT(//,5X,'ERROR - MAX 5 HOSTS PER NODE PERMITTED',//)
9340  FORMAT(//,5X,'LINE CAPACITY (BAUDS)',//)
9390  FORMAT(//,5X,'HOST LINE RATE',//)
      END
      SUBROUTINE PAT(N,LK,NET,NETB,NPLR,NCLR,NODE,NODEI,
     XNBUFI,CBUFI,H,HPLR,HCLR,HOSTS,HOSTC,HNU,NH,LGSIM,PKLGH,
     XCNTP,CHECK,HPBI,HCBI,MMR,SNET,NSP,BR,BRN,BRP,BRM,RUNNO,
     XMSM,MLM,MNGN,HCF,BITS,HARD,TI,STI)
      INTEGER NODE(5),LK(5),H(5),NET(5,5),NETB(5,5),SNET(5,5),
     XNODEI(5,5),NBUFI(5,3),CBUFI(5,3),HOSTC(5,5),HCF(25),
     XHOSTS(5,5),BU(5),HPBI(5,5,3),HCBI(5,5,3)
      REAL NPLR(5,5),NCLR(5,5),NSP(5,3),HPLR(5,5),
     XHCLR(5,5),HNU(5,5),LINEC(5,5),HLR(5,5),BRN(5),
     XBRP(5,10),BRM(5,10),STI(10,2)
      INTEGER NH,RUNNO,BR,MSM,MLM,MNGN
      REAL LGSIM,PKLGH,CHECK,XLINEC,HHC,P,BITS,CNTP,
     XHARD,MMR,TI

C     OUTPUT NETWORK CONFIGURATION DATA

      WRITE (2,9290) RUNNO
      WRITE (2,9300) N
      WRITE (2,9310) LGSIM
      WRITE (2,9315) BITS
      WRITE (2,9320) PKLGH
      WRITE (2,9321) CNTP
      WRITE (2,9325) CHECK
      WRITE (2,9326) HARD
      WRITE (2,9317) TI
      WRITE (2,9324) MMR
      WRITE (2,9322) MSM
      WRITE (2,9323) MLM
      WRITE (2,9938) MNGN
      WRITE (2,9232)
      DO 4042 I=1,NH
4042  WRITE (2,9006) I,HCF(I)
      WRITE (2,9330)
      CALL O5(N,NET)
      WRITE (2,9335)
      CALL O5(N,NETB)
      WRITE (2,9337)
      CALL O5(N,SNET)
      WRITE (2,9345)
      CALL R5(N,NPLR)
      WRITE (2,9348)
      CALL R5(N,NCLR)
      WRITE (2,9350)
      DO 1160 I=1,N
      WRITE (2,8050) NODE(I)
1160  CONTINUE
      WRITE (2,9360)
      DO 1170 I=1,N
      WRITE (2,8050) LK(I)
1170  CONTINUE
      WRITE (2,9370)
      WRITE (2,9371)
      WRITE (2,9372)
      WRITE (2,9373)
      WRITE (2,9374)
      CALL O5(N,NODEI)
      WRITE (2,9375)
      DO 1191 I=1,N
1191  WRITE (2,8210) I,NSP(I,1),NSP(I,2),NSP(I,3)
      WRITE (2,9380)
      DO 1190 I=1,N
      WRITE (2,8050) H(I)
1190  CONTINUE
      WRITE (2,9395)
      CALL R5(N,HPLR)
      WRITE (2,9398)
      CALL R5(N,HCLR)
      WRITE (2,9400)
      CALL O5(N,HOSTS)
      WRITE (2,9410)
      CALL O5(N,HOSTC)
      WRITE (2,9430)
      CALL R5(N,HNU)
      WRITE (2,9440)
      DO 2010 I=1,5
2010  WRITE (2,8065) NBUFI(I,1)
      WRITE (2,9450)
      DO 2020 I=1,N
2020  WRITE (2,8075) HPBI(I,1,1),HPBI(I,2,1),
     XHPBI(I,3,1),HPBI(I,4,1),HPBI(I,5,1)
      WRITE (2,9810)
      DO 9490 I=1,5
9490  WRITE (2,9003) BRN(I)
      WRITE (2,9820)
      CALL R10(10,BRP)
      WRITE (2,9830)
      CALL R10(10,BRM)
      WRITE (2,9835)
      WRITE (2,9836)
      DO 2400 I=1,10
2400  WRITE (2,9005) STI(I,1),STI(I,2)
      RETURN
8050  FORMAT(I5)
8065  FORMAT(5X,I6)
8075  FORMAT(5X,5I5)
8210  FORMAT(I10,3F15.9)
9003  FORMAT(F18.9)
9004  FORMAT(11I10)
9005  FORMAT(5X,2F10.3)
9006  FORMAT(5X,2I10)
9290  FORMAT(1H1,5X,'RUN NO =',I8)
9300  FORMAT(//,5X,'NO OF NODES = ',I4)
9310  FORMAT(//,5X,'LENGTH OF SIMULATION = ',F7.3,' SECS')
9315  FORMAT(//,5X,'WORD LENGTH = ',F8.1,' BITS')
9320  FORMAT(//,5X,'PACKET LENGTH = ',F8.1,' WORDS')
9321  FORMAT(//,5X,'CONTROL PACKET LENGTH =',F8.1,' WORDS')
9317  FORMAT(//,5X,'TRAFFIC INTENSITY = ',F8.6)
9322  FORMAT(//,5X,'MEAN NO OF PKTS IN SHORT MESS = ',I6)
9323  FORMAT(//,5X,'MEAN NO OF PKTS IN LONG MESS = ',I6)
```

```
9938  FORMAT(//,5X,'MEAN NO OF GENERATING HOSTS =',I6)
9232  FORMAT(//,5X,'GENERATING HOST CUM FREQ',//)
9324  FORMAT(//,5X,'MESSAGE MIX RATIO (SHORT/LONG) = ',F5.3)
9325  FORMAT(//,5X,'EPOCH FREQUENCY OF CHECK = ',F9.5,' SECS')
9326  FORMAT(//,5X,'EPOCH FREQUENCY OF HARD COPY = ',F9.5,' SECS')
9330  FORMAT(//,5X,'NET',//)
9335  FORMAT(//,5X,'NETB',//)
9337  FORMAT(//,5X,'SNET',//)
9345  FORMAT(//,5X,'NODE LINE TRANSMISSION TIME ',
     X'FOR ONE PKT(SECS)',//)
9348  FORMAT(//,5X,'NODE LINE TRANS TIME FOR ONE ',
     X'CONTROL PACKET(SECS)',//)
9350  FORMAT(//,5X,'NODE',//)
9360  FORMAT(//,5X,' LK',//)
9370  FORMAT(//,5X,'NODEI')
9375  FORMAT(//,5X,'NODE I   PROC SPEED     PKT SPEED      ',
     X'CONTROL PACKET SPEED',//)
9380  FORMAT(//,5X,'NO OF HOSTS ON EACH NODE',//)
9395  FORMAT(//,5X,'HOST LINE TRANSMISSION TIME ',
     X'FOR ONE PACKET(SECS)',//)
9398  FORMAT(//,5X,'HOST LINE TRANS TIME ',
     X'FOR ONE CONTROL PACKET(SECS)',//)
9400  FORMAT(//,5X,'HOST M/C SPEED',//)
9410  FORMAT(//,5X,'HOST M/C CAPACITY FOR PKT STORAGE',//)
9430  FORMAT(//,5X,'HOST NETWORK UTILISATION',//)
9440  FORMAT(//,5X,'NODE PKT CAPACITY',//)
9450  FORMAT(//,5X,'HOST PKT CAPACITY',//)
9371  FORMAT(//,5X,'NO OF PROCESSORS')
9372  FORMAT(//,5X,'NO OF MEM UNITS')
9373  FORMAT(//,5X,'MEM UNIT CAPACITY IN PKTS')
9374  FORMAT(//,5X,'TOTAL SIZE OF MEM IN PACKETS',//)
9460  FORMAT(//,5X,'IF BREAKDOWN REQUIRED INPUT 1 ELSE INPUT 0')
9810  FORMAT(//,5X,'NODE BREAKDOWN TIMES',//)
9820  FORMAT(//,5X,'PROCESSOR BREAKDOWN TIMES',//)
9830  FORMAT(//,5X,'MEMORY BREAKDOWN TIMES',//)
9835  FORMAT(//,5X,'STEP TRAFFIC INTENSITY ',//)
9836  FORMAT(5X,'TIME CHANGE  NEW TRAFFIC INTENSITY',//)
      END
      SUBROUTINE TRANS(PKT,TPIN,K,VEC,NREC,TIME,PTY,TPT,
     XTIN,ML,TMT,SNC,RNC)

C     RECORD PKT STATISTICS AND REMOVE PKT FROM SYSTEM

      INTEGER PKT(500,5),VEC(9),PTY(5),ML(200),SNC(5),RNC(5)
      REAL TPT(5),TIN(25,25),TMT(200),TPIN(500,2)
      INTEGER S,D,TYPE
      REAL TIME,TI,TO

      DO 3000 I=1,5
      VEC(I)=PKT(K,I)
      PKT(K,I)=0
3000  CONTINUE
      TI=TPIN(K,1)
      TO=TPIN(K,2)
      TPIN(K,1)=0.0
      TPIN(K,2)=0.0
      NREC=NREC+1
C     RECORD PACKET STATISTICS
      TYPE=VEC(3)+1
      PTY(TYPE)=PTY(TYPE)+1
C     TYPE OF PACKET PASSED
      IF (VEC(3).NE.0) GOTO 5300
      TPT(TYPE)=TPT(TYPE)+TO-TI
      GOTO 5400
5300  TPT(TYPE)=TPT(TYPE)+TIME-TI
C     TIME TO THRUPUT TYPE OF PACKET
5400  IF (VEC(3).NE.1) GOTO 5200
      I=VEC(1)-100
      J=VEC(2)-100
      SNC(I)=SNC(I)+1
      RNC(J)=RNC(J)+1
C     RECORD SENDING NODE AND RECEIVING NODE
5200  IF (VEC(3).NE.0) GOTO 5050
      S=VEC(1)
      D=VEC(2)
      IF (VEC(4).NE.1) GOTO 5100
C     IF FIRST PACKET OF MESS RECORD TIME IN SYSTEM
      TIN(S,D)=TI
5100  IF (VEC(4).NE.VEC(5)) GOTO 5050
C     IF EQUAL WHOLE MESS THRUPUTTED
      L=VEC(5)
      ML(L)=ML(L)+1
C     RECORD LENGTH OF PACKET
      TMT(L)=TMT(L)+TO-TIN(S,D)
      TIN(S,D)=0.0
C     RECORD LENGTH OF TIME TO THRUPUT MESSAGE
5050  RETURN
      END
      SUBROUTINE O5(N,ANNA)
      INTEGER ANNA(5,5)
      DO 2000 I=1,N
      WRITE (2,8000) I,ANNA(I,1),ANNA(I,2),ANNA(I,3),
     XANNA(I,4),ANNA(I,5)
 2000 CONTINUE
      RETURN
 8000 FORMAT(1H ,5X,'NODE',I2,5I10)
      END
```

```
      SUBROUTINE O10(M,ANNA)
      INTEGER ANNA(5,10)
      WRITE (2,7000)
      DO 2000 I=1,M
      WRITE (2,8000) I,ANNA(1,I),ANNA(2,I),ANNA(3,I),
     XANNA(4,I),ANNA(5,I)
 2000 CONTINUE
      RETURN
 7000 FORMAT(1H0,12X,'NODE1',5X,'NODE2',5X,'NODE3',
     X5X,'NODE4',5X,'NODE5',//)
 8000 FORMAT(1H ,5X,I2,5I10)
      END
```

```
      SUBROUTINE O50(M,ANNA)
      INTEGER ANNA(5,50)
      WRITE (2,7000)
      DO 2000 I=1,3
      WRITE (2,8000) I,ANNA(1,I),ANNA(2,I),ANNA(3,I),
     XANNA(4,I),ANNA(5,I)
 2000 CONTINUE
      DO 3000 I=48,50
      WRITE (2,8000) I,ANNA(1,I),ANNA(2,I),ANNA(3,I),
     XANNA(4,I),ANNA(5,I)
 3000 CONTINUE
      RETURN
 7000 FORMAT(1H1,12X,'NODE1',5X,'NODE2',5X,'NODE3',
     X5X,'NODE4',5X,'NODE5',//)
 8000 FORMAT(1H ,5X,I2,5I10)
      END
      SUBROUTINE R5(N,ANNA)
      REAL ANNA(5,5)
      DO 2000 I=1,N
      WRITE (2,8000) I,ANNA(I,1),ANNA(I,2),ANNA(I,3),
     XANNA(I,4),ANNA(I,5)
2000  CONTINUE
      RETURN
8000  FORMAT(1H ,5X,'NODE',I2,5F20.9)
      END
```

```
      SUBROUTINE R10(M,ANNA)
      REAL ANNA(5,10)
      WRITE (2,7000)
      DO 2000 I=1,M
      WRITE (2,8000) I,ANNA(1,I),ANNA(2,I),ANNA(3,I),
     XANNA(4,I),ANNA(5,I)
2000  CONTINUE
      RETURN
7000  FORMAT(1H0,16X,'NODE1',13X,'NODE2',13X,'NODE3',
     X13X,'NODE4',13X,'NODE5',//)
8000  FORMAT(1H ,5X,I2,5F18.9)
      END
      SUBROUTINE SUBS(NMG,H,II,JJ)

C     GIVEN ADDRESS CODE RETURNS NODE AND HOST NO

      INTEGER H(5)
      JJ=NMG
      DO 3000 II=1,5
      IF (JJ.LE.H(II)) GOTO 3010
      JJ=JJ-H(II)
3000  CONTINUE
3010  RETURN
1000  FORMAT(//,5X,'ERROR - ADDRESS CODE > NO OF HOSTS',//)
      END




      SUBROUTINE ISUBS(NMG,H,II,JJ)

C     GIVEN NODE AND HOST NO RETURNS ADDRESS CODE

      INTEGER H(5)
      IF ((II.NE.0).AND.(JJ.NE.0)) GOTO 3010
      WRITE (2,4000)
      STOP
3010  IF (JJ.LE.H(II)) GOTO 3020
      WRITE (2,34) II,JJ,H(II)
34    FORMAT('II=',I6,'JJ=',I6,I6)
      WRITE (2,5000)
      STOP
3020  NMG=0
      KK=II-1
      I=1
3030  IF (I.GT.KK) GOTO 3000
      NMG=NMG+H(I)
      I=I+1
      GOTO 3030
3000  NMG=NMG+JJ
      RETURN
4000  FORMAT(//,5X,'ERROR - ONE OF SUBSCRIPTS ZERO',//)
5000  FORMAT(//,5X,'ERROR - SUBSCRIPT OUT OF RANGE',//)
      END
      SUBROUTINE RAND(A,B,SUM)
      INTEGER OP(10),A(16),B(14)
      INTEGER C,D,E,F,SUM
      REAL R
C     EVALUATES NEW BITS FOR CHAINCODES AND SHIFT
      C=MOD2(A(8),A(15))
      DO 10 I=1,14
      II=16-I
      JJ=15-I
      A(II)=A(JJ)
10    CONTINUE
      A(1)=C
      D=MOD2(B(13),B(4))
      E=MOD2(B(3),D)
      F=MOD2(B(1),E)
      DO 20 I=1,12
      II=14-I
      JJ=13-I
      B(II)=B(JJ)
20    CONTINUE
      B(1)=F
C     PICK-OFF TEN BIT WORD FROM CHAINCODES USING
C     MODULO-2 ADDITION
      OP(1)=MOD2(A(1),B(7))
      OP(2)=MOD2(A(3),B(11))
      OP(3)=MOD2(A(5),B(9))
      OP(4)=MOD2(A(7),B(13))
      OP(5)=MOD2(A(9),B(12))
      OP(6)=MOD2(A(11),B(10))
      OP(7)=MOD2(A(13),B(8))
      OP(8)=MOD2(A(15),B(6))
      OP(9)=MOD2(A(10),B(5))
      OP(10)=MOD2(A(6),B(3))
C     SUM THIS WORD INTO DECIMAL FORM
      SUM=0
      DO 30 I=1,10
      SUM=2*SUM+OP(I)
30    CONTINUE
C     SUM IS THE RETURNED RANDOM NO IN RANGE 0 - 1023
      RETURN
      END




      FUNCTION MOD2(I,J)

C     PROVIDES MODULO-2 OR HALF ARITHMETIC
C     REQUIRED BY RAND AND GEN

      K=I+J
      IF (K.NE.2) GOTO 10
      K=0
10    MOD2=K
      RETURN
      END
      SUBROUTINE GEN(A,B,C,D,SUM)
      INTEGER OP(15),A(21),B(21),C(21),D(21)
      INTEGER SUM,I,J,K,L,M,E,F,G,H
C     EVALUATE NEW BITS FOR CHAINCODES AND SHIFT
      L=MOD2(A(8),A(15))
      G=MOD2(C(7),C(15))
      DO 1000 I=1,14
      II=16-I
      JJ=15-I
      A(II)=A(JJ)
1000  C(II)=C(JJ)
      A(1)=L
      C(1)=G
      M=MOD2(B(13),B(4))
      E=MOD2(B(3),M)
      F=MOD2(B(1),E)
      H=MOD2(D(11),D(10))
      DO 2000 I=1,12
      II=14-I
      JJ=13-I
2000  B(II)=B(JJ)
      DO 3000 I=1,10
      II=12-I
      JJ=11-I
3000  D(II)=D(JJ)
      B(1)=F
      D(1)=H
      OP(1)=MOD2(A(1),D(3))
      OP(2)=MOD2(B(13),C(9))
      OP(3)=MOD2(A(14),B(3))
      OP(4)=MOD2(B(11),D(11))
      OP(5)=MOD2(A(5),C(1))
      OP(6)=MOD2(C(15),B(8))
      OP(7)=MOD2(B(6),D(7))
      OP(8)=MOD2(A(3),C(3))
      OP(9)=MOD2(C(7),D(1))
      OP(10)=MOD2(A(7),B(5))
      OP(11)=MOD2(C(13),D(5))
      OP(12)=MOD2(D(9),A(12))
      OP(13)=MOD2(B(1),C(5))
      OP(14)=MOD2(A(9),C(11))
      OP(15)=MOD2(D(2),B(10))
C     CONVERT TO DECIMAL FORM
      SUM=0
      DO 4000 K=1,15
4000  SUM=2*SUM+OP(K)
      RETURN
      END
      SUBROUTINE ALLOC(I,PFL,PNE,P,OCC,TPROC)

C     ASSIGN PROCESSOR AND SHIFT QUEUE
C     ELSE RETURN NULL PROCESSOR

      INTEGER PFL(5,10)
      REAL PNE(5,10),TPROC(5,10)
      REAL OCC
      INTEGER P
      P=PFL(I,1)
C     GET A PROCESSOR FROM NODE I FREE LIST
      IF (P.NE.0) GOTO 1000
      RETURN
1000  PNE(I,P)=OCC
C     SET PROC BUSY FOR OCC SECS
C     ADD CUMULATIVE TIME OF PROCESSOR USAGE
      TPROC(I,P)=TPROC(I,P)+OCC
C     NOW SHIFT QUEUE UP
      DO 2000 J=1,9
2000  PFL(I,J)=PFL(I,J+1)
      PFL(I,10)=0
      RETURN
      END
```

```
      SUBROUTINE NPR(I,NI,HNE,TRY)

C     GIVES TIME NEXT PROCESS RELEASED

      REAL HNE(5,10)
      REAL TRY
      TRY=1.0
      DO 1000 K=1,NI
      IF (HNE(I,K).EQ.0.0) GOTO 1000
      IF (HNE(I,K).GE.TRY) GOTO 1000
      TRY=HNE(I,K)
1000  CONTINUE
      IF (TRY.NE.1.0) GOTO 2000
      TRY=0.0000001
2000  RETURN
      END
      SUBROUTINE PROC(I,PORD,UPRP,NPROC,PFL,PNO)

C     GIVES NEXT PROCESS TO BE EXECUTED

      INTEGER PORD(5,5),UPRP(5)
      INTEGER PNO,PFL
      IF (PNO.EQ.4) GOTO 5000
C     ALL PROCESSES FOR NODE I COMPLETED GOTO NEXT NODE
      NPROC=UPRP(1)
      IF (PFL.EQ.0) GOTO 1000
      DO 2000 J=1,3
      UPRP(J)=UPRP(J+1)
2000  PORD(I,J)=UPRP(J)
      UPRP(4)=NPROC
      PORD(I,4)=NPROC
      GOTO 3000
1000  DO 4000 J=1,3
4000  UPRP(J)=UPRP(J+1)
      UPRP(4)=NPROC
3000  PNO=PNO+1
      RETURN
5000  NPROC=5
C     GOTO NEXT NODE
      RETURN
      END




      SUBROUTINE CHOP(CUM,TOTAL,CLASS)

C     BINARY SEARCH OF CUMULATIVE FREQUENCY TABLE

      INTEGER TOTAL,CLASS
      INTEGER CUM(600)
      IF (TOTAL.GE.32316) GOTO 5000
      CLASS=256
      II=256
      DO 2000 K=1,8
      II=II/2
      IF (TOTAL.GT.CUM(CLASS)) GOTO 3000
      IF (TOTAL.GT.CUM(CLASS-1)) GOTO 4000
      CLASS=CLASS-II
      GOTO 2000
3000  CLASS=CLASS+II
2000  CONTINUE
      GOTO 4000
5000  DO 6000 CLASS=512,579
      IF (TOTAL.LE.CUM(CLASS)) GOTO 4000
6000  CONTINUE
4000  RETURN
      END
      FINISH
****
```

APPENDIX III

This section contains the Autocorrelation and Spectral Analysis program.

```
C     AUTOCORRELATION AND SPECTRAL ANALYSIS
      REAL T(101),S(101),F(101),G(101),C(101),X(10000)
      REAL W(101),R(101),L(101),U(101)
      REAL RR,CP,FP,GP,SP,TP,PI,PP,QQ,MM
      INTEGER P,Z,Q
      INTEGER COUNTR,CTQL
C     READ SAMPLE SIZE
      READ (1,8050) N
C     READ TIME LAG M
      READ (1,8050) M
      M=M+1
      MM=M
      P=1
      I=1
      DO 4000 J=1,M
      T(J)=0
      S(J)=0
      L(P)=0.0
4000  C(J)=0
      SUM=0.0
      SS=0.0
C     READ SAMPLES INTO X
      DO 4010 J=1,N
      READ (1,8010) COUNTR,TIME,CTQL
      X(J)=CTQL
      SUM=SUM+X(J)
      SS=SS+X(J)*X(J)
4010  CONTINUE
C     CALCULATE MEAN AND STANDARD DEVIATION
      XN=N
      SUM=SUM/XN
      SS=SQRT(SS/XN-SUM*SUM)
C     NORMALISE DATA
      DO 4015 I=1,N
4015  X(J)=(X(J)-SUM)/SS
      WRITE (2,8001) SUM,SS
8001  FORMAT ( ' SUM = ',F8.4,' SQUARE = ',F8.4)
      DO 4020 I=1,N
      T(P)=T(P)+X(I)
4020  S(P)=S(P)+X(I)*X(I)
      F(P)=T(P)
      G(P)=S(P)
      DO 4030 P=2,M
      Z=N-P+2
      T(P)=T(P-1)-X(P-1)
      F(P)=F(P-1)-X(Z)
      S(P)=S(P-1)-X(P-1)*X(P-1)
4030  G(P)=G(P-1)-X(Z)*X(Z)
      DO 4040 P=1,M
      J=N-P+1
      DO 4050 I=1,J
      K=I+P-1
4050  C(P)=C(P)+X(I)*X(K)
C     CALCULATE AUTOCOVARIANCE
      W(P)=C(P)
      RR=N-P+1
      W(P)=W(P)/RR
```

```
C       CALCULATE AUTOCORRELATION COEFFICIENT
        CP=C(P)
        FP=F(P)
        GP=G(P)
        SP=S(P)
        TP=T(P)
        R(P)=(RR*CP-FP*TP)/(SQRT(RR*GP-FP*FP)*SQRT(RR*SP-TP*TP))
4040    CONTINUE
        PI=3.1415926
        DO 4060 P=1,M
        PP=P-1.0
        K=M-2
        DO 4070 Q=1,K
        QQ=Q
4070    L(P)=L(P)+2.0*W(Q+1)*COS((PI*PP*QQ)/(MM-1.0))
4060    L(P)=L(P)+W(1)+W(M)*COS(PP*PI)
        U(1)=0.23*L(1)+0.54*L(1)+0.23*L(2)
        K=M-1
        DO 4080 P=2,K
4080    U(P)=0.23*L(P-1)+0.54*L(P)+0.23*L(P+1)
        U(M)=0.23*L(M-1)+0.54*L(M)+0.23*L(M)
        DO 4090 II=1,M
        P=II-1
4090    WRITE (2,8100) P,T(II),F(II),S(II),G(II),C(II),
       XW(II),R(II),L(II),U(II)
        STOP
8010    FORMAT(I9,F12.6,I10)
8050    FORMAT(I4)
8100    FORMAT(1X,I6,5F13.2,4F12.4)
        END
****
```