SIMULATION OF DISTRIBUTED COMPUTER NETWORKS

by H. F. WEEHUIZEN

Submitted to the University of Cape Town in fulfilment of the requirements for the degree of Doctor of Philosophy.

August 1987

ABSTRACT

This is a study of the simulated performance of two local area networks, Ethernet and the MAP network, respectively based on the IEEE standards 802.3 and 802.4.

The simulation language chosen is of the discrete event type rather than the more usual analytical model. This is done in order to observe the interaction between the various entities of a network in order to gain a better understanding of the method of operation of such a system.

The performance demanded of a node entity by the networks is determined. The performance of some commercially available hardware is derived from manufacturer's specifications and compared with that required by the network. It is found that there is a significant disparity, with the network requirements far exceeding that of the hardware capabilities.

The simulation models developed are used to determine the performance of the networks both with and without the limitations imposed by currently available hardware.

While the inclusion of the hardware performance causes little loss in performance for the Ethernet network, it has a highly detrimental effect on that of the MAP network. A possible

solution is found to this limitation which requires minimal change to the existing protocol.

The conclusions reached are that with currently available hardware a group of nodes are able to fully utilise the performance of the Ethernet LAN although a single pair of nodes is unable to do so. With regard to the MAP network, the network performance is limited by that of the node performance although this can be offset to a certain extent by careful choice of one of the protocol parameters, or modification of the hardware design.

ACKNOWLEDGEMENTS

I would like to thank the following people:

My supervisor, Prof S.G.McLaren for his help and encouragement throughout this thesis.

Prof M.G.Rodd who suggested this project to me and gave advice at some crucial points in the development of this thesis.

My wife, Gilda who patiently endured many long hours of solitude as I worked on this project and who proof-read what was to her a largely unintelligable manuscript in very little time.

The CSIR who allowed me to have time off to consult my supervisors.

CONTENTS

Appendices

CHAPTER 1

INTRODUCTION

The field of distributed computer networks has been in existence for more than a decade. Because of the complexity of the systems and the large number of new parameters that this sort of system introduces into the field, it is still in its embryonic stage of development. A network of computers offers advantages to all types of computing, from interconnection of mainframes to microcomputers. The interconnection of mainframes allows the sharing of resources among a wider circle of users over a greater geographical area. In the office automation environment, expensive hardware resources can be shared among the relatively inexpensive computers, messages can be interchanged and data can be shared among users. This last area of application has both introduced and popularised the concept of the Local Area Network, commonly known as the LAN. The LAN is a short haul network limited to distances of a kilometre or two, often confined to a building.

The field of interest in this thesis is the local area network, but relating to the use of the system in a control environment where integrity of the data is paramount. One of the important advantages of such a system is the increased reliability and

survivability that distributed intelligence offers over centrally controlled systems.

This is a study of the behaviour of the data transport systems and was started as the result of the development of a pilot distributed computer network in the late 1970's. It soon became apparent that there were a large number of design decisions which had to be made in the process of developing this system for which there was no easy way of finding a solution. Arbitrary decisions had to be made which could have a significant effect on the performance and reliability of the system. Thus it was decided to start this project in order to find more systematic solutions to some of the problems.

The pilot network mentioned above(1,2) is modelled after the bigger long haul networks on which most of the early research was done. Thus it is of the store and forward type and has a ring topology. This topology was chosen as it meant that the routing strategy would be simple with a consequent reduction in the number of design parameters that had to be considered.

As this work progressed it soon became apparent that most of the development in the field of local area networks centred around the single bus system using either the Carrier Sense Multiple Access with Collision Detection(CSMA/CD) or the token passing protocols. Thus for this work to be relevant the latter types of

local area networks became the subject of this study.

THE CSMA/CD PROTOCOL

This protocol was characterised in the early days by the Ethernet network but since a slightly modified version of this has been adopted as the IEEE 802.3 standard(3) most of the development has centred around this. Several commercial networks based on this are available, but they are nearly all geared for the office automation market and are targetted for the personal computer(4,5). In order to keep the cost of each node down these nodes are very basic, heavily impacting the performance of the node while there is network activity involving the local node. In an effort to minimise these effects, a master node is designated which is the file and printer server and network superviser. All communication in the network is via this node. Thus a star configuration is imposed on the system, negating most of the advantages of this type of network. It will not be considered further.

A CSMA/CD network of the type that is of interest is that produced by Intel(6,7). This is a truly distributed system offering good performance with a choice of node configurations ranging from a single processor to multiple processors with one of the processors serving as a communications processor. Software is available that provides support up to at least layer 4 of the ISO reference model for open systems interconnection(8). This provides inter-process communication between nodes at the level

that is required for a typical process control environment. A system has been commercially produced using this hardware that is aimed at the process control market(9).

THE TOKEN PASSING PROTOCOL

A major criticism that is aimed at the CSMA/CD protocol for the control environment is that the packet transport time is probabalistic rather than deterministic. This means that the transport time of a packet cannot be determined with any great deal of certainty and can become unpredictably long during heavy bus activity. For this reason the token passing protocol is recommended by several people in preference to the former.

Several busses using this protocol are defined such as the Arcnet(10),the IBM net(11) and the Proway bus(12). In an attempt to standardise this protocol the IEEE has drawn up the standard 802.4 (13). Probably the bus of this type that has attracted the most interest in the control environment is that being developed by the General Motors Electronics Company and is called the Manufacturing and Automation Protocol (MAP) architecture(14).

The oldest of this type of bus is the Arcnet. It has a fully distributed control function which would make it a candidate for the control environment. Unfortunately it is a character based protocol and is thus unsuitable for this environment. The IBM net is aimed at the office automation environment and has a slightly different protocol from the others in that it is a token

ring system as opposed to token bus. Unfortunately, in order to take advantage of existing building wiring, the physical wiring of the network is arranged in the form of a star. All the electronics handling the transport protocol are located at a central site with the co-operating nodes connected as satellites to this hardware. This is a disadvantage from a control point of view and as there are far more suitable systems available, this system will not be considered further.

The Proway bus, although aimed at the control environment is not a fully distributed system. It has bus managers and supervisors which act as contol centres for the bus. Only designated nodes can perform these functions so this system is less apposite than some of the other busses.

This brings us back to the General Moters MAP architecture. As previously stated, it conforms to the IEEE standard 802.4 which is a fully distributed protocol and the bus is designed for the industrial environment. It also has the support of most of the large industrial companies of the United States so has an excellent chance of gaining acceptance. This is the token passing protocol that has been chosen for study here.

THE STRATEGY

The aim of this thesis is to develop computer models of these networks so that they can be used to study the

characteristics of each type. The performance of these can be compared when identical stimuli are applied to both types, thus allowing a valid comparison between them. As a corollary to this, if a future network load is known the models can be used for network selection by applying that load to these models. The network that provides acceptable performance at reasonable cost would be the system of choice.

A computer model is chosen as the means of system evaluation because the cost of developing actual systems would be prohibitive. Currently an Ethernet costs about $1000 while that of of a MAP node is $5000 - $10000. Using a computer model also provides convenience as the input stimuli to the systems and the performance parameters that are monitered can readily be changed to suit particular requirements. System parameters can also be easily varied and taken beyond the limits of values normally achievable in practical systems.

The results obtained from computer models are heavily dependant on the nature of the input stimuli. Thus if the nature of the load on the network is not known, input stimuli are normally taken to be random in nature. This provides a standard basis for comparison between different computer models.

If the load for the network is known then this can be applied to the model to obtain a more accurate assessment of system

performance. This would be the only effective way of making a performance comparison between different networks when a solution to a particular application is being sought.

The system model by its very nature is always different from the system being modelled. One of the tasks of the model builder is to check that these differences are not significant. One of the simplest ways of doing this, if the actual system exists, is by model validation. This involves driving the model and the actual system with a similar set of input stimuli and then comparing the results so obtained. In one of the two systems simulated here performance figures are available for comparison. These are the published results of studies on the same systems using different modelling techniques.

Parameters monitered.

Having discussed the types of systems that are being modelled, it now becomes necessary to discuss what parameters are being studied in this project. Most papers dealing with the modelling of the CSMA/CD networks consider the performance only from the aspects of throughput(15) and throughput delay (16,17,18). These parameters are important but there are others that have an influence on the performance of the system that have not been examined. The token access methods are still in the fairly early stages of development and not a great deal of information is yet available on their performance(19,20,17).

This study examines such parameters as throughput, throughput delay, effects of buffer size and processing delay on the performance of the computer network. Performance is viewed from the aspects of throughput, cost and reliability. All these have a bearing on the survivability of the system which is one of the most important criteria in the choice of the system.

Each of the above terms will now be briefly explained so to clearly establish the aims and limitations of this study. The first parameter is throughput. This is a measure of the number of packets reaching a destination per unit time. This is a crude measure of system performance as it says nothing of the time it takes for a packet to get to its destination or how many packets were lost in the process of communication. The results are normally displayed graphically where the throughput is plotted against load to show the nature of the system saturation as maximum throughput is approached.

Throughput delay is also depicted graphically. This shows how the nature of the delay changes with change in load. It will be seen how markedly different this characteristic is for each network. The above two parameters have frequently been modelled for the Ethernet system so data for model validation is readily available for comparison.

The size of the message buffer used by the communications

processor can have a noticeable effect on the performance of some of the systems depending on the communications method used. In the store and forward protocol these buffers hold the messages for onward transmission to the destinaton node. If packet by packet acknowledgment is used in the communication strategy the buffers have to hold a copy of the packet until receipt by the destination node is acknowledged, thus buffer size can limit the rate at which packets can be handled by the network.

All systems are subject to communications failure in the their normal course of operation. Its effect can be temporary in nature when the system is influenced by noise or it can be permanent when a line failure occurs. This can be as a result of a physical line break or a hardware failure. The effects of this on the system must be studied to see how its reliability is affected. Communications strategy must be adjusted to minimise its influence.

Processing delay is the amount of time a communications processor takes to receive a packet, check its validity and take appropriate action. This time obviously has an effect on throughput but less obviously, it also determines the time that a node takes to ready itself for the receipt of the next packet. This is called recovery time. In the single bus system packets can be sent to a particular node from more than one source. If one is sent immediately after the other according to the general bus specifications the receiving node will not receive the the

second packet correctly if its recovery time is too long. The overall aim of this thesis is to compare the performance of the systems described. This needs to be examined from various aspects. Already discussed is system throughput. The next is performance in relation to cost. One might say that this aspect could be ignored for a control system because its total cost in relation to the plant being controlled is small, so the relative cost between two systems would of little account. However it is sound practice to know how cost effective a particular feature of a system is.

The following aspects are of major importance in a control environment as they are closely related to overall reliability and efficiency of the plant. Plant controlled by systems such as those discussed here rely totally on the control system for their operation. Thus any malfunction of the control system results in partial or complete failure of plant operation. This results in lost production, loss of raw materials and loss of return on capital investment.

The first of these is reliability. It can be described here as the ability of the data transport system to carry information without loss under optimum operating conditions. That is, this is an evaluation of the performance when there are no hardware failures in the transport system and data is not corrupted by system malfunction or external influence.

The second is survivability. This is a measure of the systems ability to function in the presence of external disturbances. It is an examination of the way in which the system handles the loss of information and an evaluation of the magnitude of disturbance with which the system can cope in order to function in an acceptable manner. The recovery procedures of these networks are also discussed.

The area of interest of this thesis has been discussed and two different types of networks have been chosen for study. The following chapters will discuss each network more fully while the task of simulating each type will be discussed in appendices. The results obtained from the simulation work will be described in a chapter following the discussion of each network.

CHAPTER 2

THE ETHERNET LOCAL AREA NETWORK

The Ethernet Local Area Network is a logical development of the Aloha network(21). This was originally developed for linking terminals situated on remote islands in the Hawaii group via radio link to the central computing site. The basic principle of operation of this protocol was for each remote station to transmit when it had data. If another station was transmitting simultaneously a collision would occur which would result in the central site not acknowledging receipt of the data packets. The remote station would then time out after a randomly selected period and retransmit the packet in the hopes that another collision would not occur. This rather haphazard process worked sucessfully as long as the traffic density on the radio channel was very low. The communication system collapsed at a traffic density of between 15% and 18%.

Various improvements were implemented resulting in protocols such as the Slotted Aloha, Carrier Sense Multiple Access(CSMA), (P)-persistent CSMA, CSMA with Collision Detection(CSMA/CD) and finally CSMA/CD with Binary Exponential Backoff which is the Ethernet protocol. When the IEEE 802 Committee was formed, this was submitted as one of the protocols for consideration. It was accepted with slight modification and is known as the IEEE 802.3

standard. It is described below.

The original Aloha protocol uses free space as its communication medium. This exhibits the property of a bus system in that all listeners can listen simultaneously and only one transmitter can transmit at a time. Transmission delays exist as a result of the geographical distribution of the system.

Ethernet uses a single co-axial cable as its medium. The nodes of the network are transformer coupled to this bus, providing electrical isolation from the bus and isolating any active components from it. This allows nodes to be connected and removed from the bus as well as the application or removal of power from nodes while the network is in operation. Because of the nature of the protocol no special communication procedures have to be invoked when a node is inserted or removed from the system.

The mode of communication is baseband. This makes for simple communication equipment at the individual nodes and requires no expensive, centralised repeater at some point in the system to handle frequency conversion and signal re-transmission. Unfortunately these advantages are offset by the fact that a certain part of the transceiver called the collision detection circuit is difficult to implement as it has to "listen" for other transmissions while its own station is transmitting. Its operation is described later.

Before the packet structure is described, the mechanics of the

protocol will be discussed. It is an asynchronous protocol so there is no single clock source on the bus. The data is manchester encoded thus the signal is self clocking. If a node wants to transmit, it first listens on the bus to see if it is busy. This performs the Carrier Sense function as denoted in the type description of the protocol. If it is then it waits until the bus is free. When it is, it places what is called the preamble to the packet on the bus. This allows the receivers to synchronise with the transmitter before the packet information is placed on the bus. After the packet has been transmitted there is an enforced delay before the bus again becomes available for any node to transmit. This interframe spacing or deference time is inserted to allow the receiver electronics of all the nodes to recover and ready themselves for the receipt of the next packet that is transmitted. The fact that any node may transmit when the bus is free is denoted by the Multiple Access part of the name of this protocol.

An obvious difficulty arises in this protocol when two or more nodes want to transmit at the same time. These nodes have no way of knowing if any other node is wanting to transmit simultaneously, thus a collision cannot be avoided. Collision detection circuitry is thus included in the communications hardware of each node to moniter the bus for such an occurrence. This yields the Collision Detection part of the protocol title.The detection circuitry must be enabled for a period equal to twice the the propagation delay of the bus. To understand this

consider the situation where one node placed at an end of the bus starts transmitting while a second node placed at the other end of the bus starts to transmit just as the signal from the first node arrives. The second node will detect the collision immediately but the corrupted signal will have to travel all the way back to the first node before the latter can detect that a collision has taken place. Thus the elapsed time for the first node is twice the propagation delay. This is known as the collision window.

If no collisions are detected in this period then the node is said to have acquired the bus and no further collisions will occur for the rest of that transmission.

If a collision does occur then the transmitting nodes enforce the collision by sending a jam signal to ensure that all transmitting nodes notice this condition. After this, all transmission ceases and each node that was transmitting re-schedules the transmission after a randomly selected delay. Retransmission is attempted repeatedly in the face of repeated collisions, but as this situation indicates a busy network the period from which this delay is selected is increased for each retry. This characteristic is known as backoff. Obviously this procedure cannot continue indefinitely so there is firstly a maximum period from which the delay is selected and then a maximum number of permitted retries. This last feature is included particularly to avoid unnecessary overloading of what is an already overloaded bus if multiple attempts fail.

The delay period requires a bit more explanation. The total interval from which this is selected is an integral number of intervals called slot times and is equal to two raised to the power of the retransmission number multiplied by the slot time. The actual period is a uniformly distributed random integral number of slot times within this period. The slot time, also known as the collision window is defined to be just greater than the propagation delay plus the jam time. A more rigorous definition of the latter two terms is given in Appendix B.

This fully describes the protocol of the Ethernet local area network. All that remains to be discussed is the structure of the packet that is transmitted over the network. Although this is a bit orientated protocol it is described in byte format for convenience. Its format is shown in figure 2.1. The preamble of 64 bits or 8 bytes has already been mentioned. This consists of alternate one's and nought's providing synchronisation for all the receiver clocks. The sequence starts with a one and ends with two one's in succession to indicate the start of the packet information.

After the pre-amble comes the Destination address followed by the Source address. These two fields are each 6 bytes long.

The next two bytes mark the only difference between the Ethernet and the IEEE 802.3 standards at the Data Link level. In the

Ethernet system this is an identifier used to establish the type of higher level protocol used for communication. The IEEE 802.3 standard uses these bytes to define the length of the data field which follows immediately after this field.

The last field is the frame check sequence. This consists of four bytes and is the cyclic redundancy check. This is computed over all fields except its own, and the preamble. It is discussed further in Appendix B.

As described in the latter appendix, the header information consists of 26 octets and occupies a total of 206 bit times or 20,6 usec. The slot time or collision window is defined to be 51,2 usec. In order for the transmitting node to be certain that it has transmitted its packet without collision, the minimum packet time must be at least equal to the slot time of the network. Otherwise it could have sent its packet and encounter silence on the medium before the collision window has passed. If any further activity occurred within the collision window, the node would be unable to determine whether this was a collision condition or a new transmission, possibly corrupted by noise.

The minimum frame size is thus defined to be 512 bits long plus the pre-amble which makes it 576 bits or 57,6 µsec. This gives a minimum data field of 46 bytes. If less data than this is required to be transmitted, the field must be padded.

At the end of each data transmission, there is a period of

silence on the bus which is called the recovery time. This exists to allow all data link controllers on the network to recover and to allow any reflections present on the medium to die down. This time is defined to be a minimum of 9,6µs.

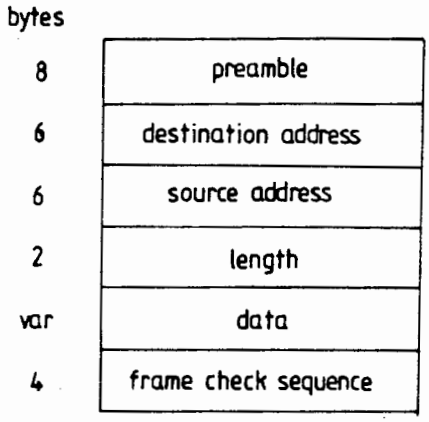

Fig 2.1 Ethernet packet format.

NODE REQUIREMENTS

The above specifications define the performance and protocol existing on the medium of the network. Implicit in these figures is the performance requirement of the node itself. All simulations and performance estimates to date have concentrated on the network performance and assumed that the node is able to supply what is required of it. This section derives a set of figures for a full performance node and then attempts to relate them to the performance of practical hardware.

The most stringent requirements will be imposed on the node when short packets are handled as the packet repetition rate will be

at a maximum. Thus the shortest possible interframe period on the bus is 57,6 + 9,6 = 67,2µs.

High performance nodes have two processors. The first is dedicated to the task of communication and typically would have a communications co-processor associated with it. In an Ethernet system, the latter would form the interface to the network and perform all the low level functions such as serialising/de-serialising, CRC generation/checking, address recognition, data handling and media access. These processors would be on a local bus with their own memory, this sub-system being linked to the main bus of the node. On the main bus would be the node processor together with its memory and the I/O interface linking the node to its environment. The tasks of this processor are all those not directly connected with that of communication.

The process of transmitting data over the network would be initiated by the node processor which would prepare a block of information to be sent over the network. This would contain the data to be sent and instructions to the communication processor telling it what to do. The communication processor would break the data block into smaller blocks for transmission if necessary or pad the frame if it less than 46 bytes, and perform housekeeping functions such as checking the order in which data is transmitted, waiting for acknowledgement and arranging for retransmission of packets if necessary. The communications coprocessor would prepare the data for transmission as described

above.

The process of receiving data is similar to that described above except that the node processor would allocate empty data buffers to the communications processor to be used when incoming messages are received.

As can be seen, a significant amount of work is done by the whole system each time action is taken to communicate over the network. Nabielsky(22) estimates that a processor would have to execute from 1000 to 10000 instructions to process the buffers for communication. The exact number is dependant on the architecture and the size of the packet being transmitted. The above architecture is fairly optimised for communication so the shortest time will be taken for the smallest packet. Assuming that the processors can execute about 1,5 million instructions a second, which is typical for current technology, the time to prepare a buffer of data for transmission can be calculated to be 667 µs. This is approximately 100 times that required by the network if the the full bandwidth is to be realised.

Examining the specifications of currently available hardware this sort of parameter is not specified. It would appear from this lack, together with a dearth of publications on this topic that this aspect of the network specification has never really been considered. An attempt is now made to derive and characterise this parameter from available data.

Specifications are available from four sources(23,24,25,6). The first two deal with the same hardware. This is the Intel SBC 550 Ethernet communications Controller. The manufacturer specifies a performance for the system i.e. communiction board with host cpu and communication software. Although not specified it is assumed that this figure is obtained with a datagram service transferring 1500 byte packets as this will yield maximum throughput. The figure quoted is 70 kbytes/sec.

Nabielsky rewrote the firmware on this board and obtained a throughput of 150 kbyte/sec for 1000 byte packets. He also produced a second figure of 16 kbytes/sec for this board using 46 byte packets.

The time taken for a node to prepare a packet of data for transmission over the network will be termed the processing time. This will now be derived from the throughputs specified by Nabielsky.

Taking the specified data rate for 1000 byte packets, 150 kbytes/sec yields a data bit rate of 1200 kbits/sec. Adding the 26 header bytes to the data bytes yields a raw data rate of

$$1026/1000 \times 1{,}2 = 1{,}2312 \text{ Mbits/sec.}$$

The efficiency of utilisation of the network can be determined for the node by dividing the maximum raw data rate of the node by the data rate defined for the network. Thus,

efficiency = 1,2312/10 = 0,12312 p u

The time to transmit 1026 bytes is equal to the packet size in bits divided by the bit rate of the network, thus:

$$1026 \times 8/10^7 = 820{,}8 \text{ µsec}$$

The processing time can be determined by taking the unused part of the bandwidth, dividing it by the efficiency (or used part of the bandwidth) and multiplying the result by the transmission time, thus:

processing time =

$$((1 - 0{,}12312)/0{,}12312) \times 0{,}8208 = 5{,}8458 \text{ msec}$$

Thus the total time taken to transmit a packet is

$$5{,}8458 + 0{,}8208 = 6{,}6666 \text{ msec}$$

Similarly, the processing time for a 46 byte data packet is 3,948 msec.

This gives two points on a curve describing the processing delay, not sufficient to adequately describe its behaviour. In order to establish its character, it is worthwhile to consider the nature of the load placed on the node as it handles a block of data during the process of communication. It can be divided into two parts. The first is that associated with the preparation of the header of the packet. This is a constant load as the same procedure has to be executed for each packet that is handled. For

transmission the header information has to be prepared and the CRC code must be calculated. During reception the header information must be decoded and the CRC code checked.

The second part is that associated with the handling of the data. The load in this part is fixed for each byte of data handled and is thus proportional to the length of the data field in the packet. There will be a fixed part associated with this, relating to the setting up of the DMA controllers and memory pointers, but this can be included in the load of the first part. Thus the overall load on the node can be characterised by a first order equation and the above two values for processing delay will completely define the equation for that communication controller. The characteristic equation is derived thus:

Slope = (5,838 - 3,948)/(1026 - 64)

= 0,00195 msec/byte

Constant = 3,948 - 0,00195 × 64

= 3,823 msec

Therefore the characteristic equation is

0,00195x + 3,823 msec

where x = no. of data bytes/packet

The Hewlett-Packard (25)reference which came to hand later is a performance brief of the LAN controllers for their scientific computers. It separately defines the performance of the transmitting and receiving sections of the controller. The receiving section exhibits a throughput characteristic that is directly proportional to packet size. From this it is obvious

that the processing delay is constant for varying packet size. The characteristic of the transmitting section is roughly exponential. If the procedure used above for calculating processing time is applied to several points on this curve it will be seen that this confirms the proposal made above. The equations are fully derived in appendix B and are reproduced below:

Processing delay, receive mode = 6.Ø26 msec. constant

Processing delay, transmit mode =

Ø,ØlØ5x + 3,39 msec where x = no. of data bytes/packet

It remains to define a characteristic equation for a state of the art controller. The Intel Ethernet Communication Engine(6) has the highest performance of the controllers examined, but its specification is inadequate for the purposes of this study. An approximate specifiction will be derived. This hardware is an LSI implementation of the earlier Intel controller(23). Both are designed to interface to the Multibus backplane and have identical architectures. The earlier system is based on the 8Ø88 CPU while the later is based on the 8Ø186. The maximum throughput for this system is defined, giving one point on the characteristic curve. The value specified is 3ØØ kbytes/sec. No packet size is specified but it can be safely assumed that this will be for the maximum packet size of 15ØØ data bytes as this will yield the maximum throughput. This throughput corresponds to a processing time of 3,78 msec. Using the similarities described above, the slope of the equation for the earlier controller will

be used for the newer system. This will yield a performance superior to the true capabilities of the system but only an upper bound is being sought for its performance.

Thus the characteristic equation for this network controller is:

Processing time = Ø,ØØ195x + Ø,8Ø43 msec.

This gives a processing time for a 46 byte packet of Ø,9252 msec. This is 13,8 times the maximum required by the network and is better than the range of performances estimated earlier in this discussion.

As mentioned earlier, the minicomputer based system in the receive mode has a processing time of 6,Ø26 msec and is a factor of 89,9 times longer than the maximum required by the network.

This has two implications for the behaviour of the network. The first of these is that a pair of nodes(one transmitting and the other receiving) cannot fully utilise the bandwidth of the network. What happens when more than one pair utilise the network is one of the topics of study in this thesis and will be discussed in the next chapter.

The other implication is what Nabielsky calls the Galloping Bit Syndrome. As is seen above there is a wide disparity in the performace of nodes, even between nodes of the same architecture where one is transmitting and the other is receiving. This can

easily give rise to the situation where the transmitting node has a higher throughput than its receiving counterpart. This will soon cause the receiving node to run out of buffer space or computation time or both, resulting in the loss of data by the receiving node. The transmitting node will be unaware of this situation until some acknowledgement is required of the receiver. The situation of the saturated node allowing valid data to be lost(pass or gallop by) gives rise to the name 'Galloping Bit Syndrome'.

Another aspect of this situation arises when 'n' nodes wish to transmit to one receiver. The receiving node which is fairly evenly matched to any one of the transmitting nodes, will readily become overloaded by the combined output of the transmitting nodes unless the node capabilities and loadings are carefully matched and controlled.

THE SIMULATION

In order to study the effect of these points on the behaviour of the network it is advantageous to build a simulation model of a local area network.

The preferred method(26) of most simulation workers would be to build an analytical model. This would work well for the determination of such values as throughput, transport delay and the effects of buffer size. This method is also good for performance evaluation where the effects of interaction between entities do not need to be taken into account. Execution speed is

good and large models can readily be accommodated.

However, if the above effects are to be studied, a simulation model would need to be developed that can take account of the interaction between entities. The discrete event simulation languge SIMULA was chosen for reasons discussed in Appendix A. This is based on the Algol language which is structured, making it easy to develop and read programs. It is less popular than GPSS but more appropriate for this particular study. Appendix A introduces the concepts of the language, thus allowing a detailed discussion of the operation of the Ethernet model in Appendix B.

The main components of the model consist of the nodes of the network, the packets, the load generator and the data logging entities. The medium, while present does not appear as major component. The reason for this will become apparent later.

All components in the model which have the same characteristic are described by one 'activity'. Different instances are created for each member of the group. Thus the model consists of activities representing the character of each group.

The first group or activity to be discussed is that of the node. It is actually divided into two separate activities in order to more closely resemble the real system. The send and receive functions of a communications processor are logically separate in their operation and only one can function at a time because of

the common(shared) communications medium. This operation is reproduced in the model.

The transmitter activity has a queue which indirectly receives packets from the load generator. Before the packet is transmitted to the receiving node, the bus status is checked. In the process of checking, the collision window is simulated which allows the collision of packets to occur. In this case the enforcement signal is generated, otherwise successful acquisition of the bus is established for the duration of the transmission. All the time delays associated with the above events are simulated. Simulations are performed both with and without the processing delay.

The model has been developed with all the node processing delays incorporated into the transmitter section. This has been done in order to simplify the development of an already complex model. It imposes certain restrictions which for the purposes of this study are not a limitation.

When bus activity occurs a transmit/receive pair must be active so the behaviour of the system is still correctly determined by the transmitting node. A problem arises when two nodes attempt to transmit consecutively to the same receiving node. As the station delay is significantly longer than the recovery or deference time, a physical system would be unable to process the second packet while the receiving node of the model would process the packet correctly. This disparity would be significant if the

behaviour of 'n' transmitting nodes attempting to communicate with one receiver was being studied. However, in this study the network is being studied in a more general sense where one to one communication is typical.

It is possible that more than one node will transmit to the same receiver during the simulation, but this is not a limitation as it is equivalent in behaviour to the transmitters sending to different receiving nodes.

The input parameters of the model are fairly extensive in order to allow maximum flexibility in the use of the model. Those that define the size of the system are number of nodes, number of packets, time of simulation and the size of the buffers. The main simulation parameters are the applied load, transmission time or packet length, and station delay. Other parameters that describe the character of the network are the propagation time, slot time, deference and enforcement times.

The parameters monitered are now discussed. The first two are applied load and throughput. These are determined by respectively counting the total number of packets transmitted and received and dividing them by the times taken to generate and receive them. The total generation period is used for the applied load while the time taken to generate the latter half of the total packets is used for the throughput. This done in order to allow the queue lengths and bus activity to stabilise before the measurement is

done. The throughput is also determined at intervals of one tenth of the total generated packets.

Transport delay is given as three values: average , minimum and maximum.

The number of collisions that occurred are also recorded. These are expressed as the total number of collisions that occurred and the number of collisions incurred by each node.

Other parameters that are monitered are the total number of packets generated and received as well as the number of packets aborted. The number of packets timed in the throughput measurement is also printed. These are printed in order to provide some checks on the validity of the simulation.

This brings to an end the description of the Ethernet network and its model. The simulations performed, together with the results obtained are presented in the next chapter.

CHAPTER 3

PERFORMANCE EVALUATION OF THE ETHERNET LAN

This chapter discusses the tests that were performed on the model of the Ethernet system and discusses some observations made on these results.

NETWORK EVALUATION

The first tests that were performed with the model were done to evaluate the network performance. This means that the station delay value was set to zero and the performance of the network and its protocol was tested.

It was stated earlier that simulation models tend to be rather large and slow. This model is no exception. Execution times were reasonable but the size of the program proved to be a limiting factor in the execution of the simulation. This problem arose in part from the limitation of the hardware design of the computer and in part from the historical development of the language. Early mainframes in this series had a maximum segment addressing range of 64 kbytes. As the development of the language started at this time, this particular implementation of the language was constrained by this. Later machines in this series had a new addressing mode incorporated in their design which circumvented the addressing problem but this version of the language was never upgraded to take advantage of it.

In order to keep the memory requirements of the models developed with this language to a minimum, memory is allocated dynamically. Thus with careful use of the language facilities, the size of the executable module can be controlled. The main method of memory control is to de-refernce all entities that are no longer needed in the model. The area of memory allocated to each instance of an activity that is created in the model is taken from a dynamic memory pool. This allocation is maintained as long as the instance is referenced by one or more reference variables. It is de-referenced by removing the entities from all queues and assigning any reference variables to the dummy entity 'NONE'. The memory is then returned to the pool and is available for re-use.

This technique allows the amount of memory used in the current model to be effectively controlled in all cases except when the network reaches saturation. Under these conditions the queue lengths of packets waiting to enter the communication become so large that the module size rapidly becomes large and reaches the memory limit. In order to obtain results in this situation the number of packets generated must be limited.

This immediately makes the validity of the results questionable because of the reduced number of packets decreasing the possibility that the packet inter-arrival time is truly random. However because the transmitter queues are fairly long, the system is driven by the bandwidth available on the network and not by the arrival of packets at the transmitter queues. The

performance is now a function of the network and not of the load generator. Provided the activity on the bus is fairly stable during the measurement period the results obtained will be a fair reflection of the system performance under saturated conditions.

The first simulation runs were done with the input parameters defining the standard Ethernet conditions. Processing delay was set to zero. Two sets of runs were executed, the first with 46 byte packets and the second with 1500 byte packets. The results are shown in tables 3.1 and 3.2, and figures 3.1 and 3.2 respectively. The figures from two other published performance results are also shown on these diagrams. Bearing in mind that these are analytical models of the network, neither of which are exact, the agreement is good.

The first of these(18) simulates the one persistent CSMA scheme with binary exponential backoff, but the backoff algorithm takes a random drawing for the delay from an exponential distribution rather than a uniform distribution. The second published work(16) simulates the non persistent CSMA scheme with binary exponential backoff so neither has exactly the same characteristics as the model produced here. The performance obtained from their tests is shown in figure 3.1. Performance spread A comes from reference 16 while that of B comes from reference 18. The performance spread of A is quite large as it varies with variation in backoff probability. The latter does not seem to easily relate to that used in the actual Ethernet model. The range covered seems to

have a much lower retransmission probability than that of the true Ethernet.

The figures compare favourably with that obtained from this simulation model.

Table 3.1

Ethernet performance. 46 byte packet. No processing delay

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 0,0576 msec
Minimum propagation time: 0,0671 msec

Applied load	Throughput	Propagation Delay Maximum	Average	Collision Incidence	Packets lost
kbytes/sec	kbytes/sec	msec	msec		
19	18	1,2	0,76	0,031	0
131	129	7,8	0,16	0,243	0
281	289	200,0	2,25	0,409	0
394	326	217,7	1,53	0,346	1
469	361	197,8	2,46	0,302	0
582	408	188,3	2,75	0,218	0
770	433	220,4	1,29	0,175	0

Table 3.2

Ethernet performance. 1500 byte packet. No processing delay

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 1,2208 msec
Minimum propagation time: 1,23 msec

Applied load	Throughput	Propagation Delay Maximum	Average	Collision Incidence	Packets lost
kbytes/sec	kbytes/sec	msec	msec		
153	150	4,4	1,32	0,017	0
306	301	10,4	1,43	0,070	0
537	528	61,6	1,84	0,211	0
766	757	100	2,82	0,402	0
1072	1038	464	36,65	0,697	14
1225	1129	482	75,33	0,655	42
1378	1161	739	91,2	0,612	60

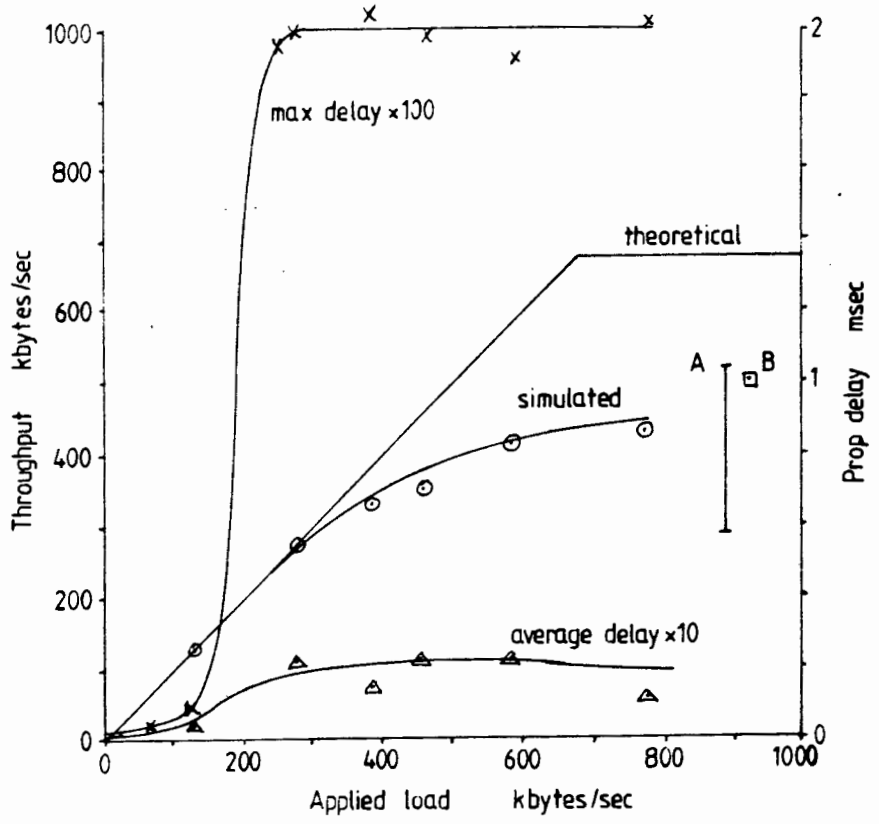

Figure 3.1 Graph of Ethernet performance with no processing delay. 46 byte packet,

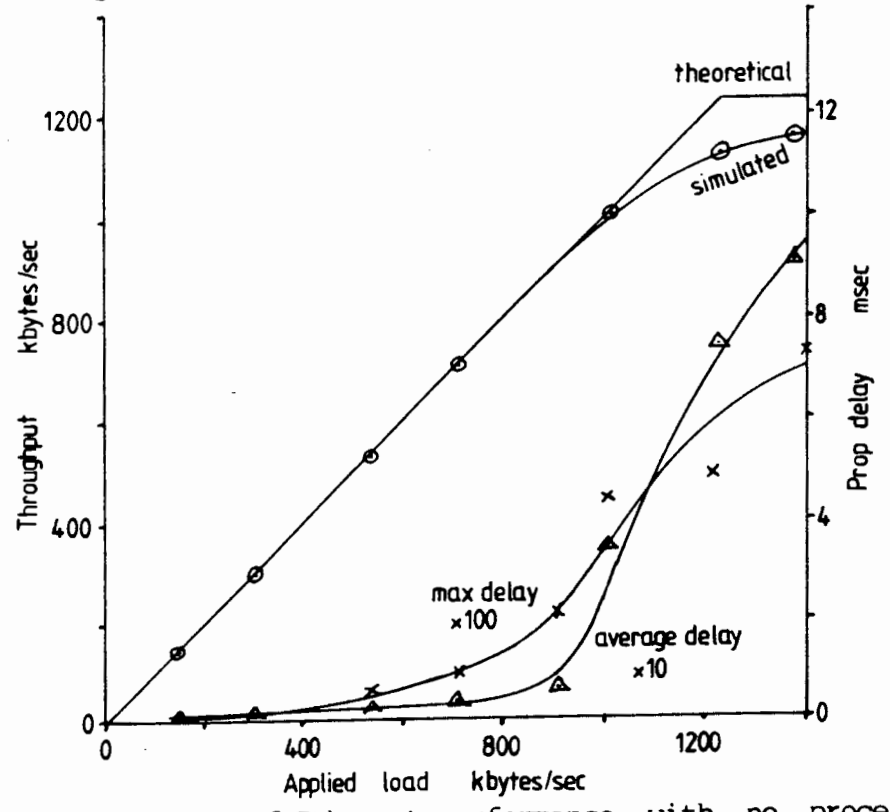

Figure 3.2 Graph of Ethernet performance with no processing delay. 1500 byte packet.

The second set of curves, figs 3.3 and 3.4 show the network performance with the processing delay included in the simulation. In figure 3.3 (46 byte packets), the performance has decreased by approximately 10% producing a performance below that of the figures generally claimed for this system. Propagation delay has increased only by a factor of two despite the fact that the processing time has increased the minimum delay by more than an order of magnitude.

For the longer packets the system performance has actually improved and the propagation delay figures have decreased by approximately one half.

Table 3.3

Ethernet performance. 46 byte packet including processing delay.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 0,0576 msec
Processing delay: 1,52 msec
Minimum propagation time: 1,59 msec

Applied load	Throughput	Propagation Delay Maximum	Propagation Delay Average	Collision Incidence	Packets lost
kbytes/sec	kbytes/sec	msec	msec		
66	65	5,8	1,93	0,097	0
113	111	6,9	2,42	0,176	0
234	229	59,0	4,90	0,178	0
291	247	14,2	5,13	0,099	0
394	281	17,6	5,49	0,014	0
488	289	6,4	5,57	0,006	0
563	289	6,7	5,56	0,004	0

In a control environment where strong constraints are imposed on the network because of the superior performance requirements, the network throughput is limited to approximately 0,3 to 0,5 times

the nominal performance(27,28). The propagation delay is then predictable with sufficient probability to be acceptable in this application. It can be seen that in this region of operation the performance degradation is not significant. It can thus be said that the inclusion of the processing delay in the performance analysis of the network actually decreases the propagation delay time of the system at moderate loads without degrading its throughput.

Table 3.4
Ethernet performance. 1500 byte packet including processing delay.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 1,2208 msec
Processing delay: 3,83 msec
Minimum propagation time: 5,05 msec

Applied load	Throughput	Propagation Delay Maximum	Propagation Delay Average	Collision Incidence	Packets lost
kbytes/sec	kbytes/sec	msec	msec		
153	151	14,2	5,25	0,014	0
306	301	25,6	5,57	0,061	0
537	528	23,2	6,30	0,158	0
766	756	52,7	7,67	0,288	0
1072	1045	248,7	20,87	0,537	3
1225	1147	422,7	50,18	0,533	19
1378	1171	497,8	53,16	0,477	24

Figure 3.5 shows the incidence of packet collisions for the system both with and without processing delay. The curve showing collision incidence without processing delay is what one would expect from a network which is asynchronous in nature. As the load increases, the probability of collision increases. The efficiency of the network decreases and system throughput shows the properties of saturation.

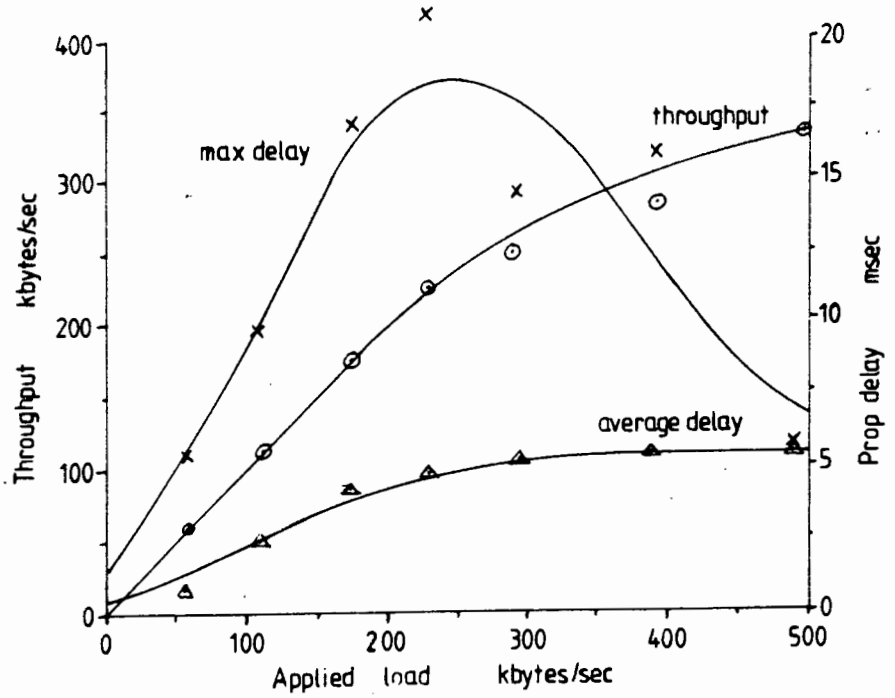

Figure 3.3 Graph of Ethernet performance with processing delay. 46 byte packet.

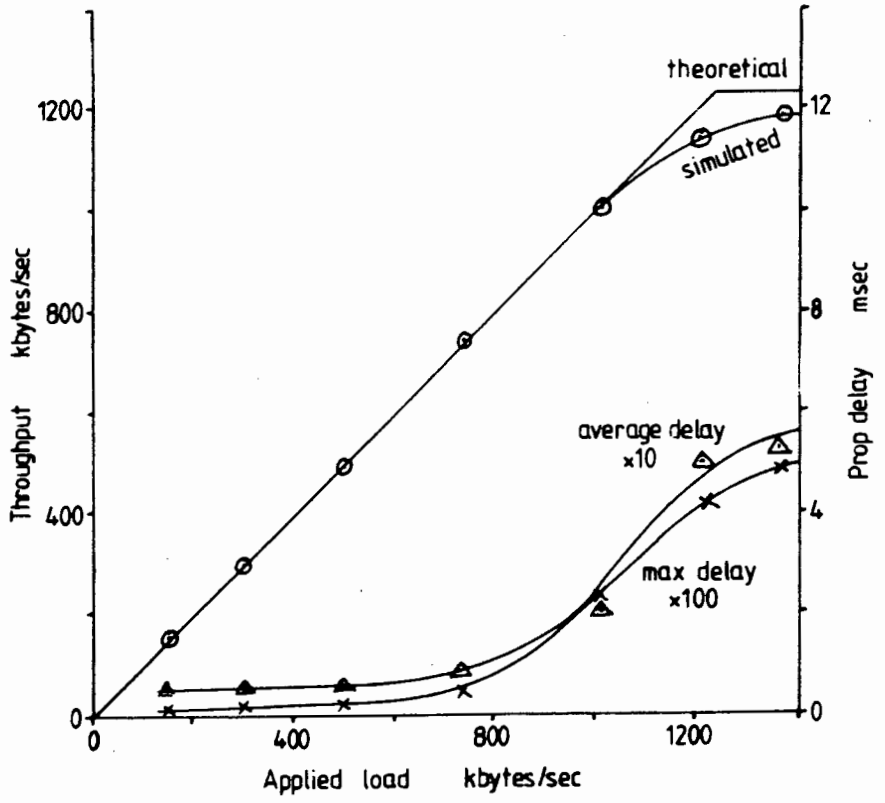

Figure 3.4 Graph of Ethernet performance with processing delay. 1500 byte packet.

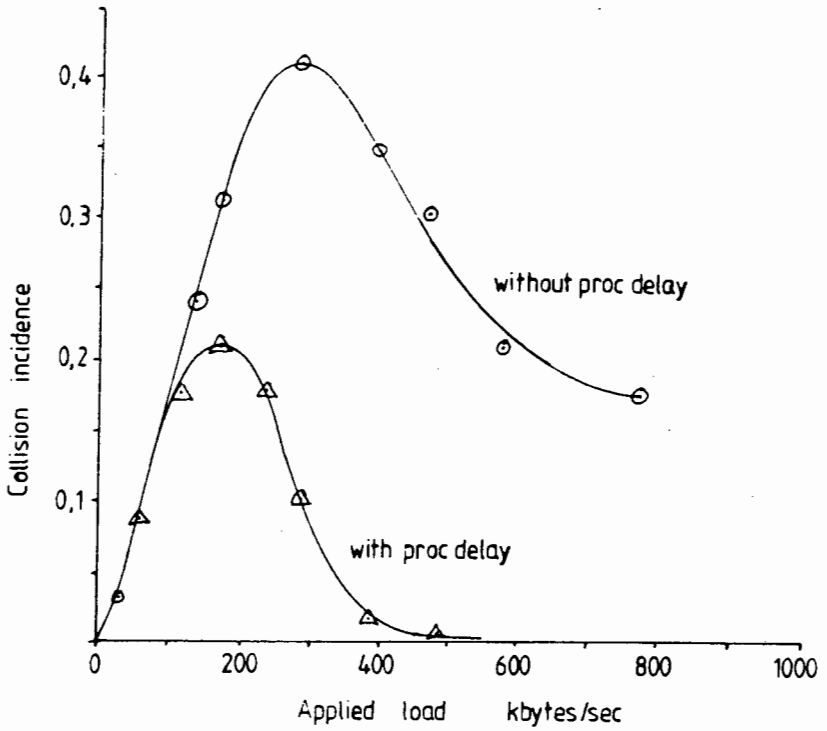

Figure 3.5 Incidence of packet collisions for systems with and without processing delay. 46 byte packets for both system.

The second curve in the figure is interesting. While it has the same shape as the first curve, its peak is considerably lower and its final value is negligeable compared with that of the first curve.

To find a possible explanation for this, it would be expedient to look at a graphical representation of the utilisation of the network bandwidth by a single node as is shown in fig 3.6. Here it can be seen that the node uses the bus for the duration of the packet transmission time and releases it for a time equal to the processing time. It can be said that the node is voluntarily restricting itself to transmit in fixed time slots and leaving the bus free for use by other nodes in the remaining time slots. This imposes a voluntary discipline on the network which is

reflected in the reduced disorder at higher throughputs.

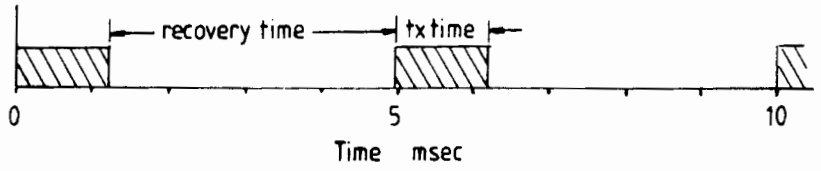

Figure 3.6 Utilisation of system bus by a single node. Times are for 1500 byte packet.

A possible scenario of the bus activity is now described. At light loads, each node sends its packets at widely spaced intervals with a minimum interval equal to the station processing time. Because of the low bus utilization and well spaced transmissions the probability of collision is small. Each node thus generally finds the bus free when it wants to transmit so few collisions occur.

As the system load increases the initial collision rate also rises with the increased utilisation of the bus. As the collision rate increases further, the nodes begin to saturate and allocate transmissions to their voluntary time slots. Once a node has established a time slot, it times its transmissions for those slots and thus avoids having to contend for the bus. This implies that the collision rate should increase again as the available time slots get used up. This has not been found to be the case. An explanation can be offered as follows. As the load on the system increases it is inevitable that more than one node will share the same time slot. The expected result is a rise in the number of collisions. This does not occur because a factor has

not been taken into account. It is that the slot times as determined by the bus constants do not correspond to the slot times determined by the nodes because a non integral relationship exists between the two. Thus the bus could become available for traffic before a slot determined by a node is due to start. This would allow a waiting node access to the bus making the bus busy when the slot time of the former node is due to start, causing it to hold off. Obviously this scenario will not always be valid and contention will still occur.

Table 3.5

Ethernet performance. 46 byte packet. Processing delay included.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 0,0576 msec.
Processing delay: 1,52 msec.
Minimum propagation time: 1,59 msec.

Buffer size	Throughput	Propagation Delay	
		Maximum	Average
packets	kbytes/sec	msec	msec
4	289	6,30	5,52
8	289	12,7	11,8
12	289	19,1	17,9
16	289	25,4	24,0

The final table, 3.5 shows the performance of this network as the size of the communications buffer is varied. There is no change in the throughput of the system but the propagation delay varies in direct proportion to the buffer size.

This completes the initial discussion on the results of the Ethernet simulation. They are discussed further in Chapter 6.

CHAPTER 4

THE MAP LOCAL AREA NETWORK

This protocol was first formulated in about 1980 by the General Motors Group in an effort to establish a standard for use in the motor car manufacturing industry. The benefits of this standardisation would be that equipment used in the assembly lines of automobile manufacture which are supplied by different manufacturers could be directly interconnected with each other. This would obviate the need for expensive and time consuming development of special interfaces between equipment supplied by different vendors. Communication rate would also be much improved. This effort has gained strong support both from work cell users and from the vendors of this type of equipment. The acronym MAP stands for Manufacturing Automation Protocol.

The effort started at a time when neither the technology nor the networking standards were available. Discussion groups for the formation of standards were already in existence so the MAP working group used the draft standards wherever possible and adopted standards when they became available. The development of this protocol has thus been one of considerable evolution and metamorphosis as the standards have become available and technology has improved. To date several systems have been demonstrated and a few pilot plants are being built.

This network is based on a broadband CATV system which is becoming the preferred type of communication medium used in the automotive manufacturing environment(29). This offers a number of advantages, namely a universal interconnection system among all computerised services and a common carrier for digital messages, audio and TV signals.

The initial development of this network commenced on a 1Mbit/sec broadband system. This was subsequently upgraded to a 5Mbit/sec system while development is now proceeding with a 10Mbit/sec network, taking advantage of the latest technology.

The broadband system can only carry data in one direction for any particular band, thus the ability to be able to broadcast information from any node is accomodated by using one band for all transmitters to transmit to a headend which is placed at one end of the network. This rebroadcasts the data on another band to all the receivers attached to the network.

The headend remodulator which performs a function similar to that of a repeater is a costly additional item that is not normally found in the TV environment. A conventional repeater takes the incoming signal, amplifies and frequency translates it in the process of retransmitting it. This is a noisy process which decreases the signal to noise ratio. Any distortion present in the signal passes uncorrected through the repeater. The remodulator demodulates and reconstitutes the signal, thus

eliminating any distortion. It performs an error check on the signal before remodulating and transmitting it. This process also minimises noise on the system.

The protocol itself is that of the token passing type. When a node receives a special packet called the token, it is allowed to transmit data over the network until a certain time has elapsed or until it has no more packets to send. It then passes the token to the next station in turn which then transmits its packets over the network. This continues until all transmitting nodes have had a turn when the token is passed again to the first node on the network. Thus the token is passed around in a logical ring which exists on a physical bus system.

At startup or on loss of token, the token originates from the node with the highest physical address. It is then passed to the node with the next lower address. This is repeated until the node with the lowest address passes the token to the node with the highest address. The process of determining the token passing order is determined at startup or on loss of token by a contention process. This process is more fully described in Appendix C.

This protocol appears to be much simpler than that of the Ethernet system but in practice the communication interface of the nodes turn out to be much more complex and thus considerably more expensive. The modem linking the node to the co-axial cable,

and the taps required for the connection are expensive as a result of their complexity and quality.

The first tests done with this system have proved to be dissappointingly slow. Thus an Enhanced Performance Architecture (EPA) has been defined which uses the carrier band technique and only three of the seven layers of the OSI reference model in an attempt to improve the network performance. The Carrier band technique is similar to the baseband system in that no carrier is used. Phase coherent frequency shift keying is used to encode the information onto the network.

Gateways to vendor networks in work cell environments have been developed to link to existing hardware. This has proved to be a complex and costly solution which is slow in execution.

One reason for the poor performance of the network is obvious. This is because of the complexity of the gateways used. At each gateway the message has to be decoded to the application level and then re-coded as required by the new protocol - a computation intensive process which is thus time-consuming and slow. If performance is to be good gateways must be avoided.

A second reason for poor performance has become apparent in this study of the behaviour of the network. It will be discussed later.

PACKET FORMAT

This bit orientated protocol is also described in byte format for convenience. Its format is shown in figure 4.1. This protocol is asynchronous in the transfer of information from the node to the headend. The packet transmission thus starts with a preamble to allow the receiver of the headend to synchronise with the signal from the current token holder. It is rebroadcast by the headend on the forward channel to all the receivers of the nodes in the network. The headend transmits continuously, sending a special code sequence which denotes 'silence' when no information is being handled on the network.

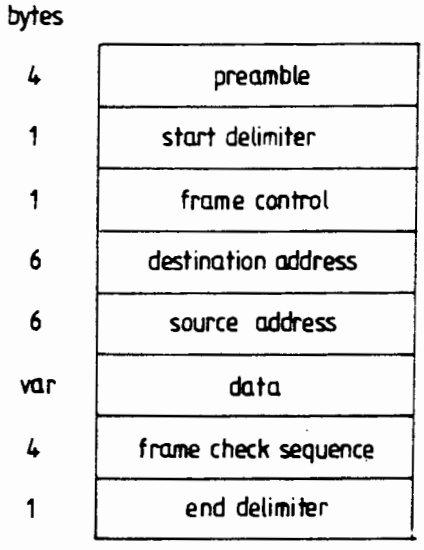

Figure 4.1 Packet format of the IEEE 802.4 protocol.

In the descriptions that follow each field of the packet has a special code sequence or set of codes that do not always directly translate into a binary pattern. They are not presented here as they are not pertinent to this study. They are fully described in section 4 of reference 13.

The preamble is defined to have a minimum duration of two microseconds to allow stations to recover from the previous packet received. In the case of the 10 Mbit/sec network the above requirement necessitates the presence of four bytes of preamble as an integral number must be sent.

Following the preamble is a one byte field called the Start Delimiter which defines the start of the packet information. This is matched at the end of the packet by the End Delimiter.

The Frame Control field defines the type of packet being sent - whether it is a low level control packet, network management data, or information transfer between nodes.

Following these come the Destination and Source address fields. These can be either 16 or 48 bits long. In this simulation the 48 bit fields are used.

The Data field comes next which can be from 0 to 8174 bytes long. Its format is defined by the upper layers of the protocol but is transparently handled as data at this level.

The last frame before the End Delimiter is the Frame Check Sequence. It is four bytes long and is more fully described in Appendix C.

As described in the above appendix, the header information consists of 23 bytes and occupies a total of 184 bit times or 18,4 microseconds. The inter-packet recovery period is included

in the preamble so the maximum time available to a node for the preparation of a packet with one byte of data is 19,2 microseconds.

NODE REQUIREMENTS

The description above pertains to the protocol that is defined for the medium that exists on the bus. As with the Ethernet system, the designers have assumed that it is possible for a node to supply information as required by the network. No attempt has been made to determine what is required of a typical node.

Having described the network protocol it now becomes necessary to determine the requirements of the node in order to be able to fully utilise the network capability.

The specifications of the IEEE 802.4 protocol allow a minimum data field length of one byte in the packet. This is however unrealistically short as no upper level protocol information can be included. A more realistic figure would be about 10 bytes if the first three or four layers only of the OSI reference model were being used. This gives a minimum transmission time of 25,6 usec per packet. No minimum time is specified between packet transmissions so this figure would be a reasonable lower bound on the computing time available for a MAP node to process data for a packet. This is approximately 2,6 times more severe than that required for an Ethernet node.

From the figures determined in Chapter 2 the time needed to perform this function by a system realisable with current hardware would be approximately 260 times longer than that available.

Assume as before that a high performance node would have two or more processors where one would be used for the communication process while the others would be used for the main processing functions. The communications processor would again do all the low level communications functions such as serialising/deserialising, CRC generation/checking, address recognition, data blocking and media access. The latter function would be shared with the modem which contains analogue circuitry comprising a narrowband transmitter and modulator in the transmitting section and a narrowband receiver in the receiver section.

The architecture would be similar to that described for the Ethernet system, i.e. the communications processor together with its communications co-processor would have a local bus. On this bus would be the local memory, part of which would be dual port with the second port linked to the main node bus. The latter bus is common to the main processors, allowing interprocessor communication, including communication with the communications processor.

The communication process would be initiated by a node processor

which would prepare a block of information to be handled by the communication network. This would contain the data to be handled as well as instructions to the communications processor telling it what to do. In addition to the functions mentioned above, this processor in the transmitting process would perform other functions such as waiting for acknowledgements, arranging retransmission of packets and restoring the token in the event of communication failure.

The receiving process would additionaly check that the packets are delivered to the host in the correct order.

Currently available hardware for the MAP network is extremely limited. Indeed most manufacturers have only preliminary information which certainly does not include any performance figures. Comparing the architecture of the Intel MAP communications board with that of their Ethernet board, it is seen that their architecture is identical, and except for the communications co-processors, the integrated circuit types are identical. From this it would be fairly safe to assume that their performances would be similar. This assumption was made when all the performance figures were produced. Subsequent to this, Intel have produced some estimated performance figures(30) which actually show that the performace is only approximately two thirds that of the Ethernet system. The original figures will still be used as this performance will soon be reached, either by this manufacturer or some other producing similar hardware.

The characteristic equation of the MAP node has thus been taken to be the same as that of the Ethernat node. This has the added advantage that the network performances can be compared where the node architectures are identical. This allows a comparison of the demands made on the nodes by the network protocols.

In view of the the above figures it can be seen that the nodes on this network are also unable to fully utilise the bandwidth available to them on the network. Nodes of differing performance communicating with each other can also potentially suffer from the Galloping Bit Syndrome if the performance of the transmitter is greater than that of the receiver. The situation of 'n' nodes wanting to transmit to one receiving node does not present a difficulty in this network as only the token holder is permitted to transmit. All other nodes would have to wait their turn to own the token before transmitting.

THE SIMULATION

The simulation model that was developed for the Ethernet system was used as a basis for this model. Those parts specific to the modelling of that system were removed and new components simulating the MAP protocol were inserted.

As with the Ethernet model the main components of the model are the nodes, the packets, the load generator and the data logging entities. The medium is not apparent in the model as its characteristics do not influence the behaviour of the network

beyond the determination of the network constants.

This model does not simulate the acquisition of the token at start up or the recovery of the system from a lost token as these procedures, although probabilistic in nature are fairly predictable in behaviour. Their simulation would add significantly to the complexity of the model without adding appreciably to the value of the simulation.

The node entity retains its division into two activities. In fact the receiver activity is retained from the first simulation.

The transmitter activity also has an input queue which indirectly receives its packets from the load generator. Before the activity can transmit any packets that are in its transmit queue, it must wait until it has received the token. Once the latter is in its possession, it can transmit over the network until its queue is empty or until the token holding time has elapsed. The token must then be passed to the next node.

Parameters for the model are determined in accordance with the specifications laid out in the IEEE 802.4 standard and are determined for a network having a size as determined by the performance of the transmitter and receiver sections of the modems and by the physical properties of the cable. This system is comparable in size to that of the Ethernet network. The processing delay is incorporated into the model so that its

effects on system behaviour can be studied.

As with the previous model all the processing delays have been incorporated into the transmitting section while the receiver section of the node exhibits none. This has again been done for ease of programming and has no side effects as a transmitting node can transmit to only one receiving node. If the data handling capabilities of the transmitting and receiving nodes are disparate with the tranmitting node having the higher throughput, the Galloping Bit Syndrome would be evident. This is however a limitation of the network rather than of the simulation model.

The input parameters for the simulation model are the number of nodes, number of packets, size of transmission buffers and time of simulation. Other parameters that determine the character of the network are the propagation time and slot time(different from the Ethernet slot time - see Appendix C).

The main simulation parameters are applied load, throughput, transmission time(packet size) and processing delay.

The method of data collection and reporting are identical to that for the Ethernet simulation as the same routines are used in both models.

The simulation runs executed, together with the results obtained are described in the next chapter.

CHAPTER 5

PERFORMANCE EVALUATION OF THE MAP LAN

This chapter describes the simulation runs that were performed with the MAP model. These results are presented and some observations are made on the behaviour of the network.

NETWORK EVALUATION

The first test runs that were done with the model were performed with the processing delay set to zero. This allows the performance of the protocol to be evaluated. These tests were then repeated with the processing delay set equal to that of the commercially available Ethernet nodes. This value was chosen firstly because it is the best available figure and secondly because it enables the performances of the two networks to be meaningfully compared.

RESULTS

The results of the first test runs are shown in tables 5.1 and 5.2. These are presented graphically in figures 5.1 and 5.2. As for the results obtained with the Ethernet model both the actual and the theoretical figures are given, but in this case the two curves are so similar that they are indistinguishable from each other. It can be clearly seen that in this case the synchronous protocol of the MAP network is significantly better

than that of the asynchronous one of the Ethernet system.

Table 5.1

MAP performance. 46 byte packet. No processing delay.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 0,0544 msec.
Token holding time: 20 msec
Minimum propagation time: 0,545 msec.

Applied load	Throughput	Propagation Delay Maximum	Average
kbytes/sec	kbytes/sec	msec	msec
206	203	0,59	0,18
281	277	0,65	0,19
394	388	0,94	0,24
469	461	1,20	0,28
582	573	2,15	0,39
770	754	4,65	0,91
939	838	23,0	1,94

Table 5.2

MAP performance. 1500 byte packet. No processing delay.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 1,2208 msec.
Token holding time: 20 msec.
Minimum propagation time: 1,22 msec.

Applied load	Throughput	Propagation Delay Maximum	Average
kbytes/sec	kbytes/sec	msec	msec
153	150	3,51	1,40
306	301	5,86	1,52
537	528	13,2	1,82
766	754	16,7	2,39
1072	1057	52,7	6,80
1225	1200	118	17,8
1378	1231	182	41,3

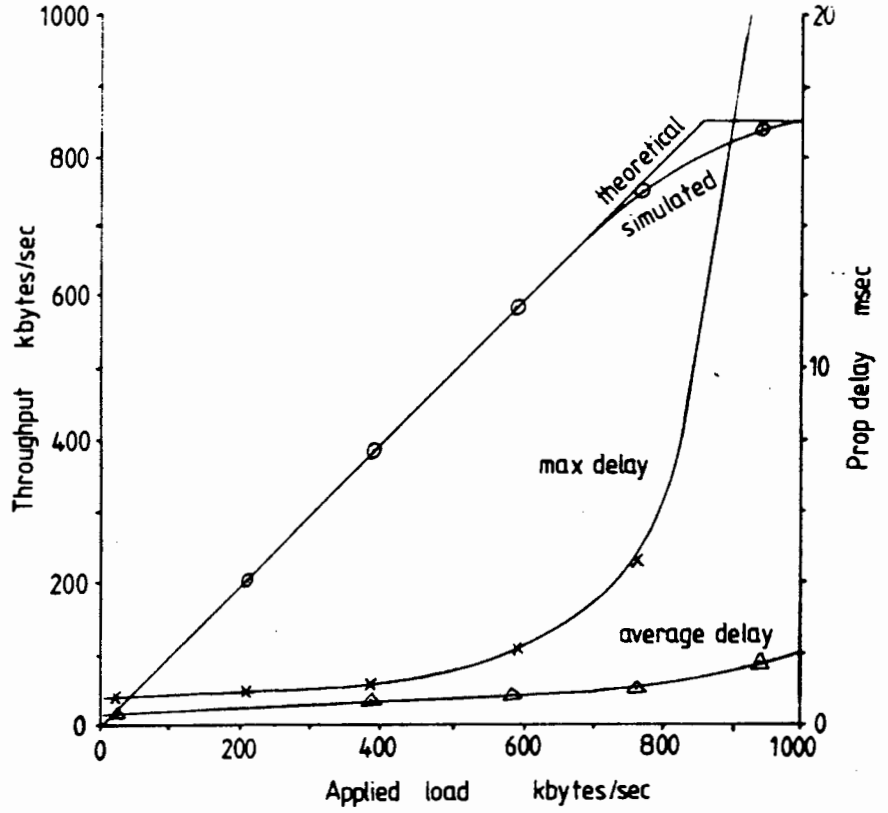

Figure 5.1 Graph of MAP performance excluding processing delay. 46 byte packet.

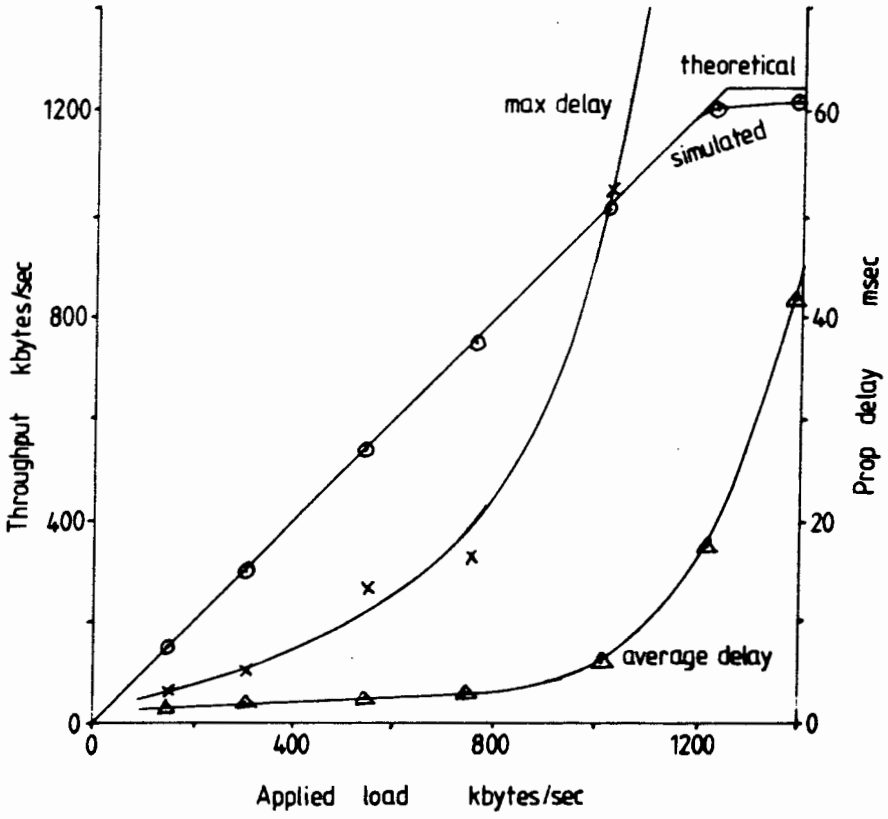

Figure 5.2 Graph of MAP performance excluding processing delay. 1500 byte packet.

The second set of test runs include the node processing delay. Their results are tabulated in table 5.3 and 5.4 while they are presented graphically in figures 5.3 and 5.4. The performance is unexpectedly poor.

Initially it was thought that there was an error in the simulation model. Diagnostic output statemants were added to the program to check its sequence of operation and find the source of the problem. Analysis of the output produced showed that the simulation was working correctly and that the poor performance arose from the token passing protocol interacting with the nodes. The degradation occurs because the network performance is related very closely to that of the node performance. In fact it is equal to the performance of the individual nodes.

Table 5.3

MAP performance. 46 byte packet. Processing delay included.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 0,0544 msec.
Processing delay: 1,52 msec.
Token holding time: 20 msec.
Minimum propagation time: 1,57 msec.

Applied load	Throughput	Propagation Delay Maximum	Average
kbytes/sec	kbytes/sec	msec	msec
9,3	9,2	3,75	1,59
18,8	18,5	14,1	1,73
28,2	27,7	18,1	1,92
37,5	31,7	189	48,6
51,7	31,4	189	59,2
65,7	31,4	189	108

To understand how this situation arises, it is necessary to consider the operation of an individual node(31). Referring to figure 3.6 a diagrammatic representation of bus utilisation with respect to time is illustrated. The basic repetition interval of utilisation is equal to the processing delay of the node. This is the time it takes the node to prepare a packet of data, send it over the network and ready itself to the handling of the next packet. Only a small portion of this time is actually spent utilising the bus. The rest of the time the bus is unused. Because only one node holds the token at a time no other node is able to utilise the bus in this dead period. The network thus becomes a logical star with the token holder being the hub, transmitting packets to the receiving nodes at the periphery of the star.

Table 5.4

MAP performance. 1500 byte packet. Processing delay included.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 1,2176 msec.
Processing delay: 3,82 msec.
Token holding time: 20 msec.
Minimum propagation time: 5,03 msec

Applied load	Throughput	Propagation Delay Maximum	Average
kbytes/sec	kbytes/sec	msec	msec
76	75	15,4	5,18
153	151	17,7	5,48
229	226	43,3	6,13
306	301	50,0	7,33
382	373	206	91,9
459	355	461	251
613	351	738	334

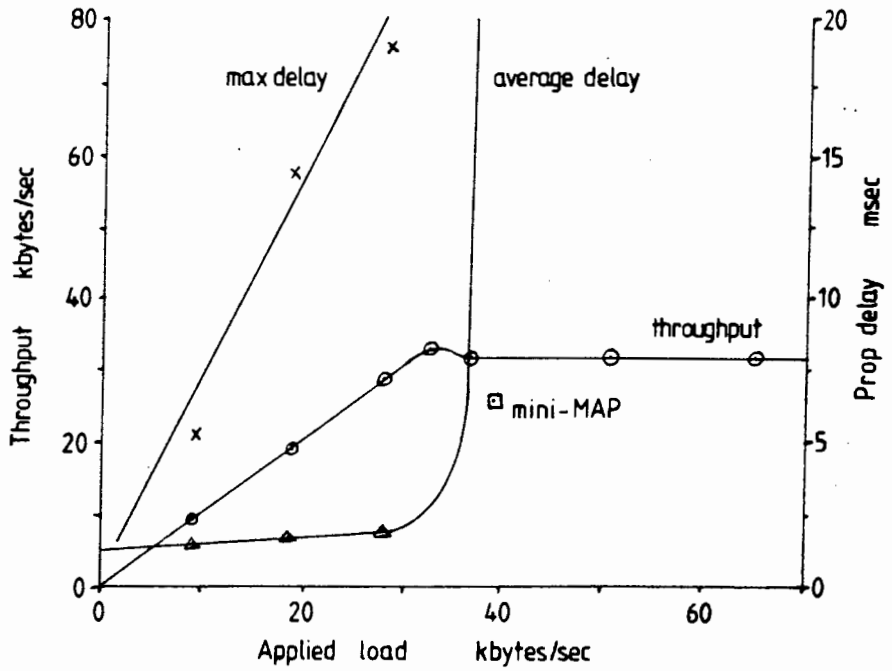

Figure 5.3 Graph of MAP performance. Node processing delay included. 46 byte packets.

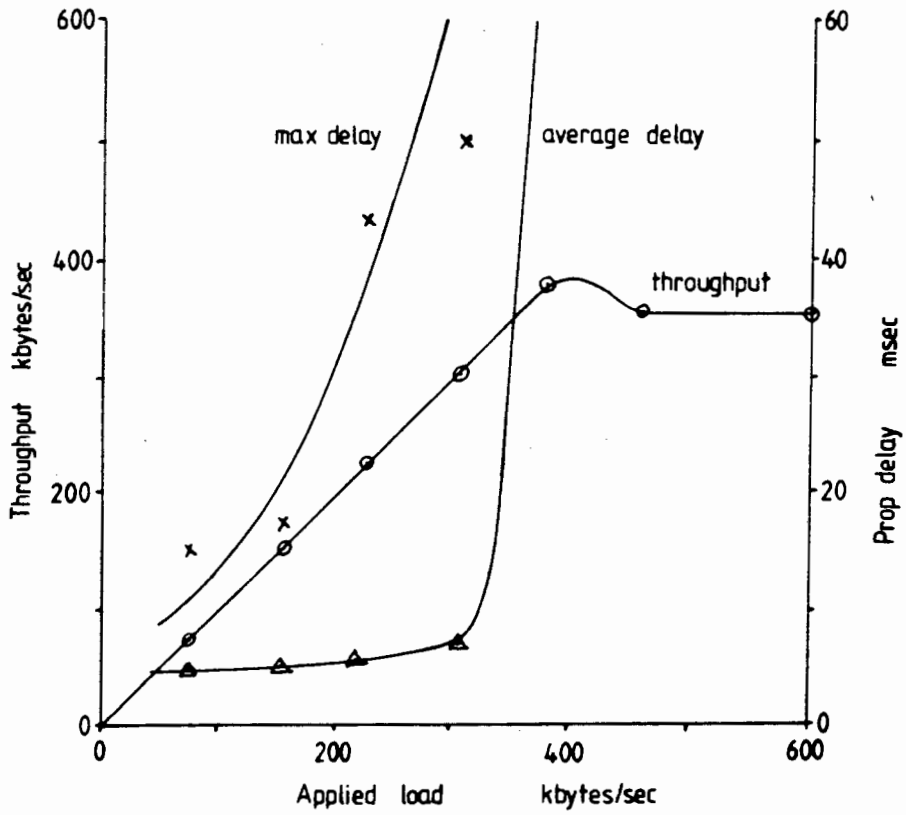

Figure 5.4 Graph of MAP performance. Node processing delay included. 1500 byte packets.

The effective bandwidth of the network at any instant is the bandwidth of the token holder at that instant. The bandwidth of the network then becomes the average of the bandwidths of the individual nodes that are participating in the transmission process. (Some nodes in the network could be listeners only and will thus not contribute to its effective bandwidth).

From this it can be deduced that for any network adhering to the standard protocol, the bandwidth of the network is less than or equal to that of the node with the largest bandwidth.
Some months after these results were produced, they were verified when the performance of a pilot commercial network was obtained that was running with the new mini-MAP EPA architecture(32). This figure is shown as a single point on figure 5.2. This shows excellent correlation with the current results.

Note that the propagation delay times of table 5.3 take fixed values when the system is in saturation. This is a limitation of the model of the MAP system and is not a true reflection of the system behaviour.

Lastly a set of test runs were executed where the communication buffer size was varied. The results are shown in table 5.5 where it can again be seen that buffer size has no effect on network performance other than to proportionately vary the propagation delay.

Table 5.5

MAP performance. 46 byte packet. Processing delay included.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 0,0544 msec.
Processing delay: 1,52 msec.
Token holding time: 20 msec
Minimum propagation time: 1,57 msec.

Buffer size	Throughput	Propagation Delay	
		Maximum	Average
packets	kbytes/sec	msec	msec
4	56,12	32,8	24,3
8	56,12	65,6	50,0
12	56,12	98,5	76,2
16	56,12	131	103

Further discussions on the behaviour of this network and a comparison with the Ethernet system is given in the next chapter.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

Chapters 3 and 5 presented the results obtained from the tests performed on the models of the two types of networks and offered an explanation of the effects observed in them. This chapter compares the behaviour of the two networks and suggests ways of improving the performance of the MAP system.

As discussed in chapter 3 the Ethernet system exhibits a node to node performance that is poor compared with that of the theoretical performance of the network. The reason for this is because of the extremely high performance demanded of the communications and node processors in each node. This performance is considerably in excess of the capability of currently available hardware.

Because of the nature of the operation of the protocol however, the network performance is not compromised because nodes are able to use the network in the times when other nodes are preparing information for transmission. As discussed in chapter 3 the nodes voluntarily organise themselves into their own time slots on the network thus minimising collisions and increasing throughput. In this way the processing delay of the node actually improves system throughput beyond what would be expected or predicted when taking into account the processing delay.

The increased order on the bus resulting from this results in the transport delay time being less than that measured on the model that has no processing delay. The spread of propagation delay is much smaller thus making the network less indeterminate than is expected.

In contrast, the MAP network is capable of demanding an even higher performance from its nodes to realise its full potential. For a node with the same capabilities as an Ethernet node, by definition it must offer the same node to node performance as that of the Ethernet system. However the network performance is effectively reduced to that of the node that holds the token. The reason for this is that only the token holder is permitted to transmit. Thus no other node wishing to transmit is allowed to make use of the bus free time between transmissions as is the case in the Ethernet system. This is rather a serious drawback that will limit the performance of the system for the forseeable future unless some method is found for improving its performance.

Studying the behaviour of the transmitting section of the node as it is currently required to perform, it can be seen that it undergoes a short period of intense activity and then experiences a relatively long period where it is not permitted to do anything. This is very inefficient use of a hardware resource which is normally avoided in order to maximise its use. With the current relatively low cost of hardware this bad utilisation

could be excused because of the benefits provided by a network. Unfortunately the performance demanded of the hardware is so high, in fact currently unrealisable that the cost of the communications hardware that would provide a reasonable performance is prohibitive. Thus at best this protocol yields a costly system that has only a moderate performance. In comparison the performance demanded of the Ethernet hardware is much less severe as it only utilises the bus when it has data to transmit and it does not compromise the performance of the system if it has a low performance. Assuming that the bus loading is not high, the activity of the communications processor is determined by its communication load.

In looking for a way of improving the performance of the token passing bus, it is obvious that if the load on a node could be spread more evenly, this would both improve the system throughput and reduce the peak load on the node. The basic protocol cannot be altered so any changes must be within the options provided by the IEEE standard.

Currently the protocol allows the token holder to transmit packets for a predetermined time before passing the token on. The Ethernet system by contrast allows a node to transmit a packet and release the bus for the use of other nodes while the first node prepares the next packet of data for transmission. This results in much better use of the processing power available at a node.

A similar principle could be introduced in a token passing protocol if each node were only allowed to transmit one packet before passing on the token. Only that hardware directly associated with the bus need be capable of sustaining the high data rates of the bus while the communications processor hardware could be of lower performance. This could process data for communication over the bus while other nodes are transmitting on the network as is the case in the CSMA/CD network. This implies that the hardware associated with the handling of the token is done by the high speed dedicated logic on the bus interface rather than by the communication processor, as participation by the latter would cause a loss in system performance.

In order to verify this, the model of the MAP network was modified so that the node could pass on the token as soon as the transmission of a data packet is complete. The parameters that were used in the previous simulations are used in this run except that the token holding time was made equal to the packet transmission time. The results of this test run are shown in table 6.1 while they are shown graphically in figure 6.1. It can be seen the the performance improves by a factor of approximately six. This is for a ten node system. As the number of nodes increases the token rotation time will increase allowing further performance improvements. This benefit will cease when the node processing time becomes equal to or greater than the token holding time. This is illustrated in table 6.2 and figure 6.2.

Table 6.1

MAP performance. 46 byte packet. Processing delay included.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 0,0544 msec.
Processing delay: 1,52 msec.
Token holding time: 0,050 msec.
Minimum propagation time: 1,57 msec.

Applied load	Throughput	Propagation Delay Maximum	Average
kbytes/sec	kbytes/sec	msec	msec
93	92	12,1	2,99
197	190	12,3	5,38
291	238	10,8	5,24
404	281	7,70	5,67
498	293	7,19	5,65

Table 6.2

MAP performance. 46 byte packet. Processing delay included.
Variable number of nodes.

Number of packets generated: 2000
Packet transmission time: 0,0544 msec.
Processing delay: 1,52 msec.
Token holding time: 0,050 msec.
Minimum propagation time: 1,57 msec.

Number of nodes	Throughput
	kbytes/sec
10	294
20	475
30	493
40	560
50	537
60	522

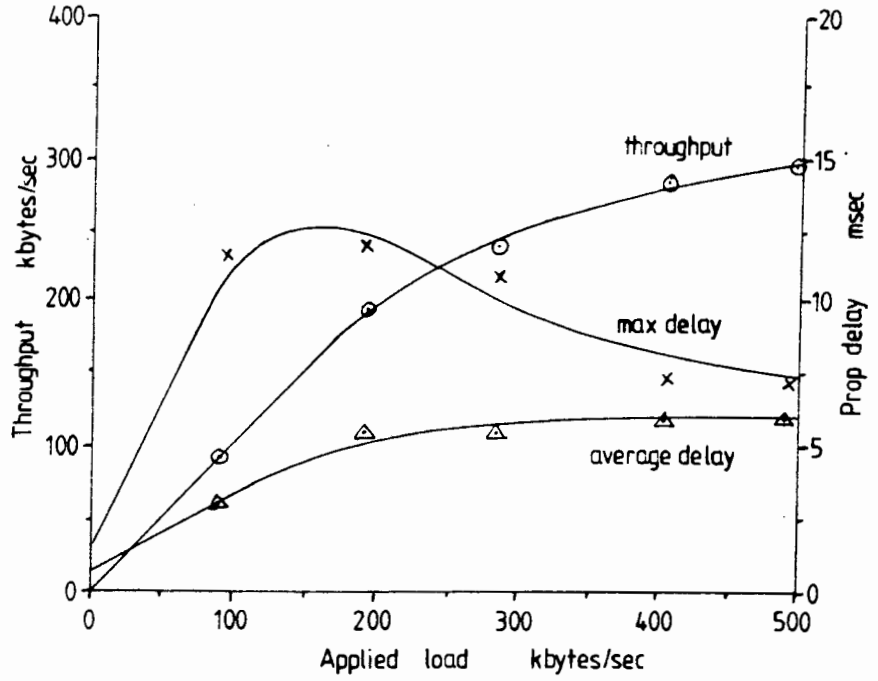

Figure 6.1 Graph of MAP performance with processing delay included. Token holding time is just less than packet transmission time. 46 byte packets.

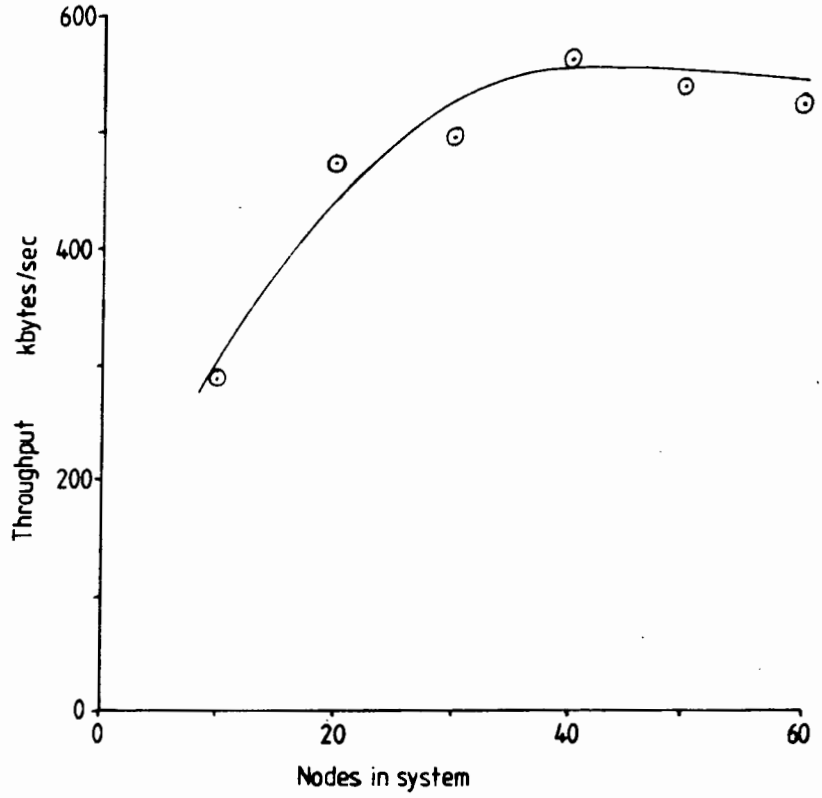

Figure 6.2 Graph of MAP performance with processing delay included. Token holding time less than packet transmission time. Variable number of nodes in network. 46 byte packets.

Table 6.3

MAP performance. 46 byte packet. Processing delay included.

Number of nodes: 10
Number of packets generated: 2000
Packet transmission time: 0,0544 msec.
Processing delay: 1,52 msec.
Minimum propagation time: 1,57 msec.

Token holding time	Throughput
pkts/period	kbytes/sec
1	293
2	56
3	38
5	36
7	34
9	33

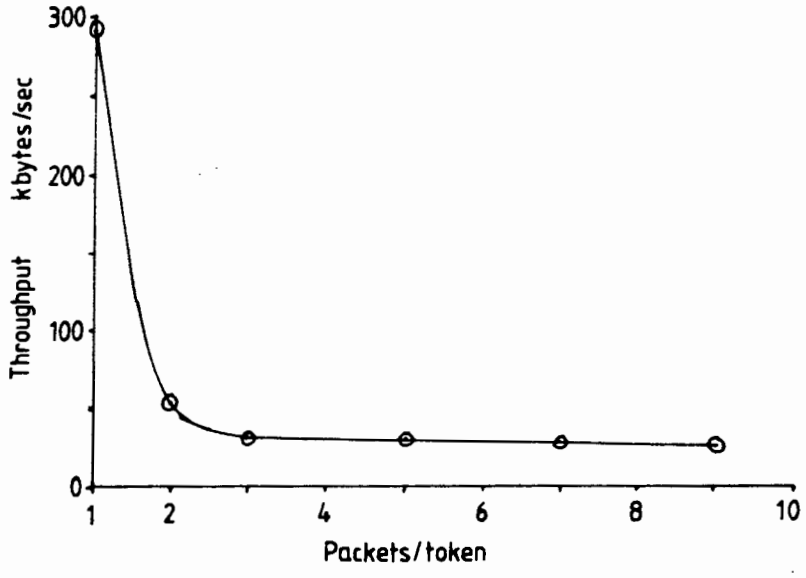

Figure 6.3 Graph of MAP performance with processing delay included. Variable token holding time. 46 byte packets.

A series of runs were executed where the total number of nodes in the network was varied. Although the results seem to indicate a slight fall in performance after a maximum is reached, it is more likely that this value will be constant. This maximum is reached with relatively few nodes in the network. Table 6.3 and figure 6.3 indicate how the network performance varies with variation in the number of packets transmitted each time the node has the token. There is an extremely rapid fall off with increase in the number of packets transmitted. This would be expected because of the large increase in time required to handle the extra packet beyond that of the packet transmission time.

This idea of the above discussion can be extended if the memory capacity of the high performance section can be increased so that the data of several packets can be stored consecutively. The packets could be prepared by the communications processor in the absence of the token and placed in the high speed buffer. This would be flushed by the high speed hardware as soon as the token is received allowing several packets to be transmitted at the full capability of the bus. The preparation of the buffer will take some time so this procedure would only benefit the large network where the token rotation time is long. Obviously this technique would preclude the IEEE 802.4 option of allowing remote stations to acknowledge the receipt of packets. The process of acknowledgement would have a significant impact on the performance of the network as it would require some intelligent response from the remote node. This would require a processor to

execute an algorithm which as has been shown will be too slow if even moderate performance is to be achieved.

The Ethernet system has established itself in the office automation environment and in the Boeing TOP protocol which is intended for use in the technical office environment. While TOP can be regarded as complementary to the MAP protocol which is intended for the automation of the factory floor, it actually started in competition with MAP in the office environment. While MAP is still in the experimental stage, TOP has largely matured and is already in use.

MAP is gaining wide support in America for use on the factory floor but as yet the standard is not stable and it is only being used in pilot installations. It is favoured in this environment because of its deterministic behaviour and because of the advantages that a broadband system offers, namely noise immunity and flexibility arising from the multi-channel support.

However as this study shows, the MAP protocol is very taxing on the hardware, currently offers a performance that is poorer than the Ethernet system and has a propagation delay that has a probabilistic element arising from load dependant delays.

The Ethernet system clearly has a better performance currently although there is definite upper limit on system performance determined by the physical properties of the network. The MAP

protocol is currently limited by the capabilities of the electronic hardware while neither the protocol nor the physical properties of the network pose a limitation on the maximum system performance. It is difficult to see much improvement arising from improved hardware performance for this LAN in the forseeable future.

The last topics to be discussed are one of reliability of the network and its suitability to the control environment. They will be discussed together as they are interrelated. Control networks are required to be dependable and have high reliability. This means that they should have redundancy, dynamic maintainance, fault containment and isolation and no single point of failure. Regarding survivability, the effects of noise on the systems need to be considered as well the recovery procedures used by the network when communication is interrupted as a result of severe noise disturbance or cable failure.

Considering the Ethernet system first, current hardware designs do not lend themselves well to redundancy as they do not cater for redundancy of the communication systems of the nodes, but there is no inherent deficiency in the standard that hinders the development of a redundant system.

The latter system has excellent dynamic maintainance performance as nodes can be connected and removed from the network while it is operational.

Any single bus system has poor fault isolation and containment properties and the Ethernet system is no exception. Two points in its favour are that all node connections are via passive transformers and many nodes available commercially have diagnostic tools for cable maintenance incorporated in them as standard equipment thus allowing rapid location of faults in the network.

Ethernet is a fully distributed architecture and thus has no single point of failure. It is a broadband system with node interfaces that have only a small section of analogue circuits for the collision detection. It is thus more susceptible to noise disturbance than a broadband network but is otherwise highly stable over time. Recovery from noise disturbance is rapid as the transmitting node will sense the disturbance as a collision or the receiving node will reject the packet because of a bad CRC check. All that is required of the network is to establish the fact that packets were corrupted and to retransmit them. All other transmissions would continue as normal. This is a simple and efficient method of recovery for the communication medium.

Turning now to the MAP token passing system, the potential for adding redundancy to the network is poor. One of the supposed advantages of the broadband network is that all the factory communications can be put on to one backbone cable thus obviating the need for a multiplicity of cabling. In the control environment this philosophy is specifically avoided as the loss

of the cable means the loss of the system. In addition as many signals are carried on the one cable, failure of the communication equipment on another channel could result in erroneous access of the control channel.

There is a large amount of analogue circuitry involved with the hardware of a broadband system which is both a strength and a weakness of the system. The tuned circuits defining the channels of the medium greatly reduce the effects of noise on the network but at the same time they are prone to drift decreasing the reliabilty of the network and demanding a significant maintenance load.

The headend forms a single point of failure for the MAP network, something which a distributed system normally tries to avoid. Because of its presence, fault isolation is not practicable as all communication on the bus must go via the headend. Thus any failure on the bus isolates the transmitters from the receivers, causing complete failure of the network.

Isolation of a fault by removal of a node causes all bus activity to cease while a lengthy contention procedure is executed to determine the successor of the previous node. This can take seconds to complete(33), hardly suitable for a real time system.

As mentioned above, the noise immunity of the broadband system is better than that of the baseband. However when noise interference does cause the loss of the token, all stations must go through a

contention process to find which will be the originator of the new token, also a process lasting several seconds. Thus a severe noise disturbance can cause communication failure lasting several seconds.

This study has yielded interesting results relating to the propagation delay characteristics of the networks. Looking at the results of the simulation of the Ethernet system without the processing delay, it can be seen that while the average delay is reasonable, the maximum delay is significantly longer indicating a large spread in delay times with poor predictability. The latter characteristic has been the main reason why this network has been rejected for use in real time applications.

When the processing delay is included in the simulation, the average delay increases by an amount related to that of the propagation delay but the maximum delay decreases by a large factor to a value less than twice that of the average delay. This greatly reduces the probabalistic aspect of the performance to an acceptable value.

The model of the MAP system with no processing delay shows a propagation delay which is almost independant of system load. This characteristic is what has made this protocol attractive to the real time user.

When the processing delay is included in the simulation the

picture changes significantly in that the performance drops off by a factor of approximately twenty. Because the storage time of the communication buffer is included in the simulation, the propagation delay becomes unacceptably long as saturation is reached.

When the protocol is modified to allow the token holder to transmit only one packet before passing on the token. the system throughput and average propagation delay are nearly the same as that of the Ethernet system. The maximum delay is slightly better than that of the Ethernet system. This system also shows an element of indeterminacy because of the unpredictable storage times in the communications buffers arising from the varying communication load. As shown in the results the difference is less than a factor of two which is not significant in systems such as these.

From the above it can be seen that there is not a significant difference between the two systems. Generally the MAP token passing protocol is preferred in the industrial environment because of its high performance and deterministic behaviour over the Ethernet system. This study shows that when cognisance is taken of the capabilities of realisable hardware then the picture changes significantly. The performance of the MAP system deteriorates dramatically to a figure that is significantly worse than that of the Ethernet system. After adjusting one of the parameters of the system it is possible to improve the performance to be almost equal to that of the Ethernet system.

The propagation delay of the latter system becomes far less indeterminate to the extent that it becomes comparable to that of the MAP network. Conversely the MAP system is less deterministic when system loading is taken into account. If node removal and insertion and the effects of noise are taken into account the advantages of the MAP network are even further diminished.

These figures make the Ethernet system a viable alternative to the MAP system because its performance is equal to or better than that of the MAP system and it is cheaper. The latter is true because Ethernet is a more mature system and it is less complex. In the long term the token passing protocol still has the potential advantage of speed but whether this is realisable is dependant on suitable hardware becoming available to meet the extremely severe demands made of the communication systems of the nodes.

REFERENCES

1. Peberdy, N.J. and Rodd, M.G., "Hydra - An experimental distributed computing network", SACAC Symposium on Distributed Computer Control Systems, Durban, April (1981).
2. Weehuizen, H.F. and Rodd, M.G., "Simulation of data transport systems of distributed computer networks", Symposium on Distributed Computer Control Systems, Durban, April (1981).
3. IEEE Standards for Local Area Networks: Std 802.3: "Carrier sense multiple access with collision detection", IEEE Standards Board, New York, (1985).
4. 3 Com, "Etherlink", 3 Com Corporation, Mountain View.
5. Intel, "OpenNET", Intel Corporation, Santa Clara.
6. Intel Corporation, "O.E.M systems handbook, iSXM 552 communications engine", Intel, Santa Clara, 1986.
7. Intel Corporation, "O.E.M. systems handbook, iRMX Net", Intel, Santa Clara, 1986.
8. Zimmermann, H., "OSI reference model - The ISO model for open systems interconnection", IEEE Trans. COM-28 no 4, 425-432 (1980).
9. Iverson, W.R., "Distributed system controls factory", Electronics, 56, no 21, 161-162 (1980).
10. Murphy, J.A., "Token passing protocol boosts throughput in local networks", Electronics 8 sept 1986.
11. Lineback, J.R. and Barney, C.,"Ti readies its chips for IBM token-ring LAN", Electronics, 58, no.28, 16-17 (1985).

12. Instrument Society of America, Technical Committee SP-72.1 Proway-LAN.

13. IEEE Standards for Local Area Networks: Std 802.4: "Token passing bus access method", IEEE Standards Board, New York, (1985).

14. General Motors, "MAP specification 2.1" March 1985.

15. Tobagi, F.A., "Multiaccess protocols in packet communication systems" IEEE trans. COM-28 468-488 (1980)

16. Shacham, N. and Hunt, V.B., "Performance evaluation of the CSMA/CD channel access protocol in cimmon channel networks", Proc of IFIP TC6 International In-depth Symposium on Local Computer Networks, Florence, April (1982).

17. Arthurs, E. and Stuck, B.W., "A theoretical performance analysis of polling and carrier sense collision detection communication systems", Ibid.

18. Vo-Dai, T., "Throughput - delay analysis of the non slotted and non persistent CSMA/CD protocol", Ibid.

19. Bux, W., Closs, F., Jansen, P.A., Kummerle, K., Muller, H.R. and Rothauser, E.H., "A local area communications network based on a reliable token ring system", Ibid.

20. Ansaldi, W., Olobardi, M. and Traverso, A.M., "Definition and development of a protocol for an industrial plant control network", Ibid.

21. Abramson, N., "The Aloha system - Another alternative for computer communication", AFIPS Fall Joint Computer Conf. 37 AFIPS Press (1970).

22. Nabielsky, J., "Interfacing to the 10 Mbps Ethernet:

Observations and Conclusions", Computer Communication Review" 14 no 2 (1984).

23. Intel Corporation, "OEM systems handbook, iSBC 550 Ethernet communications controller", Intel (1983).

24. Intel Corporation, "OEM systems handbook, iNA 960 networking software", Intel (1985).

25. Hewlett Packard. "LAN 1000 performance brief", part no 5953-5016, (1985).

26. Samari, N.K. and Schneider, G.M., "A queueing theory based analytical model of a distributed computer network", IEEE trans on computers c-29 no 11 (1980).

27. Kopetz, H., Lohnert, H., Merker, W. and Pauthner, G., "The Architecture of MARS", Techn. Univ. Berlin, Bericht No MA 82/2, (1982).

28. Tschammer, V., Klessmann, H. and Wawer, W., "A concept for an adaptive channel access control for real time LANs", Fourth Annual European Fibre Optic Communications and Local Area Networks Exposition, Amsterdam, June (1986).

29. Johansson, M., "MAP protocol: history and lower layers", RUGSA Symposium on Real Time Networking Computing and Control", Johannesburg, August (1986).

30. Intel Corporation, "Workshop on Local Area Networks", Pretoria, March (1987).

31. Weehuizen, H.F., Rodd, M.G. and McLaren, S.G., "The simulated performance of practical Ethernet and MAP networks", 7th IFAC Workshop on Distributed Computer Control Networks", (DCCS-86) Mayschoss, FRG. October (1986).

32. Mellish, M., of Gould Inc., U.S.A., personal communication, August (1986).

33. Damsker, D.J., "Critique to MAP and Proway", (DCCS-86) Ibid.

34. Atkins, M.S., "A comparison of SIMULA and GPSS for simulating sparse traffic", Simulation 93-100 (1980).

35. Gerla, M. and Kleinrock, L., "On the toplogical design of distributed computer networks", IEEE Trans. on Commun. COM-25, (1977).

36. Hogeweg, P., "Simulating the growth of cellular forms", Simulation, Sept (1978).

37. Intel Corporation, "The Ethernet", Version 1.0, Intel, Santa Clara (1980).

APPENDIX A

SELECTION AND EXPLANATION OF THE SIMULATION METHOD

LANGUAGE CHOICE

The system being simulated is digital in nature rather than analogue so the choice of a modelling technique must obviously be from among those that are discrete in nature. Of the discrete modelling techniques, the first choice is between an analytical approach such as queuing theory, and a discrete event simulation method.

Before this choice can be made, the character of the modelling to be done must be discussed. Some of the parameters of the distributed computer transport network that are to be modelled are throughput, transit delay and the effects of buffer size. These could equally well be modelled using either of these techniques though some workers(26) feel that the queuing theory approach has definite advantages for large networks. The advantage claimed is shorter computing time, resulting in a more cost effective solution. The same authors say that simulation does give a better insight into the behaviour of the network which is one of the reasons that a simulation method was chosen. This study was undertaken in order to gain just such an understanding and is thus the main justification for using the simulation approach. As can be seen from the findings in the main

part of the text, this was important in discovering the limitations of the token passing protocol.

Having decided on a simulation method the next choice lies between a fixed-time-step method or a next-event method. The fixed-time-step methods are characterised by models written in standard scientific languages like FORTRAN while the other is characterised by the use of the simulation languages such as SIMSCRIPT, GPSS and SIMULA. Because of the ready availablity of languages like FORTRAN and because it is generally agreed(34,35) that the fixed-time-step method is more efficient in the simulation of very active systems, the natural choice for a simulation like this would seem to be a fixed-time-step method. However, other workers have shown(34) that discrete-event models are clearer and easier to develop than fixed-time-step models. This is because typically, simulation languges are used which offer sophisticated data structures simplifying both the tasks of developing the model and describing its operation. Hogeweg(36)has shown that discrete-event models are clearer and easier to develop than fixed time step models. The task of simulating the transport system of a computer network is fairly complex so it was decided to use a simulation language for this study.

The three most common languages currently used are SIMSCRIPT, GPSS and SIMULA. The first of these was not available on either of the two computers used for the study so the choice lay between the second two languages. GPSS is the more popular of the two and is based on the FORTRAN language, while SIMULA is based on Algol

60 and is thus a structured language. GPSS provides all the tools necessary for most simulation tasks. It allows a model to be developed quickly and generates a comprehensive standardised report at the completion of its simulation run. Unfortunately the program once written is not very readable as it consists of a sequence of instructions, each with a string of numeric parameters following it. If results other than those that are produced by the standard report generator are required then it is not an easy task to link a FORTRAN routine to the GPSS program to provide the extra results. SIMULA on the other hand is a superset of Algol 60 so has all the facilities of a standard scientific language available. It has only limited reporting facilities which are available as standard procedures and these vary depending on which compiler is used. Superficially it can then be said that GPSS would be the language of choice where the development time is expected to be short and the standard report presents an adequate picture of the system being simulated. However, before a final choice can be made, it is necessary to have a closer look at the system to be simulated. The data transport system of a distributed network consists of the communication medium, the tramitter and receiver hardware, the communications processor and the link to the node or host processor. The communications processor has a number of tasks to perform. It controls the transmitter and receiver as well as supervises encoding and decoding of the data. It builds and disassembles data packets and generates and checks cyclic redundancy codes. It also handles the retransmission of packets

in the event of a communications failure. From this it can be seen that the simulation will consist of the interaction of a large number of entities which are themselves fairly complex. They will each have their own data structure which will need to be accessed by other entities in the model to provide the correct interaction in the process of simulating the operation of the network.

Unfortunately this kind of interaction is not easily handled in GPSS as individual entities cannot be referenced. This means that the parameters of one entity are not readily available to other entities. A solution to this difficulty is to hold a copy of the local variables as a sub-array in a global array of variables which is accessible to all entities. Entities can thus access parameters of other entities by accessing a copy of those parameters in the global array. This often leads to programming errors as the representation of the data in this way does not reflect the structure of the original data. This also results in slow execution as two lots of parameters have to be updated in the process of simulation. The handling of two data sets makes development difficult and the process is prone to errors. An alternative technique that can be used in GPSS is the 'UNLINK' operation. Using this, program development becomes easier as the data structures more closely resemble the system being simulated. Copies of the interacting entities are linked into chains, allowing the entities so linked to access the parameters of other entities in the chain. This technique also leads to an improvement in execution time but it is still difficult to

program correctly. Data has to be transferred between the copy and the original entity and buffering must be used to ensure correct updating of information.

SIMULA on the other hand has a powerful tool called the 'class' concept which allows the difficulties mentioned above with GPSS to be overcome. The major features of the 'class' in SIMULA are that many instances of it can exist at the same time, and that each instance possesses its own local variables. These can readily be accessed by other objects in the simulation. When each instance is activated, either remotely or locally, it processes its code. These features make it possible for the simulation model to closely resemble the actual system, making both the development and the understanding of the model much easier.

The results required from this simulation study are also a fair bit different from that presented by the standard GPSS report. These factors combined strongly favoured the use of SIMULA for this study. In addition, this work was started in Britain where Algol is the favoured scientific computer language. A fair amount of expertise was available in the use of SIMULA both at the campus where the work was started, and at the central computer site in London.

Thus SIMULA was chosen as the language to be used for the modelling of these networks.

A DESCRIPTION OF THE SIMULA LANGUAGE

As has been mentioned in the previous section, SIMULA is a simulation language that allows the modelling of systems using the discrete event technique. This technique allows the simulation of most physical systems that are dependant on time. For instance, if it is desired to optimise the sequence of operations in a machine shop performing a series of operations on a piece of material, the machine shop can be viewed from a simulation point of view as a series of operation completions. While an operation is being performed on the material, nothing of significance occurs, except to say that the machine doing the operation is busy. When the operation completes, the material is moved to the next machine and the first machine becomes available. These latter events are all significant from the simulation point of view.

This approach has two advantages. The first is that the computer performing the simulation is only active when a significant event occurs. The second is that the program describes a sequence of actions rather than a set of permanent relationships. Thus, continuing the above example, instead of the machine acting on passive pieces of material which must all be identical in nature from the permanent relationship point of view, each piece of material can have its own character. This is possible using SIMULA because the permanent relationships are subordinated allowing the modeling of a great variety of decision rules and interactions.

The language has been built on a few basic concepts common to a wide variety of problems thus giving the user a systematic approach to the simulation of systems. These tools have been integrated into the Algol language in such a manner as to facilitate readability. To facilitate the introduction of the concepts built into the language an example will be described which is familiar to most people. It is chosen to be sufficiently complex to force a methodical approach to its description and is a time organised system in which actions occur in parallel. The model is that of a production machine shop mentioned briefly above. Material enters the shop from a materials handling bay and leaves as the finished product via despatch. The jobs to be performed are determined from orders handled via reception. The outside is excluded from the system except that it determines the arrival pattern of orders and materials, and the nature of the operations to be performed. Inside the shop is a variety of machine tools, groups of which perform different functions. The orders determine the operations to be performed by the machine tools to produce the finished product. Thus the path of the material through the machine shop is determined for each order, both by the sequence of operations to be performed and by the load on each machine tool as determined by the total work load of the shop.

If an operation has to be performed on a piece of material by a machine that is standing idle, that job has to wait until an

operator is available. If the machine is busy then the job has to be queued until the machine is free. When the machine becomes available the required operation can be performed on the material. The time it takes to do the operation will depend upon the size and type of material and the size of the machine amongst other things. On completion of the operation the material is removed from the machine and transferred to the next machine, where operations appropriate to this machine are performed. On completion of all operations the completed product is handed to despatch. When a machine has finished its operation on a particular piece of material it takes the next piece of material waiting at the front of its queue and performs the required operation. When its queue is empty the machine becomes idle.

The model of the machine shop described above will help the user to study the overall behaviour of the machine shop, its throughput, the sizes of the queues for the various machines and their workloads.

The model described above will now be used to introduce the concepts of SIMULA and its constructs. The shop can be divided up in to the groups of machine tools mentioned above, and reception which will represent the interaction from outside. The first subsystem to be discussed is that of the machine tools. This can be divided into a queue of jobs waiting to be done and the machine tool itself. This is characterised by the sequence of operations it has to perform, and a description of each operation that is to be performed. Thus the machine in the simulation model

consists of a list of operations(a data structure) and a description of the operations(an action pattern). These two form an object of the language which in SIMULA is called a 'Class'. The structure of this is identical to that of the procedure in Algol or any other structured language. The class used to describe the character of one of the machine tools can be used to describe the character of all machine tools having similar characteristics as many copies can be made of a class. The action pattern for each class would be identical for all copies but each copy would have its own data structure. This allows the character of each to vary, simulating either local differences or modification by external influences. Obviously previous history could also be stored, providing memory effects.

The queue subsystem provides SIMULA with its list processing capability. This can be broken down into several elements, the first of which is the header called HEAD. This defines the queue and can be referenced by other objects in the program. It contains pointers to the first and last elements in the queue. The other elements are the queue objects. They are objects of the type described above and in the present example would be orders placed in the queue of a machine tool waiting to be processed. These have the typical features of queues, namely the concepts of 'first' and 'last' relative to the queue, and 'next' relative to its members.

The operations that can be performed on queues include that of

insertion and transfer from one queue to another, and deletion. These can be used in conjunction with the relational operators mentioned above to provide a flexible and useful set of tools for queue control.

The first object mentioned, reception, which represents the outside world provides the stimulus for the model. Here the standard random drawing procedures are provided. Despatch, which receives the completed work is not represented by any special object. Data collection procedures for performance evaluation must be written by the user. In the compiler used for this application, a histogram package in the form of a standard procedure is provided to aid in the process of data gathering.

Now that the main concepts of the language have been introduced some of the remaining features will be discussed. Some of these increase the power of the language while others are important for the successful execution of a model. The first of the latter type is one of scheduling. This allows the sequencing of events with respect to time. The process of simulation is started by activating a class object. This executes its action sequence and in the process activates another class object or schedules itself to be activated at a later date.

The process of activating a task is handled by a special object called the sequencing supervisor. Execution of the activation command causes an event notice to be placed in an event list which is an ordered set of event notices. The sequencing

supervisor moniters this list and activates the object that is at its head. This will be the next significant event in the simulation. Thus time in the simulation advances in irregular steps and corresponds to the intervals between significant events.

The activation commands take on various forms. The first of these allows the immediate activation of another object after the present one has completed. This form allows simulation of the simultaneous operation of more than one process in that there is no time increment between the completion of the operation of the current object and the activaton of another object. One object does however complete before the next starts so the operation is still sequential at any time instant.

If two events are scheduled for the same time then they are executed in the same order that they are placed in the event list. This order can be modified by the use of extra controls in the activation command.

Extensions to this command allow the scheduling of objects at a specific time or at a certain time after the current. They may also be scheduled relative to the activation time of other objects. The facility also exists to change the scheduled time for an object by using the 'reactivate' command. This is identical in all other respects to to the 'ACTIVATE' command.

The corollary of the activation command is the 'PASSIVATE'

command. This stops the operation of an object and leaves it dormant and in such a state that it can be activated again at a later time. Terminating an object removes it from the simulation permanently, provided that there are no references of any sort to that object.

The 'HOLD' statement allows an object to be suspended and rescheduled at a time equal to the current time plus the delay time. This is very useful for simulating the time it takes for a certain operation to be performed.

The last feature to be described briefly is the one that makes SIMULA preferable to GPSS in this study. It is the property called connection and has two forms. The first, called the genitive notation, allows one object to access a single local variable of another object. The form of the mechanism is very simple. If an object requires access to a variable named VAR in an object named MACHINE, then it accesses that variable as MACHINE.VAR. The second form of the connection facility is the 'INSPECT' statement. Through this an object can access all of the variables of another object. This has the added facility of allowing conditional clauses and has a construct that is very similar to the CASE statement of Algol or PASCAL.

Using the above loose description of the SIMULA language, the machine shop example above will be described in a mixture of the formal constructs of Algol and SIMULA and informal English language statements. The formal statements will be shown in upper

case while those that are informal will be shown in lower case with comments shown thus <comment>. The operation of the simulation model will then be described.

The SIMULA block in which all the above statements are valid appears as the main Algol block. Thus the basic block structure of the SIMULA model is as follows:

```
SIMULATION CLASS machine_shop;
  BEGIN
  declarations;
  description of simulation model;
  END OF SIMULATION BLOCK;
```

The construct "SIMULATION CLASS" invokes the full simulation facilities of the language. If this is not used, sub-sets of the facilities are available but these are not relevent to the task at hand, so will not be discussed further.

The three main components in the simulation model are the orders, the machines and reception. The first of these is described as follows:

```
PROCESS CLASS order_description(number); INTEGER number;
  BEGIN REAL ARRAY operation(1:total_machines);
  REAL arrival_time;
  END;
```

This has the simplest description as it contains only a data

structure(the order) and no action pattern. In this case the data structure consists of two variables. The first of these is the operation time which is an array, the elements of which will be used to control the processing by each machine that performs an operation on it. The second will be used for logging statistical data that will be used for data collection at the end of the simulation. The parameter included in the class declaration will be used to identify the instance of the order while it is active in the simulation.

The second object in the simulation is the machine. This is somewhat more complex as it has both a data structure and an action pattern. Its class description is given below:

```
PROCESS CLASS machine_description(type); INTEGER type;
  BEGIN
  WHILE TRUE DO
    BEGIN
    WHILE CARDINAL(que(type) > Ø DO
      INSPECT que(type).first DO
        BEGIN HOLD(operation_time(type));
        IF type <= total_machines THEN
          BEGIN
          order(number).INTO(que(type+1));
          IF CARDINAL(que(type+1)) = 1 THEN
          ACTIVATE machine(type+1);
          END;
        ELSE
          BEGIN
```

```
            log statistics;
            order(number).OUT;
            order(number) = NONE;
            END;
        END of while;
    PASSIVATE;
    END of true loop;
  END of class machine_description;
```

Here the class object is declared by the PROCESS CLASS statement. It follows the structure of the Algol procedure, thus it is fully defined in the BEGIN..END(of class machine) pair.

The "WHILE TRUE DO" construct forms an infinite loop within which the actions of the class object are executed. Thus each class object is an entity within itself. Activity within the class is started from another object in the simulation model and is stopped by the PASSIVATE statement. Reactivation of the class instance causes execution to commence at the instruction immediately following the PASSIVATE statement.

The statement "WHILE CARDINAL(que(type)) > Ø DO" requires a fair amount of explanation. The variable "que(type)" is a reference to a HEAD object. In this case, the input queue for machine(type). CARDINAL is a procedure that returns the length of the queue as an integer. The WHILE construct allows the subsequent statements to be executed as long as the queue is not empty. When it does become empty execution proceeds to the next statement which is

the PASSIVATE statement. Action thus ceases until an object is placed in the queue. The object that does the placing in the queue must check the status of the queue and if it was empty it must activate the current "machine" object.

The statement "INSPECT que(type).first DO" uses the two SIMULA connection constructs. The "que(type).first" construct is a reference to the predefined variable of the queue (HEAD object) which points to the first object in the queue. The INSPECT..DO construct allows the current object to access the variables of the queue object. Thus the complete statement firstly provides access to the HEAD object and then further provides access to an object in the linked list of that object.

This connection statement is effective in the block immediately following the inspect statement and is defined by the "BEGIN..END of while;" pair immediately following the statement.The first statement in this block, DELAY(operation_ time(type)) accesses the variable "operation time(type)" which is local to the first object in the queue defined by "que". This is the "order" object which is being used to define the operation time for the current machine. This illustrates the ease with which one object can access variables which are local to another.

The reserved word DELAY is one of the scheduling terms in SIMULA. It causes the current process to cease executing (the same effect as PASSIVATE) and to be re-scheduled at a time equal to the current time plus the delay time which in this case is the

operation time.

The next statement of interest is "order(number).INTO(que(type+1))". This appears within the connection(INSPECT) block and as the variable "number" is not defined within the current CLASS, the object referenced in the INSPECT statement is scanned for the variable name. As this is the reference name of the first object in the queue, that object is acted upon by the second part of the statement under discussion. The expression "INTO(que(type+1)) causes the object to be removed from any queue that it may be in and then be placed in the designated queue (in this case "que(type+1)"). A point to note here is that an object may be a member of only one set at a time. Some versions of SIMULA have a means of overcoming this limitation. This will be discussed later.

If this object is placed in a queue whose related machine object is dormant(passivated), a mechanism must exist which will activate that machine object. This situation occurs in the present model when the machine object finds its queue empty and passivates itself. The statement "IF CARDINAL(que(type+1)) = 1 THEN" is the conditional execution statement which checks the length of the queue. If it is found to be equal to one then the corresponding machine object is activated in the second half of this statement. The reason for it checking for a length of one rather than zero is because the length is checked after the order object has been inserted into the queue.

The ELSE part of the IF..THEN..ELSE construct is executed in this object when the third (and last in this simulation model) machine has performed its operation for a particular order.

The informal words "log statistics" indicate that any data that should be recorded for report generation relating to the order just handled should be done so at this point. The two instructions that follow remove this object from the simulation. The first of these, "order(number).OUT" removes it from its queue. The second, "order(number) = NONE" de-references the order_description object. As will be seen in the full program listing that follows, each object is referred to by a reference variable rather than the actual class name. As long as an instance of a class can be referenced, that object is retained in the simulation. As soon as all references to a particular object are removed, it is removed from the simulation. Thus in the two statements just discussed, the first removes the object from any queue that it may be in while the second removes the link between the object and its reference variable.

The third major component of the simulation is the machine shop reception which will generate the orders for the simulation. In this example it will simply generate orders at pseudo-random intervals. It is written as follows:

```
PROCESS CLASS generator_description;
  BEGIN INTEGER I,J;
```

```
FOR I = 1 TO total_orders DO
  BEGIN order(I) = NEW order_description(I);
  FOR J = 1 TO total_orders do
    order(I).operation_time(J) = UNIFORM(min,max,u_seed);
  order(I).INTO(que(1));
  IF CARDINAL(que(1)) = 1 THEN
    ACTIVATE machine(1);
  HOLD(NEGEXP(intvl,seed));
  END;
END of generator;
```

It has the class structure as described above. The first line to note is the FOR..DO construct. This provides the counter for the number of orders to be generated for the shop in the block that follows. The next line contains the statement "order(I) = NEW order_description((I)". This causes a new instance of the class "order_description" to be generated that is identified as instance "I" and it is assigned to the reference variable "order(I)". At this point the instance exists but it has no data, has not executed any actions and is unrelated to any other object in the simulation. The first statement after this uses the genitive connection to access one of the array variables in the order object and assigns values to it. The "UNIFORM" procedure draws a number from the uniform distribution in the range (A,B). The third parameter of the procedure is a control variable which is used to select the next number in the sequence when the procedure is next called. This drawing procedure and the next discussed below form the inputs to the model.

The next statement places it into a queue where it can begin to take part in the simulation. Again the length of this queue must be checked to see if the corresponding machine object must be activated. The next point of interest is the argument of the "HOLD" instruction. This is one of several random drawing instructions. It draws a number from a negative exponential distribution with a mean of 1/intvl. The "seed" is a control value which is used the next time that that particular procedure is called. If the same procedure is used more than once in the simulation model then the initial value of the seed in each call to the procedure can be given a different value. This provides a different starting point in the pseudo-random sequence for the same drawing procedure.

A complete listing of the the simulation model for the machine shop will now be given.

```
SIMULATION CLASS machine_shop;
   BEGIN
   PROCESS CLASS order_description(number); INTEGER number;
      BEGIN REAL ARRAY operation(1:total_machines);
      REAL arrival_time;
      END of order_description;

   PROCESS CLASS machine_description(type); INTEGER type;
      BEGIN
      WHILE TRUE DO
        BEGIN
```

```
        WHILE CARDINAL(que(type)) > 0 DO
          INSPECT que(type).first DO
            BEGIN HOLD(operation_time(type));
            IF type <= total_machines THEN
              BEGIN
              order(number).into(que(type+1));
              IF CARDINAL(que(type+1)) = 1 THEN
                ACTIVATE machine(type+1);
              END;
            ELSE
              BEGIN
              log statistics;
              order(number).out;
              order(number) = NONE;
              END;
          END of while not empty;
        PASSIVATE;
      END of true loop;
    END of machine_description;

  PROCESS CLASS generator_description;
    BEGIN INTEGER I,J;
    FOR I = 1 TO total_orders DO
      BEGIN order(I) = NEW order_description(I);
      FOR J = 1 to total_machines DO
        order(I).operation_time(J) = UNIFORM(min,max,u_seed);
      order(I).INTO(que(1));
      IF CARDINAL(que(1)) = 1 THEN
```

```
        ACTIVATE machine(1);
      HOLD(NEGEXP(intvl,seed));
      END;
    END of generator_description;

COMMENT start of main program;
  BEGIN
  INTEGER K,seed,total_machines,total_orders;
  REAL intvl,simulation_period;
  REF(HEAD) ARRAY que(1:total_machines);
  REF(order_description) ARRAY order(1:total_orders);
  REF(machine_description) ARRAY machine(1:total_machines);

  total_machines = 3;
  total_orders = 500;
  simulation_period = 1000;
  FOR K = 1 TO total_machines DO
    machine(K) = ACTIVATE NEW machine_description(K);

  ACTIVATE NEW order_generator;
  HOLD(simulation_period);
  report results from statistics logged;
  END of main program;
END of simulation;
```

Except for report generation, this is a working program. The section that requires explanation now is the main program.

The first few lines are standard Algol declarations. The first group of lines of note are the three beginning with "REF(HEAD) ARRAY que...". This is a SIMULA declaration of a reference variable that will be assigned to a class instance in the main body of the program.

The two lines that follow this group assign values to some of the constants in the program. They could be entered as values from a data file but are entered here as constants to keep the example simple.

The FOR loop creates and activates the main objects of the simulation. Each is in turn first created and then activated. They initialise themselves and then become dormant when they find that their queues are empty. Once this process is complete, action proceeds to the next line in the main program. This is the line that creates and activates the order generator class object. This operates as described previously, creating orders and activating the machine objects if their queues were empty.

The next line causes the main program to stop its actions while the main simulation proceeds. The hold time here can be any time great enough to ensure that all other simulation activity has completed. This imposes no penalty on the actual computing time because, this being a next event simulation, time will advance in one increment from the last simulation event time to the end of the simulation period.

Once the simulation is complete and the main program has re-started, the data will be collected from the simultion run and printed out by the routines that report the results of the run.

THE ACTUAL LANGUAGE USED

The program as described above is written in what is popularly called SIMULA. Its exact title is SIMULA 67 and is the version that is used and implied in most references. This is the language that was used while the author was in England for the initial studies. The language that has been used for most of the work is an earlier version called simply, SIMULA. As far as can be ascertained the origin of the language is the same as that of SIMULA 67. It is a slightly simpler version of the latter, having fewer of the constructs described above, but having no less capability. The price for this is that the capability is achieved with less convenient constructs. A speed penalty could also exist for the execution time of the simulation but no information is available on this. Also, some of the reserved words defining the language are different.

As all the programs described in this thesis are written in SIMULA rather than SIMULA 67 it is necessary to describe the differences and to show how the program described above would be written in SIMULA. For the remainder of this appendix and all the other appendices, the reference to SIMULA will imply the early version to be described. If the more common version. SIMULA 67 is

referred to, it will be done so explicitly.

The first major difference is that SIMULA has only the SIMULA 67 simulation class available. This is the version of the language that has been described above and is the full language set that SIMULA 67 provides. The other versions are subsets of this. As it is necessary to use the simulation class for this study this difference is of no consequence here.

The only other difference is that the genitive connection is not implemented in SIMULA. This means that the "INSPECT" statement has to be used in its place which leads to a less convenient style of programming. In some of the constructs such as those used for queue processing where there are the words "first" and "last", it is not possible to use the "inspect" statement. These are available in SIMULA through the use of standard procedures. The queue that is to be referenced is transferred to the procedure as a parameter.

Table A.1 below lists the keywords of SIMULA 67 and their equivalents in SIMULA.

Table A.1
Keywords of SIMULA 67 and SIMULA

CLASS	ACTIVITY
REF(type)	ELEMENT (class object)
	SET (queue)

The second entry in the table requires some explanation. The reference variable (REF) of SIMULA 67 is universal in that it is

the only construct of this type. It defines a reference variable of any type where the specific type is defined in parentheses after the descriptor. This provides very strong typing as that variable is then only allowed to reference that particular type of object.

There are two declaractions in SIMULA on the other hand that only restrict the typing to class objects or sets(queues). These are shown in the table above.

The sample program written above in SIMULA 67 will now be given below in SIMULA.

```
SIMULA BEGIN
   INTEGER seed,total_machines,total_orders;
   REAL intvl;
   SET ARRAY que(1:total_machines);
   ELEMENT ARRAY order(1:total_orders), machine(1:machines);

   ACTIVITY order_description(number); INTEGER number;
     BEGIN
     REAL ARRAY operation(1:total_machines);
     REAL arrival_time;
     END of order_description;

   ACTIVITY machine_description(type); INTEGER type;
     BEGIN
     WHILE TRUE DO
```

```
        BEGIN
        WHILE CARDINAL (que(type)) > 0 DO
          INSPECT FIRST(que(type)) DO
            BEGIN
            HOLD(operation_time(type));
            IF type <= total_machines THEN
              BEGIN
              TRANSFER(order(number),que(type+1));
              IF CARDINAL(que(type+1)) = 1 THEN
                ACTIVATE machine(type+1);
              END;
            ELSE
              BEGIN
              log statistics;
              REMOVE(order(number));
              order(number) = NONE;
              END;
           END of while not empty;
        PASSIVATE;
        END of true loop;
      END of machine_description;

    ACTIVITY generator_description;
      BEGIN
      INTEGER i,j;
      FOR i = 1 TO total_orders DO
        BEGIN
```

```
      order(i) = NEW order_description(i);
      FOR j = 1 TO total_machines DO
      operation_time(j) = UNIFORM((min,max,u_seed);
      IF cardinal(que(1)) = 1 THEN
      ACTIVATE machine(1);
      HOLD(NEGEXP(intvl,seed));
      END;
    END of generator_description;


  min = 0;
  max = 0.5;
  seed = 4;
  u_seed = 6;
  total_machines = 3;
  total_orders = 500;
  simulation_period = 1000;
  FOR i = 1 TO total_machines DO
     BEGIN
     machine(i) = NEW machine_description(i);
     ACTIVATE machine(i);
     END;
  ACTIVATE NEW generator_description;
  HOLD(simulation_period);
  report results from logged statistics;
  END of simulation;
```

Points of difference to note are firstly, in the machine

description, the reference to the first element in the que. The lack of the genitive connection is overcome in this instance by the use of the standard procedure called "FIRST(S)" which returns the pointer to that object.

The "INTO" construct in this same object is replaced by the procedure "TRANSFER(X,S)", where X is the object to transferred and S is the queue to which it is to be transferred. In the generator description object that follows, the function of the "INTO" construct is performed by the "INCLUDE(X,S)" procedure where parameters have the same meaning as above.

The "REMOVE" procedure of SIMULA performs the equivalent function of the "X.OUT" construct of SIMULA 67. These differences are listed in table A.2 below.

TABLE A.2

SIMULA 67	SIMULA
S.FIRST	FIRST(S)
X.INTO(S)	TRANSFER(X,S)
	INCLUDE(X,S)
X.OUT	REMOVE(X)

where X is an object reference and
S is a set(queue) reference.

There are a few other minor differences and limitations such as the location of the global variable declarations. In the main program the creation and activation of the machine instances is done in one line in SIMULA 67 while it must be done in two lines in the earlier language.

This appendix has introduced the basic concepts of the SIMULA language and explained its method of operation. It will serve as a basis for explaining the development of the models of the various networks simulated.

APPENDIX B

FULL DEFINITION, AND SIMULATION MODEL OF THE ETHERNET LAN.

The Ethernet network is qualitatively described in chapter 2. This appendix defines all the parameters used in the simulation and develops the simulation model of the network.

ETHERNET PARAMETERS

This section defines those Ethernet parameters that relate to the simulation of the network.

The first of these parameters is not used directly in the model but is used to define several other parameters and is thus defined first. The presentation of these parameters may appear to be irregular, but this sequence allows the definitions to be built on what has previously been defined.

Propagation Delay.

In the Ethernet specification(37) this is defined in table 7-1 and section 7.1.2. It consists of the total of the worst case delays in cable segments, repeaters and transceiver circuitry and allows for the round trip delay in the network. The time is 44.99 usec. In terms of bit times this is equal to 450. Converting this back to the time domain, this gives an effective propagation delay of 45 μsec.

Jam Time.

A Jam is also known as enforcement. This occurs when the transceiver has detected a collision after the transmitter has started to send data. In order to ensure that the other station also detects the collision the former continues to transmit for at least a further 32 bit times, but not more than 48 bit times. Thus the maximum jam time is defined to be 4,8 µsec.

Slot Time.

Also referred to as the collision window, this is the minimum time in which a station can acquire the network i.e. the minimum time in which a transmitting node can be certain that no collision is going to occur. It is also an upper limit on the length of the jam signal generated on the detection of a collision. It is equal to the propagation delay time plus the maximum jam time plus a safety factor.

Thus, slot time = 45 + 4,8 + 1,4 = 51,2 µsec (512 bit times).

Logically, this is also used as the quantum in the scheduling for the re-transmission during backoff.

Deference time.

This is the time that any station must wait after a transmission ceases, before it can transmit a packet. It is intended to provide an interframe recovery time for both the station controllers and the physical medium. It has a defined minimum time of 9,6 µsec (preferred), but any time up to a maximum of

1Ø,6 µsec is permitted.

Binary exponential backoff.

This is the technique that is used to resolve the contention that will arise on the medium when more than one station attempts to transmit within a collision window. Each station delays its retransmission attempt by a randomly selected, integral number of slot times. Thus if no other station starts transmitting, the station that selects the shortest retransmission interval will be able to seize the medium. The other stations involved in the original contention will only attempt to retransmit after the new collision window has passed. Of course if two stations delay by the same amount then another collision will occur, but the probability is small.

The period of this retransmission delay is the slot time multiplied by the uniformly distributed random integer r in the range Ø <= r < 2^k where k = min(n,1Ø), n being the count of retransmission attempts. The original transmission is defined as attempt number 1. The number of retries is limited to 16 after which any further attempts are aborted.

Packet format

The format of the Ethernet packet is shown in figure 2.1. It consists of five fields excluding the preamble. The first two are address fields and can be one of two types. The first is a

Physical address. This uniquely identifies a particular station on the network. It is different from all other addresses

on the network.

Multicast address. This is a multi-destination address associated with a group of logically related stations. Included in this definition is the broadcast address which addresses all nodes on the network.

The first bit of these fields determines whether the address is physical(Ø) or multicast(1). The address field is 48 bits long, allowing for the possibility that each station has a unique address worldwide. If this were implemented, address conflict would not occur on any international link between local area networks of this type.

The first field of the packet is the Destination Address Field. This can be either a physical or multicast address and defines the node or nodes for which the packet is intended to be transmitted.

The second field is the Source Address Field. This identifies the node that is sending the frame. Logically it only makes sense if this address field is of the physical type.

The third field, as defined in the IEEE 802.3 standard indicates the length in bytes, of the data field of the packet. In the Ethernet standard this is defined as the type field and is used by the Data Link Layer to interpret the protocol used for the higher layers in a multi-protocol environment.

The next or fourth field is the data field. This can vary in length from 46 to 1500 bytes. The reason for a minimum size

restriction will be discussed later. The format of the data transmitted is arbitrary and is defined in the higher layers of the protocol used.

The last field of the data packet is the Frame Check Sequence. It is determined from the contents of the source, destination, length and data fields treated as one sequence. The encoding is defined by the generating polynomial

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

The CRC is determined mathematically for a given frame by the following procedure:

a. The first 32 bits of the frame are complemented.

b. The n bits of the packet data sequence are then used as coefficients of a polynomial M(x) of degree n-1, where the first data bit of the data sequence is the x^{n-1} and the last bit is the x^{0} term.

c. M(x) is multiplied by x^{32} and divided by G(x) which produces a remainder R(x) of degree less than or equal to 32.

d. The coefficients of this remainder are taken as a 32 bit sequence. This is complemented and the result is the CRC which is used as the frame check sequence.

A possible implementation of this is given in reference 37 and is shown below.

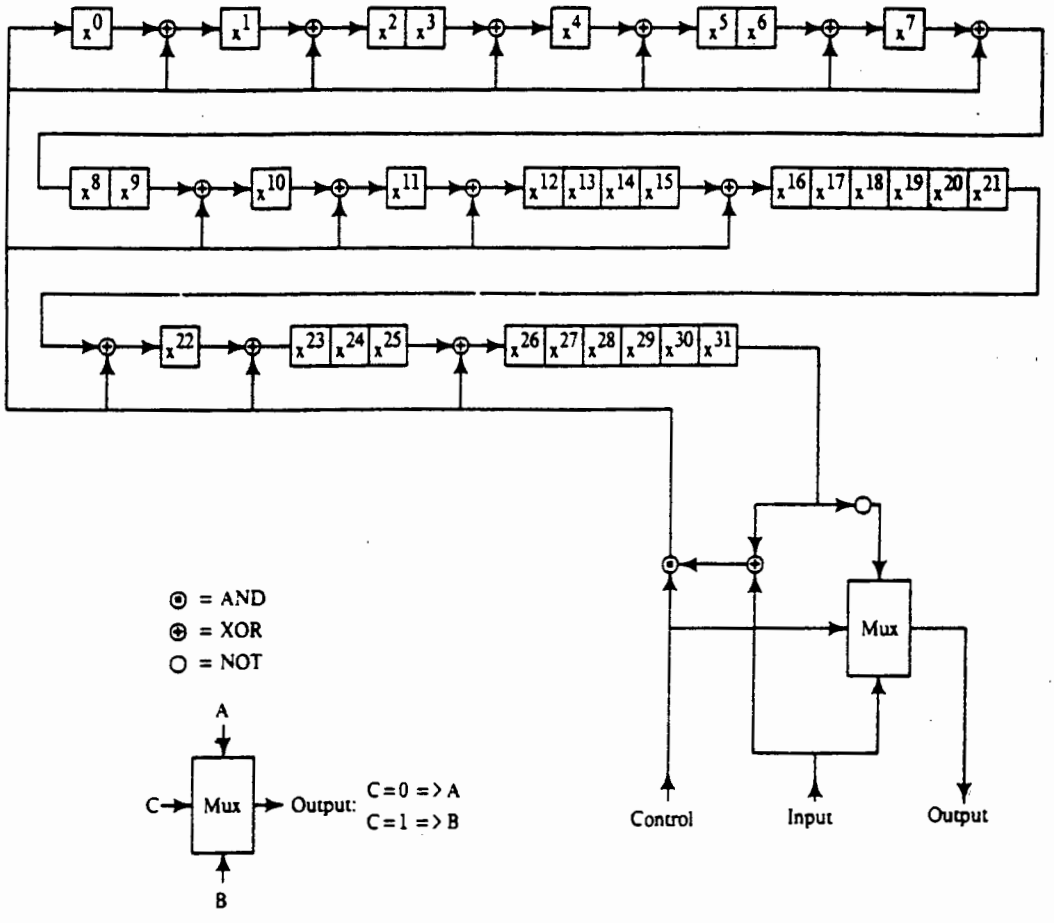

Fig B.1. CRC Generator/checker

The feedback shift register does the function of division of the data sequence by the generating polynomial G(x). The circuit has an input, an output and a control input. Initialisation and result detection circuits are not shown. When the control signal is high, the input data is fed into the shift register and sent to the output via the multiplexer. When the the control line is low the feedback path is disabled and the complement of the shift register data is shifted to the output.

Before data is transmitted the shift register is preset to an all one's condition. The control input is then set to one and the

complete data sequence is clocked through the shift register. The output line provides the signal that is transmitted over the network. When the data sequence is complete, the control is driven low and the complement of the contents of the shift register are clocked onto the network providing the CRC for the packet.

Checking the CRC on reception starts in a similar manner to the above where the shift register is preset to all one's and the control input is held high. The incoming data is then clocked through the shift register and the output signal goes to the receive logic.

At the end of the data sequence defined by the length field the control signal could be driven low and the incoming CRC could be checked with the contents of the shift register. This however would require a separate shift register to hold the incoming value and steering circuitry to control it. A simpler solution is to continue to clock the data through the shift register and at the end of the packet sequence to examine the value in the shift register. This value is constant for a correct CRC value and independant of the data sequence.

Packet transmission times

The header of the Ethernet packet consists of the fields shown below with their corresponding sizes in bytes:

Preamble	8
Destination address	6

Source address	6
Length	2
CRC	4
Total	26 bytes

Thus the total length of a 46 byte packet is 46 + 26 = 72 bytes

Therefore the transmission time is:

72 × 8 × 0,1 = 57,6 microseconds

Similarly the total length of the 1500 byte packet is 1500 + 26 = 1526 bytes.

Therefore the transmission time is:

1526 × 8 × 0,1 = 1220,8 microseconds.

THE CHARACTERISTIC EQUATION OF THE HEWLETT PACKARD NODE

The performance of this node is described in reference 25. The performance curve for their node in the transmitting mode is reproduced here in figure B.2. Performance figures derived from this are shown in table B.1.

TABLE B.1

Performance of Hewlett Packard Node (Transmitting)

	Packet Size (bytes)				
	50	100	500	1000	1500
Throughput (kbyte/sec)	13,182	27,727	117,727	190,454	243,182
Proc delay (msec)	4,44	3,51	3,80	4,43	4,95

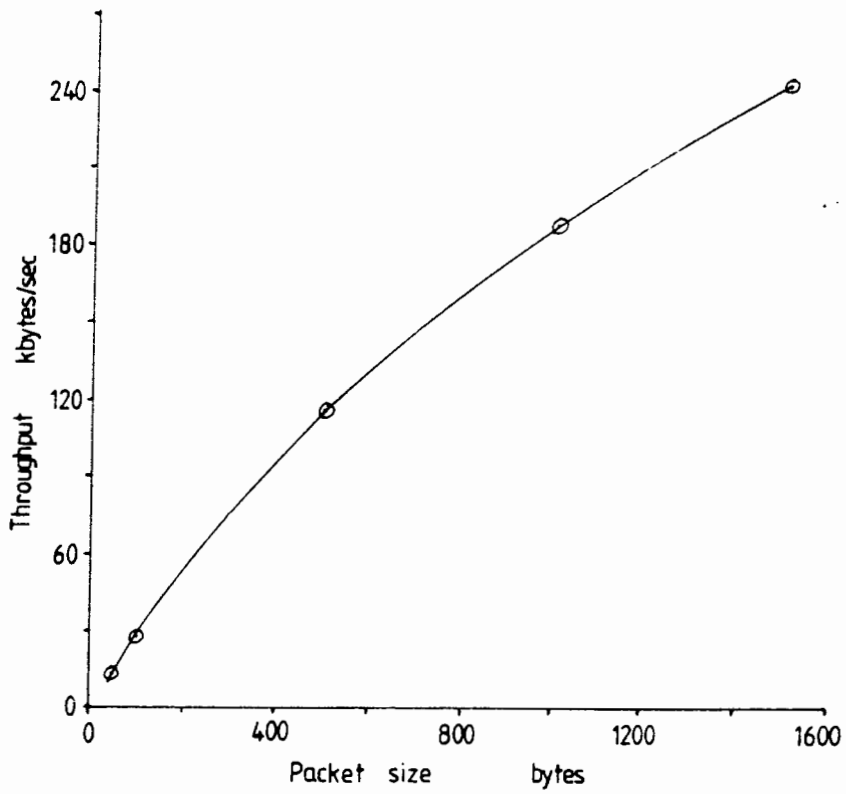

Figure B.2 Graph of performance of Hewlwtt Packard node in transmitting mode.

These results are shown graphically in figure B.3 and from this the characteristic equation is derived:

processing delay = 3,39 + 0,00105x msec

where x is no of data bytes.

THE SIMULATION

The SIMULA language that is used for this simulation has been discussed in Appendix A. The concepts and major new constructs that are available in this language have been introduced and illustrated by means of a simple example.

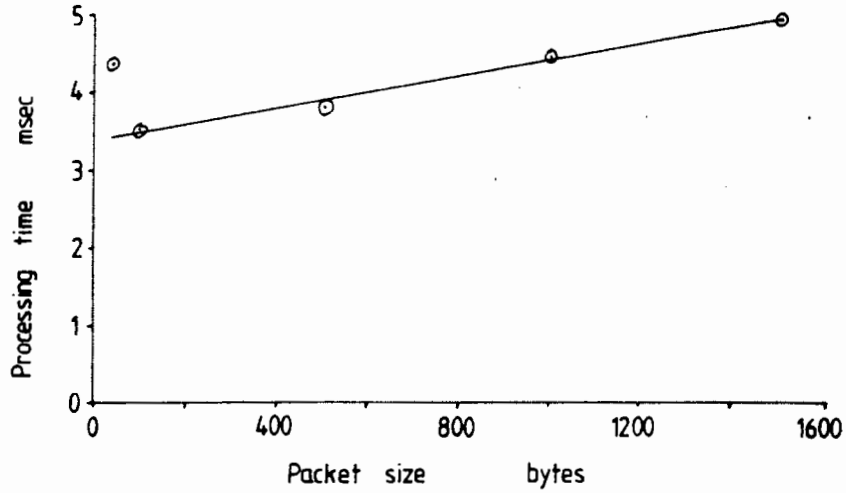

Figure B.3 Graph of processing delay with respect to packet size for the Hewlwtt Packard node.

This appendix will use the material presented there to describe the compositiion and operation of the Ethernet model developed for this study. The concepts and parameters introduced in the first part of this appendix are used in the development of this model.

The major entity of the simulation is that modelling the operation of the node. This will be the first to be discussed together with the packet entity as there is strong interaction between them. The other entities will be discussed as appropriate.

The node and packet entities.

The node entity is very easily divided into two parts. These are the transmitting and receiving sections. Logically they can be regarded as two separate entities and are treated as such in this simulation. However there is one very important aspect where they

cannot be regarded as independant and this relates to the communication load or throughput. The maximum throughput for a node(which is a constant for a given packet size) is the sum of the transmitting and receiving throughputs. Thus if the transmitting load is large, then the receiving load must be small and vice versa. In the model this is simulated by placing all the time(throughput) constraints in the transmitting section and allowing the receiving section to be perfect in that it has no time delays. This is justifiable on the grounds that there must always be a transmit/receive pair active, thus the throughput is determined by the transmitting node. This does have the limitation that if two or more nodes transmit to the same receiving node, the latter will be handling two or more times its nominal maximum load. This simulation is intended to deal only with one to one communication so the above would not be a problem. However, when the simulated packet load is generated, the source and destination addresses are randomly determined. Thus the situation of two nodes sending to the same receiving node could easily arise.

The obvious solution would be to modify the model to avoid this problem. This could be done by changing the method of allocating source and destination addresses so that they are always allocated in a manner that does not allow conflict.

However, when the above situation is considered, the anomaly does not introduce an error beyond the limitation described above. The character of each node is still correctly determined by the

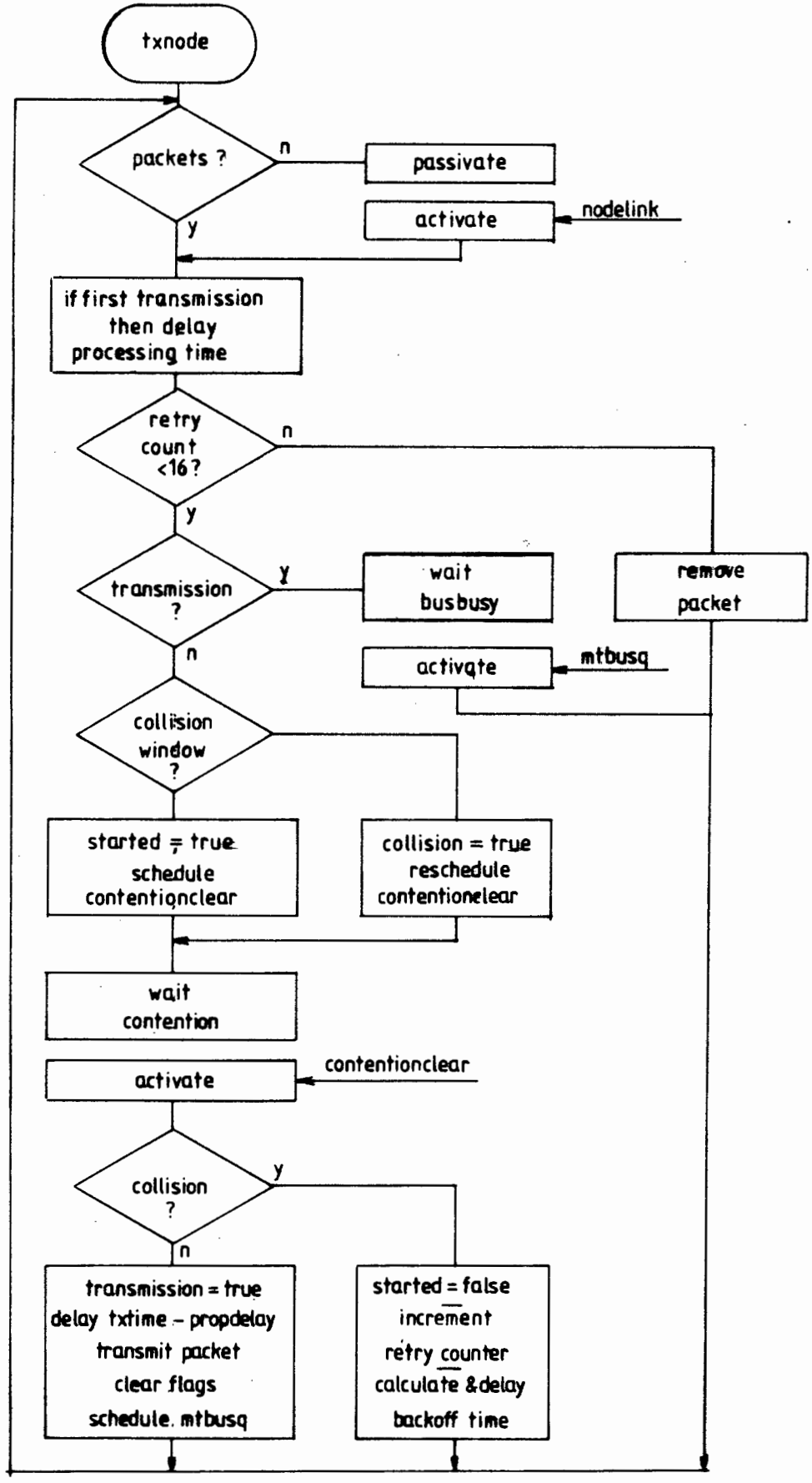

Figure B.4 Flow chart of the Node activity.

transmitter entity and the receiver section handles the data as if two or more separate nodes were receiving the data. This behaviour of the receiving node is possible because it exhibits no delays in the processing of its data. In fact all the receiving entities of the nodes could be modelled by a single instance of this entity. However logging of the results is done more easily when the individual receiver sections are present.

A flow chart of the transmitter section of the node activity is given in figure B.4 while the program listing follows below. The flow chart follows normal rules with one addition that allows the inclusion of the activate/passivate constructs. These use the normal action box but the passivate box has no exit while that of the activate instruction has no entry. On the side of the activate box is an arrow with the name of the activity that can activate this activity at that point.

```
ACTIVITY txnoad(n); INTEGER n;
  BEGIN REAL bkoftime;
  INTEGER rcountl;
  FOR dum=1 WHILE TRUE DO
    BEGIN
    FOR dum=1 WHILE NOT EMPTY(outbuf(n) DO
      BEGIN
      INSPECT FIRST(outbuf((n) WHEN packet DO
        BEGIN
        rcountl = rcount;
        IF NOT retry THEN HOLD(stationdelay);
```

```
    retry = TRUE;
    END;
  IF rcountl LEQ 15 THEN
    BEGIN
    IF transmission THEN
      BEGIN
      WAIT(busbusy);
      REMOVE(CURRENT);
      END;
    ELSE
      BEGIN
      IF NOT started THEN
        BEGIN
        started = TRUE;
        ACTIVATE contentionclr DELAY propdl--;
        END;
      ELSE
        BEGIN
        collision = TRUE;
        REACTIVATE contentionclr DELAY enforcement;
        END;
      WAIT(contention);
      REMOVE(CURRENT);
      IF NOT collision THEN
        BEGIN
        transmission = TRUE;
        HOLD(trtime - propdly + deference);
        INSPECT FIRST(outbuf(n)) WHEN packet DO
```

```
            BEGIN
            TRANSFER(packet(no), inbuf(dest));
            IF CARDINAL(inbuf(dest)) EQL 1 THEN
              ACTIVATE rxnode(dest);
            END;
          notrans(n) = notrans(n) + 1;
          started = transmission = colloision = FALSE;
          ACTIVATE mtbusq;
          END;
        ELSE
          BEGIN
          started = FALSE;
          INSPECT FIRST(outbuf(n) WHEN packet DO
            BEGIN
            rcount = rcount + 1;
            dcount = dcount + 1;
            backoff(dcount,bkoftime);
            END;
          HOLD(bkoftime);
          END of started;
      END of rcountl leq 15;
    ELSE
      BEGIN
      REMOVE(FIRST(outbuf(n)));
      txpktabt(n) = txpktabt(n) + 1;
      END of rcountl gtr 16 and not empty statement;
    END;
```

```
      PASSIVATE;
      END of true loop;
    END of txnoad;
```

Before the operation of the above transmission part of the node can be explained, the code for the packet entity must be given as the operation of the former is closely linked with that of the latter. The code for the packet activity follows:

```
ACTIVITY packet(no); INTEGER no;
  BEGIN
  BOOLEAN ack, retry;
  INTEGER srce, dest, rcount, dcount;
  REAL endtime;
  REAL ARRAY timoftx(1:nonodes);
  END of packet;
```

As can be seen, the packet activity is purely a data structure and has no action description. The txnoad activity inspects these data and takes action appropriate to their values.

Referring to the node activity, after the local declarations the first statement is the WHILE TRUE loop. This forms the basic framework within which the node activity takes place. The next is the WHILE NOT EMPTY(outbuf(n)) statement. This loop is active while the transmitter queue has information to be sent. When this is empty, the activity passivates.

If the queue is not empty then the node inspects the first packet in the queue and stores one the values. If this is the first time that a transmission is to be attempted(retry flag is false) then the node activity delays itself for a time equal to the station delay as determined earlier in this appendix.

The next action taken is determined by the value of the local copy of the retry count for the current packet. For the first attempt at transmission, activity on the Ethernet bus is checked. This corresponds to the carrier sense operation of the actual system. If the bus is busy then the node activity waits in the 'busbusy' queue. As explained later the node that is currently busy on the bus will flush this queue on completion of its activity.

If there is no activity then the node checks to see if the collision window time frame has been started by checking the status of the 'started' flag. If it has not then the flag is set and an activity that clears the contention status is scheduled to run after a time equal to the network propagation delay. This must run regardless of whether there is contention or not.

If there is contention then the collision flag is set and the contention clearing algorithm is re-scheduled to run at the end of the collision enforcement period.

After either of these last two alternatives have been executed, the node activity waits in the contention queue until it is re-

activated by the contention clearing activity.

On reactivation the node activity removes itself from the queue (REMOVE(CURRENT)) and checks to see if a collision did in fact occur. If none occurs then the 'transmission' flag is set indicating that the bus is busy. The activity again delays itself for the remainder of the transmission time which is equal to the packet transmission time plus the deference time minus the time which has already elapsed. The elapsed time is equal to the propagation delay which occured during the collision window interval.

After this delay the destination address of the packet is checked and the packet is transferred to the input queue of the destination node. If the receiver section of the latter is not active then it is activated (ACTIVATE rxnode(dest)). After this some data logging activity is performed, all the bus flags are cleared and any other nodes waiting on an idle bus are flushed from the 'busbusy' queue by the 'mtbusque' activity. After this the cycle repeats and the program once again checks the contents of the input queue at the beginning of the node activity.

If the collision flag is set at the end of the collision window interval, the bus flag is cleared and the retry counters are incremented. A new backoff interval is then calculated according to the algorithm described earlier. The node then waits for this period (HOLD(bkoftime)) and then execution returns to the start

of this activity where the queue is checked and transmission is again attempted.

If the number of retries for a packet exceeds 16 then the transmission attempt is aborted. This number of 16 includes the original transmission attempt.

The receiver part of the node entity will now be discussed. The code for this is given below:

```
ACTIVITY rxnoad(n); INTEGER n;
  BEGIN
  INTEGER temp;
  FOR dum =1 WHILE TRUE DO
    BEGIN
    FOR dum = 1 WHILE NOT EMPTY(inbuf(n)) DO
      BEGIN
      HOLD(rxanaltime);
      INSPECT FIRST(inbuf(n)) WHEN packet DO
        BEGIN
        REMOVE(pkt(no));
        pkt(no) = NONE;
        life(no) = TIME - life(no);
        IF life(no) GTR maxproptime THEN
          maxproptime = life(no);
        IF life(no) LSS minproptime THEN
          minproptime = life(no);
        retrytotl = rtrytotl + rcount;
```

```
        IF rcount GTR Ø THEN
          noretries = noretries + 1;
        nofterm = nofterm + 1;
        END of not empty loop;
      PASSIVATE;
      END of true loop;
    END of rxnoad;
```

The structure of tnis entity is very similar to that of the transmitting section. It has the TRUE loop which forms the basic cyclic function of the receive process. Within this is the 'queue not empty' loop which is opened by the statement 'FOR dum = 1 WHILE NOT EMPTY(inbuf(n)) DO'. This checks the receive buffer and causes the receive function to be performed as long as the input queue is not empty.

The first statement within this loop delays the receive function by a time equal to the receive processing time. After this the packet is removed from the queue and its reference variable is set equal to the null vector. This de-references that particular packet entity completely and allows it to be removed from the simulation. This releases the memory space that it occupied and allows it to be re-used.

The next few statements collect some of the raw data for the report generation. The first of these calculate the time taken for the packet to be transmitted across the medium. The variable 'life(no)' is set equal to the time when the packet is released

into the system. The next few statements check for maximum and minimum propagation times and accumulate the count of packets that experienced re-transmissions.

When the input buffer is empty the execution sequence passes from the 'NOT EMPTY' loop to the next executable statement. This is the 'PASSIVATE' statement. The activity becomes dormant until it is activated by a transmitter node inserting a packet and checking for an empty queue.

The next activity of importance is the packet generation entity. It is given below:

```
ACTIVITY pktgen;
  BEGIN
  INTEGER start, finish, e, s, t;
  s = t = 1;
  e = 3;
  FOR seq = (1,1,nopkts) DO
    BEGIN
    pkt(seq) = NEW packet(seq);
  try1:
    start = randint(1,nonodes,s);
    finish = randint(1,nonodes,e);
    IF start EQL finish THEN GOTO try1;
    INSPECT pkt(seq) WHEN packet DO
      BEGIN
      srce = start;
```

```
      dest = finish;
      END;
    INCLUDE(pkt(seq),nodebuf(start));
    IF CARDINAL(nodebuf(start)) EQL 1 THEN
      ACTIVATE pktserver(start);
    IF seq EQL nopkts//2 THEN ACTIVATE thruputlog;
    IF seq EQL (k*nopkts//10) THEN ACTIVATE intvldatalog;
    HOLD(negexp(rate,t));
    END;
  ACTIVATE thruputlog;
  END of pktgen;
```

The first few statements declare local variables and give initial values to the seeds used in the random number generators. The 'FOR' loop defines the basic loop of the activity. This representation is peculiar to this version of the Algol language and means 'FOR seq =1 STEP 1 UNTIL nopkts DØ'. The variable seq gives each packet its identity number and runs from 1 to 'nopkts', an input variable which determines the number of packets generated for the run.

The next statement generates a new instance of a packet entity and references it with the name of 'pkt'. The next two statements generate the start and finish addresses for the packet. These are generated by random drawing procedures which have a uniform distribution in the defined range. In this case it is in the range of the number of nodes in the system. The values 's' and 'e' are seeds which define a starting point and are used to

determine the next number in the sequence.

After this a check is made to see if the source and destination addresses are the same. If they are, another set of addresses is drawn in an attempt to obtain dispárate addresses.

Once the addresses are determined the packet is placed in a queue called 'nodebuf' of an entity named 'pktserver'. The reason for this is explained later. If this queue is empty then the 'pktserver' is activated. After this data logging activities are initiated at certain points in the process of the simulation.

After these operations the packet generator delays for a random time that has a negative exponential distribution with a mean equal to 1/rate. The value 't' serves the same function as those in the drawing procedures described above.

After all the packets have been generated, execution proceeds to the next executable statement which is 'ACTIVATE thruputlog'. This activates some more data logging procedures. After this the action sequence for the packet generator comes to an end and the activity ceases to exist.

It was mentioned earlier that the generated packets were placed in the queue of the activity 'pktserver'. Its code is as follows:

```
ACTIVITY nodelink(n); INTEGER n;
  FOR dum = 1 WHILE TRUE DO
```

```
BEGIN
FOR dum = 1 WHILE NOT EMPTY(nodebuf(n) DO
  BEGIN
  FOR dum = 1 WHILE CARDINAL(outbuf(n)) GEQ txbufsize DO
  HOLD(trtime + stationdelay);
  INSPECT FIRST(nodebuf(n)) WHEN packet DO
    BEGIN
    life(no) = TIME;
    TRANSFER(pkt(no)),outbuf(n));
    END;
  IF CARDINAL(outbuf(n)) EQL 1 THEN
  ACTIVATE txnode(n);
  END of not empty loop;
PASSIVATE;
END of true loop and nodelink;
```

This entity supplies the packets to the transmitter node at a rate at which it can handle them. The latter statement requires some explanation as the packet generator is nominally supposed to determine the rate at which packets are supplied to the network.

The packet generator together with the nodelink entities provide the load to the communication network. The packet generator determines the rate at which packets are produced but this makes no allowances for the characteristics and behaviour of the individual nodes. The nodelink entity moniters the size of the transmitter queue with which it is associated and limits it to a size which is typical of a practical system. It thus simulates

the action of the host of a network node while the queue size control emulates the number of buffers that are available for passing data between host and communications processor.

This entity has the infinite 'while' loop as its basic structure. It has an input queue which receives the data packets from the packet generator and transfers them to the transmitter queue of its associated transmitting node. It does this as soon as possible subject to the condition that the transmitter queue is never longer than four packets. If it is then this entity waits until there are less than four before transferring the next packet. Its input queue is not limited in length except by the constraints of the simulation language itself. For light loads the system is thus driven by the characteristics of the packet generator but as the load increases the buffers become full and the network determines the behaviour of the system.

There are three other entities essential for the function of the model which have been mentioned but not explained. These are 'mtbusque', 'cntentnclr' and 'backoff'.

The activity 'mtbusque' is shown below:

```
ACTIVITY mtbusque;
FOR dum =1 WHILE TRUE DO
  BEGIN
  FOR dum = 1 WHILE NOT EMPTY(busbusy) DO
    ACTIVATE FIRST(busbusy);
```

```
  PASSIVATE;
  END of true loop;
```

It does the simple task of flushing all transmitter activities from the 'busbusy' queue when a node has successfully completed the transmission of a packet. The transmitter entities place and suspend themselves in this queue if they find that the medium is busy ('transmission' flag is set). This entity activates the first activity in the queue. The first action of the re-activated task is to remove itself from the queue and do other functions before returning control to 'mtbusque'. This then activates the next entity, and so on until the queue is empty. It ceases activity when the queue is empty.

The statements for 'cntentnclr' are as follows:

```
ACTIVITY cntentnclr;
  FOR dum = 1 WHILE TRUE DO
  BEGIN
  FOR dum = 1 WHILE NOT EMPTY(contention) DO
    ACTIVATE FIRST(contention);
  collision = FALSE;
  PASSIVATE;
  END of true loop;
```

If contention arises on the bus when one or more transmitter entities attempt to use the bus after the flag 'started' has been set but before 'transmission' has been set, then all contending entities are placed in the 'contention' queue. The above entity

is activated after a delay equal to the enforcement period. It works in the same manner as the 'mtbusque' activity but clears the 'collision' flag after flushing the queue. This does not give rise to an indeterminate situation when the re-activated entities test the state of this flag. Each re-activated entity gains control as soon as it is re-activated and checks the status of this flag before it can be cleared by the above activity.

The last entity to be discussed that forms an active part of the simulation is the procedure 'backoff'. This implements the backoff algorithm that is used to determine the backoff time in the event of a collision. It is given below:

```
PROCEDURE backoff(bkofint,bkoftime); INTEGER bkofint;
                                     REAL bkoftime;
  BEGIN
  IF bkofint GTR 10 THEN bkofint = 10;
  bkoftime = slot * RANDINT(0,2^bkofint,seedr);
  END of backoff;
```

In accordance with the algoritm, the first line limits the retry integer, one of the input parameters, to a maximum value of 10. The next statement takes this integer and uses it to draw a random number in the range between 0 and the square of the backoff integer. This random number is then multiplied by the slot time to determine the backoff interval. It is returned as the output parameter.

The next aspect to be discussed is that of data logging for the output of the results. There are four major parameters monitered in the simulation. The first of these is throughput. This simply entails counting the number of packets that reached their destinations and dividing that by the time that it took for all of them to reach their destinations. A slight problem arises in a simulation of this sort in the choice of a suitable time interval for this measurement. Firstly, the end point of this period cannot be coincident with the end of the simulation period as the latter is chosen to be arbitrarily long to ensure that all simulation activity has ceased at the termination of the run. The only other suitable end point is at the completion of the generation of all packets. Not all packets could have reached their destination at this time, but the resultant loss of information from this can be made small by the generation of a suitably large number of packets in the simulation.

Secondly the start point of this period cannot be the beginning of the simulation as no packets will have been generated, let alone arrive at their destination. Also stable conditions will take some time to be established so false readings will be obtained if the early part of the simulation is included in the timing interval. A way to avoid this is to start timing at the halfway point in the process of generating the packets for the run.

An activity is written to perform this function and is called

thruputlog. The two statements 'IF seq EQL nopkts//2 THEN ACTIVATE thruputlog' and 'ACTIVATE thruputlog' in the packetgen activity activate this entity at the start and end points of the time interval respectively.

Another activity called 'periodatalog' is activated at intervals of one tenth the total number of packets to be generated in the 'packetgen' activity. This logs the average length of the transmit buffers and determines the average throughput for the interval since the activity was last activated.

A procedure called 'report' which is executed at the end of the simulation period prints the parameters discussed above and determines the applied load by dividing the total number of packets generated by the time noted by 'thruputlog' at the end of the generation period.

The average propagation time is also determined and printed as well as the maximum and minimum propagation times determined during the simulation. The procedure is listed below:

```
PROCEDURE report;
  BEGIN
  INTEGER i;
  REAL sum, average;
  WRITE(seq,'packets were generated');
  WRITE(nofterm,'packets reached destination);
  FOR i = (nofterm//2,1,nofterm) DO
```

```
    sum = sum + life(i);
  average = sum/(nofterm - nofterm//2);
  WRITE('average transmission time',average,'secs');
  WRITE('minimum pkt propagation time',minproptime,'secs');
  WRITE('maximum pkt propagation time',maxproptime,'secs');
  WRITE('applied load:',seq/endofintvl,'pkts/sec');
  WRITE('throughput:',throughput,'pkts/sec');
  WRITE('no of pkts timed',timed);
  WRITE('no of pkts aborted',txpktabort);
  END of report;
```

The first two 'WRITE' statements in this procedure allow the number of packets that were generated and those that reached their destinations, to be checked. This check is done to make sure that the simulation period is long enough.

The second last 'WRITE' statement allows a check on the number of packets that were timed. This is done to allow a visual check that the sample is large enough to obtain reliable results from the run.

Before the last few points of the program are discussed the complete listing will be given.

```
BEGIN
INTEGER nonodes,nopkts;
FORMAT f1(s3,i4,s7,a1);
INTEGER mllim,mulim,mtxbufsize,low,incr,high,rate;
```

```
REAL msimperiod,mtrtime,mtxanaltime,mrxanaltime,
  mslot,mpropdly,menforcement,mdeference,mtimeout;
BOOLEAN macknowledge;
READ(nonodes,nopkts);
WRITE(fl,'CSMA/CD Single Bus System with',nonodes, ' nodes');
READ(msimperiod,mtxbufsize,mtrtime,mtimeout);
READ(mpropdly,mdeference,mslot,menforcement);
READ(mtxanaltime,mrxanaltime,low,incr,high);
READ(macknowledge);
READ(mllim,mulim);
FOR rate = (low,incr,high) DO

  SIMULA BEGIN
  INTEGER dum,seedr,seq,llim,ulim,txbufsize,r,nofterm,timed,
    retrytotl,noretries,k;
  REAL simperiod,trtime,txanaltime,rxanaltime,slot,propdly,
    enforcement,endofintvl,throughput,maxproptime,minproptime,
    deference,timeout,stationdelay,processfactor;
  BOOLEAN transmission,started,collision,acknowledge;
  SET busbusy,contention;
  ELEMENT mtbusq,contentionclr,thruputlog,intvldatalog;
  INTEGER ARRAY txpktabt(1:nonodes),notrans(1:nonodes),
    retrans(1:nonodes);
  REAL ARRAY life(1:nopkts),tax(1:10),bufutil(1:10),
    periodlifeavge(1:10);
  SET ARRAY comsbuf(1:nonodes),outbuf(1:nonodes),inbuf(1:nonodes),
    nodebuf(1:nonodes),tolist(1:nonodes);
  ELEMENT ARRAY txnode(1:nonodes),rxnode(1:nonodes),
```

```
    pkt(1:nopkts),pktserver(1:nonodes),pktimer(1:nopkts);
  FORMAT
    f1(i5,s28,i5,s6,a1),
    f2(s26,r11.5,s6,a1),
    f3(i5,s29,a1),
    f4(:nonodes:(x2,i4),a1),
    f5(s12,i4,s9,a1),
    f6(s17,i6,a1),
    f7(s22,r10.3,s4,r10.3,s6,a1),
    f8(s31,i4,a1);
  LIST
    l1(FOR r = (1,1,nonodes) DO notrans(r)),
    l2(FOR r = (1,1,nonodes) DO txpktabt(r));

  PROCEDURE report;
    BEGIN
    INTEGER i;
    REAL sum,average;
    WRITE (f3,seq,'packets were generated.');
    WRITE ('transmissions from nodes');
    WRITE (l1,f4);
    FOR i = (nofterm//2,1,nofterm) DO
      sum = sum + life(i);
    average = sum/(nofterm - nofterm//2 + 1);
    WRITE(f2,'average transmission time:', average, ' secs');
    WRITE (f2,'min pkt propagation time: ',minproptime,
                                          ' secs.');
    WRITE (f2,'max pkt propagation time: ',maxproptime,
```

```
                                                ' secs.');
    WRITE (f5,'applied load: ', seq/endofintvl,' pkts/sec.');
    WRITE (f5,'throughput  : 7, throughput, ' pkts/sec.7);
    WRITE (f6,' no of pkts timed: ' ,timed);
    WRITE (f8,'average no of retries/pkt:',retrytotl/nofterm);
    WRITE (f8,'no of pkts that had collisions:', noretries);
    WRITE ('no of pkts aborted , each node:');
    WRITE (l2,f4);
    HPRINT (bufutil,tax);
    HPRINT (periodlifeavge,tax);
    END of report;

  PROCEDURE rmvepktimer(no); INTEGER no;
    BEGIN
    remove(pktimer((no));
    pktimer(no) = NONE;
    IF seq GEQ llim AND seq LSS ulim THEN
      WRITE('timo cancelled',no,time);
    END of rmvepktimer;

  PROCEDURE backoff(bkofint,bkoftime); INTEGER bkofint;
                                       REAL bkoftime;
    BEGIN
    IF bkofint GTR 1Ø THEN bkofint = 1Ø;
    bkoftime = slot * RANDINT(Ø,2^bkofint,seedr);
    END of backoff;
```

```
ACTIVITY periodatalog;
  BEGIN
  INTEGER l,total,last,number;
  REAL sumlife;
  FOR dum = 1 WHILE TRUE DO
    BEGIN
    total = sumlife = 0;
    FOR l = (1,1,nonodes) DO
      total = total + CARDINAL(outbuf(l));
    bufutil(k) = total/nonodes;
    FOR l = (last+1,1,nofterm) DO
      sumlife = sumlife + life(l);
    number = nofterm - last - 1;
    periodlifeavge(k) = sumlife/number;
    last = nofterm;
    tax(k) = TIME;
    k = k + 1;
    PASSIVATE;
    END;
  END of periodatalog;

ACTIVITY throughputlog;
  BEGIN
  INTEGER startno;
  REAL startofintvl;
  startno = nofterm;
  startofintvl = TIME;
  PASSIVATE;
```

```
    timed = nofterm - startno;
    endofintvl = TIME;
    throughput = timed/(endofintvl - startofintvl);
    END of throughputlog;

  ACTIVITY packet(no); INTEGER no;
    BEGIN
    INTEGER srce,dest,rcount,dcount;
    REAL endtime;
    REAL ARRAY timoftx(1:nonodes);
    BOOLEAN ack,retry;
    END of packet;

  ACTIVITY mtbusque;
    FOR dum = 1 WHILE TRUE DO
      BEGIN
      FOR dum = 1 WHILE NOT EMPTY(busbusy) DO
      ACTIVATE FIRST(busbusy);
      PASSIVATE;
      END of true loop;

  ACTIVITY cntentnclr;
    FOR dum = 1 WHILE TRUE DO
      BEGIN
      FOR dum = 1 WHILE NOT EMPTY(contention) DO
        ACTIVATE FIRST(contention);
      collision = FALSE;
      PASSIVATE;
```

```
    END of true loop;
ACTIVITY timocheck(n); INTEGER n;
  BEGIN
  BOOLEAN found;
  FOR dum = 1 WHILE TRUE DO
    BEGIN
    FOR dum = 1 WHILE CARDINAL(tolist(n)) NEQ Ø DO
      BEGIN
      INSPECT FIRST(tolist(n)) WHEN packet DO
        REACTIVATE CURRENT AT endtime;
      INSPECT FIRST(tolist(n)) WHEN packet DO
        IF endtime - Ø,Ø1 * trtime LEQ TIME THEN
          BEGIN
          REMOVE(pktimer(no));
          pktimer(no) = NONE;
          INCLUDE(pkt(no),outbuf(n));
          retry = FALSE;
          retrans(n) = retrans(n) + 1;
          END of remove loop;
      END of cardinal # Ø loop;
    PASSIVATE;
    END of true loop;
  END of timocheck;

ACTIVITY nodelink(n); INTEGER N;
  FOR dum = 1 WHILE TRUE DO
    BEGIN
    FOR dum = 1 WHILE NOT EMPTY(nodebuf(n)) DO
```

```
        BEGIN
        FOR dum = 1 WHILE CARDINAL(outbuf(n)) GEQ txbufsize DO
          HOLD(trtime + stationdelay);
        INSPECT FIRST(nodebuf(n)) WHEN packet DO
          BEGIN
          life(no) = TIME;
          TRANSFER(pkt(no),outbuf(n));
          END;
        IF CARDINAL(outbuf(n)) EQL 1 THEN
          ACTIVATE txnode(n);
        END of not empty loop;
      PASSIVATE;
      END;


  ACTIVITY txnoad(n); INTEGER n;
    BEGIN
    INTEGER rcountl;
    REAL bkoftime;
    FOR dum = 1 WHILE TRUE DO
      BEGIN
      FOR dum = 1 WHILE NOT EMPTY(outbuf(n)) DO
        BEGIN
        INSPECT FIRST(outbuf(n)) WHUEN packet DO
          BEGIN
          rcountl = rcount;
          IF NOT retry THEN HOLD(stationdelay);
          IF NOT retry AND acknowledge THEN
```

```
        BEGIN
        pltimer(no) = PROC(pkt(no));
        IF seq GEQ llim AND seq LSS ulim THEN
          WRITE('timo logged',n,no,TIME);
        INCLUDE(pktimer(no),tolist(n));
        IF CARDINAL(tolist(n)) EQL 1 THEN
          ACTIVATE timocheck(n);
        IF endtime EQL Ø THEN timoftx(n) = TIME;
          endtime = TIME + timeout;
        END;
      retry = TRUE;
      END of retry;
    IF rcount1 LEQ 15 THEN
      BEGIN
      IF transmission THEN
        BEGIN
        WAIT(busbusy);
        REMOVE(CURRENT);
        END;
      ELSE
        BEGIN
        IF NOT started THEN
          BEGIN
          started = TRUE;
          ACTIVATE contentionclr DELAY propdly;
          END;
        ELSE
          BEGIN
```

```
    collision = TRUE;
    REACTIVATE contentionclr DELAY enforcement;
    END;
  WAIT(contention);
  IF seq GEQ llim AND seq LSS ulim THEN
    WRITE('cntnend',n collision,TIME);
    REMOVE(CURRENT);
    IF NOT collision THEN
      BEGIN
      transmission = TRUE;
      HOLD(trtime - propdly + deference);
      INSPECT FIRST(outbuf(n)) WHEN packet DO
        BEGIN
        transfer(pkt(no),inbuf(dest));
        IF seq GEQ llim AND seq LSS ulim THEN
        WRITE('pkt xferred',n,no,dest,TIME,
                CARDINAL(inbuf(dest)));
        IF CARDINAL(inbuf(dest)) EQL 1 THEN
          ACTIVATE rxnode(dest);
        END;
      notrans(n) = notrans(n) + 1;
      started = transmission = collision = FALSE;
      ACTIVATE mtbusq;
      END;
    ELSE
      BEGIN
      started = FALSE;
```

```
                INSPECT FIRST(outbuf(n)) WHEN packet DO
                  BEGIN
                  rcount = rcount + 1;
                  dcount = dcount + 1;
                  backoff(dcount,bkoftime);
                  IF seq GEQ llim AND seq LSS ulim THEN
                    WRITE('backoff',n,no,dcount,bkoftime,
                          TIME);
                  END;
                HOLD(bkoftime);
                END;
              END of started;
            END of rcountl leq 15;
          ELSE
            BEGIN
            REMOVE(FIRST(outbuf(n));
            txpktabt(n) = txpktabt(n) + 1;
            END of rcount gtr 16 and not empty;
          END;
        PASSIVATE;
        END of true loop;
      END of txnoad;

  ACTIVITY rxnoad(n); INTEGER n;
    BEGIN
    INTEGER temp;
    FOR dum = 1 WHILE TRUE DO
      BEGIN
```

```
FOR dum = 1 WHILE NOT EMPTY(inbuf(n)) DO
  BEGIN
  HOLD(rxanaltime);
  INSPECT FIRST(inbuf(n)) WHEN packet DO
    IF ack THEN
      BEGIN
      REMOVE(pkt(no));
      IF acknowledge THEN rmvepktimer(no);
      pkt(no) = NONE;
      life(no) = TIME - life(no);
      IF life(no) GTR maxproptime THEN
        maxproptime = life(no);
      IF life(no) LSS minproptime THEN
        minproptime = life(no);
      retrytotl = retrytotl + rcount;
      IF rcount GTR 0 THEN
        noretries = noretries + 1;
      nofterm = nofterm + 1;
      IF seq GEQ llim AND seq LSS ulim THEN
        WRITE('pkt removed', n.no,rcount,TIME);
      END;
    ELSE
      BEGIN
      IF acknowledge THEN
        BEGIN
        rmvepktimer(no);
        ack = TRUE;
```

```
            temp = srce;
            srce = dest;
            dest = temp;
            TRANSFER8pkt(no),outbuf(n));
            IF CARDINAL(outbuf(n)) EQL  1 THEN
              ACTIVATE txnode(n);
            IF seq GEQ llim AND seq LSS ulim THEN
            WRITE('ack pkt sent',n,no,srce,dest TIME);
            END;
          END;
        END of not emppty loop;
        PASSIVATE;
        END of true loop;
      END of rxnoad;

  ACTIVITY pktgen·
    BEGIN
    INTEGER start,finish,e,s,t;
    s = t = 1;
    e = 3;
    FOR seq = (1,1,nopkts) DO
    BEGIN
      pkt(seq) = NEW packet(seq);
      try1:
      start = RANDINT(1,nonodes,e);
      finish = RANDINT(1,nonodes,e);
      IF start EQL finish THEN GOTO try1:
      INSPECT pkt(seq) WHEN packet DO
```

```
        BEGIN
        srce = start;
        dest = finish;
        IF NOT acknowledge THEN ack = TRUE;
        END;
      INCLUDE(pkt(seq),nodebuf(start));
      IF seq GEQ llim AND seq LSS ulim THEN
      WRITE('main pkt generated', seq,start,finish.TIME);
      IF CARDINAL(nodebuf(start)) EQL 1 THEN
        ACTIVATE pktserver(start);
      IF seq EQL nopkts//2 THEN ACTIVATE thruputlog;
      IF seq EQL (k*nopkts//10) THEN ACTIVATE intvldatalog;
        HOLD(NEGEXP(rate,t);
      END;
    ACTIVATE thruputlog;
    END of pktgen;

k = 1;
seedr = 5;
processfactor = 1;
simperiod = msimperiod; txbufsize = mtxbufsize;
trtime = mtrtime; timeout = mtimeout;
propdly = mpropdly; deference = mdeference;
slot = mslot; enforcement = menforcement;
txanaltime = mtxanaltime; rxanaltime = mrxanaltime;
acknowledge = macknowledge; llim = mllim; ulim = mulim;
IF acknowledge THEN
```

```
    WRITE('reply to datagram included in simulation')+
  ELSE
    WRITE('simulation of datagram service only');
  WRITE(f1,nopkts,'pkts assigned. simperiod is',
    simperiod,' secs');
  WRITE(f2,'transmit buffer size:',txbifsize);
  WRITE(f2,'pkt transmission time:',trtime,' secs);
  WRITE(f2,'propagation delay:',propdly,' secs');
  WRITE(f2,'deference time   :',deference,' secs');
  WRITE(f2,'slot time        :',slot,' secs');
  WRITE(f2,'station processing factor:',processfactor,' p u');
  stationdelay = (1,978 * trime + 0,001407) * processfactor;
  WRITE(f2,'station delay          :',stationdelay,' secs.');
  WRITE(f2,'transmit analysis time  :',txanaltime, ' secs.');
  WRITE(f2,'receive analysis time:',rxanaltime,' secs.');
  WRITE(f2,'packet intervals: max',max,' min:',min,' secs.');
  minproptime = simperiod;
  FOR r = (1,1,nonodes) DO
    BEGIN
    txnode(r) = NEW txnoad(r);
    rxnode(r) = NEW rxnoad(r);
    pktserver(r) = NEW nodelink(r);
    ACTIVATE txnode(r);
    ACTIVATE rxnode(r);
    END;
  contentionclr = NEW cntentnclr;
  mtbusq = NEW mtbusque;
  intvdatalog = NEW periodatalog;
```

```
    thruputlog := NEW throughputlog;
    ACTIVATE NEW pktgen;
    HOLD(simperiod);
    ACTIVATE thruputlog;
    report;
    END of simula block;
END of rate loop;
END of program
```

The simulation model resides within the SIMULA BEGIN - END block. In order to increase the flexibility of this model, all the sizes of the main parameters are defined in terms of variables. The Simula block thus has an outer block which establishes the value of these variables.

In order to speed up the process of simulation this outer block generates a range of values for the applied load and causes the simulation to be repeated for each value generated.

The main body of the program first of all receives the parameter data from the outer block of the program. This is an anomalous mode of programming but seems to be the only way that this compiler permits these parameters to be passed into the simulation block.

The next section prints out the values of the parameters received while the last part controls firstly the creation of the model and starts its operation. After delaying for the simulation

period the reporting of the results is initiated.

A debugging aid is included in the complete listing which is not shown in the earlier partial listings. This consists of the statements 'IF seq GEQ llim AND seq LSS ulim THEN', etc. By choosing the values of llim and ulim, this printout can be suppressed or the activity of the model can be displayed for a selected time period. Each statement names the activity that is active and displays a number of parameters that are relevent at that time instant. This is extremely useful for verifying the operation of the model.

APPENDIX C

FULL DEFINITION AND SIMULATION MODEL OF THE MAP LAN.

The MAP network has been introduced in chapter 4. A more rigorous description is given in this appendix followed by a description of the simulation model developed for this study.

THE MAP NETWORK

The MAP network is a logical token passing ring superimposed on a bus structure. Currently two mutually exclusive methods of signalling are used for communication over the network. The first is the broadband technique using commonly available CATV components for the communication system. The information signalling rate is currently 10 Mbit/sec. This system is characterised by the requirement of a headend remodulator as the broadband system allows communication in only one direction on each band. The headend receives the transmissions from all the nodes on one band, decodes and reconstitutes the signal and then broadcasts it to all nodes on an upper band. This has two distinct disadvantages. The first is that the integrity of the system is dependant on that of the headend. This is the one feature that distributed systems are normally designed to avoid. The second is that because of its crucial role in the operation of the system it is a costly item. This implementation of the network supports all seven layers of the ISO reference model.

The second signalling method uses the carrierband technique. This was recently introduced as part of the Enhanced Performance Architecture(EPA) in an attempt to improve the performance of the network. This is defined in chapters 12 and 13 of reference 13. This technique is similar to baseband in that there is no carrier in the signal but the signalling is phase coherent This avoids the use of a headend and thinner co-axial cable can be used. Currently the signalling rate is 5 Mbit/sec.

As part of the EPA this implementation leaves out OSI layers 3 to 6 in an attempt to improve the performance of the system. These layers deal with internet communication, data integrity and presentation(encoding format). Omission of the internet capability and data presentation layers is understandable because internet communication would be handled by the broadband gateway and co-operating processes would handle data of the same format.

It is however surprising that the fourth layer is omitted as this ensures integrity of the data and could be a serious omission in a manufacturing or process control environment.

NETWORK PARAMETERS

There are several parameters defined for the MAP network, most of which will not be discussed here as they deal mainly with the establishment of the initial token or are used to determine the address of the successor in the token passing sequence. Some of these are also used to allow nodes to enter or leave the network.

They are not discussed here as this study deals only with the performance of a correctly functioning network. These parameters are the transmission path delay, token holding time, station delay and token passing time.

Transmission Path Delay

This parameter is defined as:

Transmission path delay = Longest (physical medium delay +
amplifier delay +
repeater delay)

This is a generalised equation and can be simplified for most practical circuits to the worst case medium delay plus repeater delay. The term 'amplifier' is not referred to in any of the sections giving a fuller description of this parameter.

The physical medium delay is derived from the cable parameters and the permitted signal strengths for transmitters and receivers.

The maximum permitted signal level is 53dBmV (sec 14.9.5.4) while the nominal input level to the receiver is 3 dBmV (sec 14.9.10). Thus the maximum permitted attenuation along the bus segment is 50 db. Referring to the specifications for the Suhner class SA cable, type 17272 which is equivalent to the semi-rigid cable typically used in a MAP network, the cable loss is 3,5 dB/100m at the lowest permitted forward channel frequency of 252 MHz for the 10 Mbit/sec MAP newtwork. Thus the maximum permitted segment

length is 50/3,5 × 100 = 1430 m.

Thus transmission path delay is 4,33 × 2 × 1430 = 12,38 usec. assuming a velocity of propagation of 4,33 nsec/m. along the cable.

This parameter is not directly used in the simulation but gives a useful comparison with that of the Ethernet system. This factor would play a part in the token transmission time at some stage in the course of the circuit of the token but is dependant on the layout of each physical network. It is thus ignored in this study, again yielding a performance slightly better than will be achieved in practice.

Packet format

The format of the IEEE 802.4 packet is shown in figure 4.1. It consists of seven fields excluding the preamble. The preamble performs the dual functions of synchronising the headend receiver to the incoming signal and providing a recovery period for the node electronics between each transmitted packet. The minimum permitted duration is 2 microseconds. The 10 Mbit/sec broadband system has a further requirement that the preamble be a minimum of four bytes long. This is the longer of the two times and thus determines the length of the preamble.

Following the preamble is the Start Delimiter. This marks the beginning of the information in the packet. It is matched by the End Delimiter after which there is no further information.

Between these two delimiters a maximum of 8191 bytes of information is allowed.

After the Start Delimiter comes the Frame Control field. This defines the nature of the packet i.e whether it is a data packet or the token, or a low level protocol function. It is one byte long.

The next two fields are the Destination and Source fields respectively. They are each 6 bytes long and have a definition that is identical to that of the Ethernet address fields.

After the address fields comes the data field. This is absent for the low level protocol control functions but transparently carries the data and control functions of the higher level protocols. It can be of variable length up to a maximum such that the maximum permitted length for the overall packet is not exceeded.

The last field but one is the Frame Check Sequence. This is four bytes long and is identical to that described for the Ethernet CRC.

Packet transmission times

As is described above the header of the MAP packet consists of the following fields and their respective lengths in bytes:

Preamble	4
Start Delimiter	1
Frame Control	1
Destination Address	6
Source Address	6
Frame Check Sequence	4
End Delimiter	1
Total	23

Transmission times of 46 byte and 1500 byte packets are determined so that the performance of the Ethernet and MAP networks can be directly compared. Thus,

Total length of 46 byte packet is 46 + 23 = 69 bytes.

Therefore transmission time is:

69 × 8 × 0,1 = 55,2 microseconds.

Total length of 1500 byte packet is 1500 + 23 = 1523 bytes.

Therefore transmission time is:

1523 × 8 × 0,1 = 1218,4 microseconds.

Token holding time

This parameter determines how long a node may retain the token before it is obliged to pass it on to its successor. It is an arbitrary value that needs to be optimised. For the initial studies this value was initially set at greater than ten times that of the processing delay plus transmission time. As is shown in the main part of this study, it has a great influence on network performance.

Station Delay

This is defined as the time from the arrival of the last piece of information at a node to the placing of the first piece of information on the bus by that same node. The latter occurs as an immediate response to the receipt of the former. This situation arises when a token is passed on, an immediate response is required by the original sending node, or when a node responds to an interrogation from another node.

This parameter is not used explicitly in this simulation. It is taken into account in the processing delay. Indeed the processing delay could be called the station delay but this has no direct equivalent in the Ethernet model.

Token Transmission Time

For this study the token transmission time is taken to include the transmission time over the medium as well as the time it takes the communications processors (the send/receive pair) to execute their decision algorithms. The obvious time to choose for this is the processing delay, but as has been shown in the main text this is excessively long. In order to derive the maximum performance from the network the token passing algorithm could be implemented in hardware. In this simulation the token processing time is assumed to be zero so the token passing time is equivalent to the transmission time. This assumption is justified on the grounds that an upper limit on the system performance is

sought. Lower performance is always easy to achieve!

THE SIMULATION.

The model developed for the MAP simulation differs from the Ethernet model only in the transmitting node entity and a few minor entities that are intimately related to the former. This has been done both for convenience and to keep the models as similar as possible. The rest of this appendix will describe these new entities and name those that have been deleted. A complete listing of the program is given at the end.

As implied above, the transmitting and receiving sections of the node are simulated by two separate entities with all the time delays placed in the transmitting section. This does not impose the limitation which arises in the Ethernet system that a receiving node could be forced to handle more than its maximum load if two or more nodes attempt to transmit to it. In the token passing system only the token holder is permitted to transmit so each transmitting node would be obliged to take its turn in transmitting to that particular node. This system is however subject to the Galloping Bit Syndrome that was discussed with the Ethernet system.

A flow chart of the transmitter activity is shown in figure C.1 with a listing of the node entity following. The numbers on the right side of the listing refer to the line numbers as they appear in the listing at the end of this appendix.

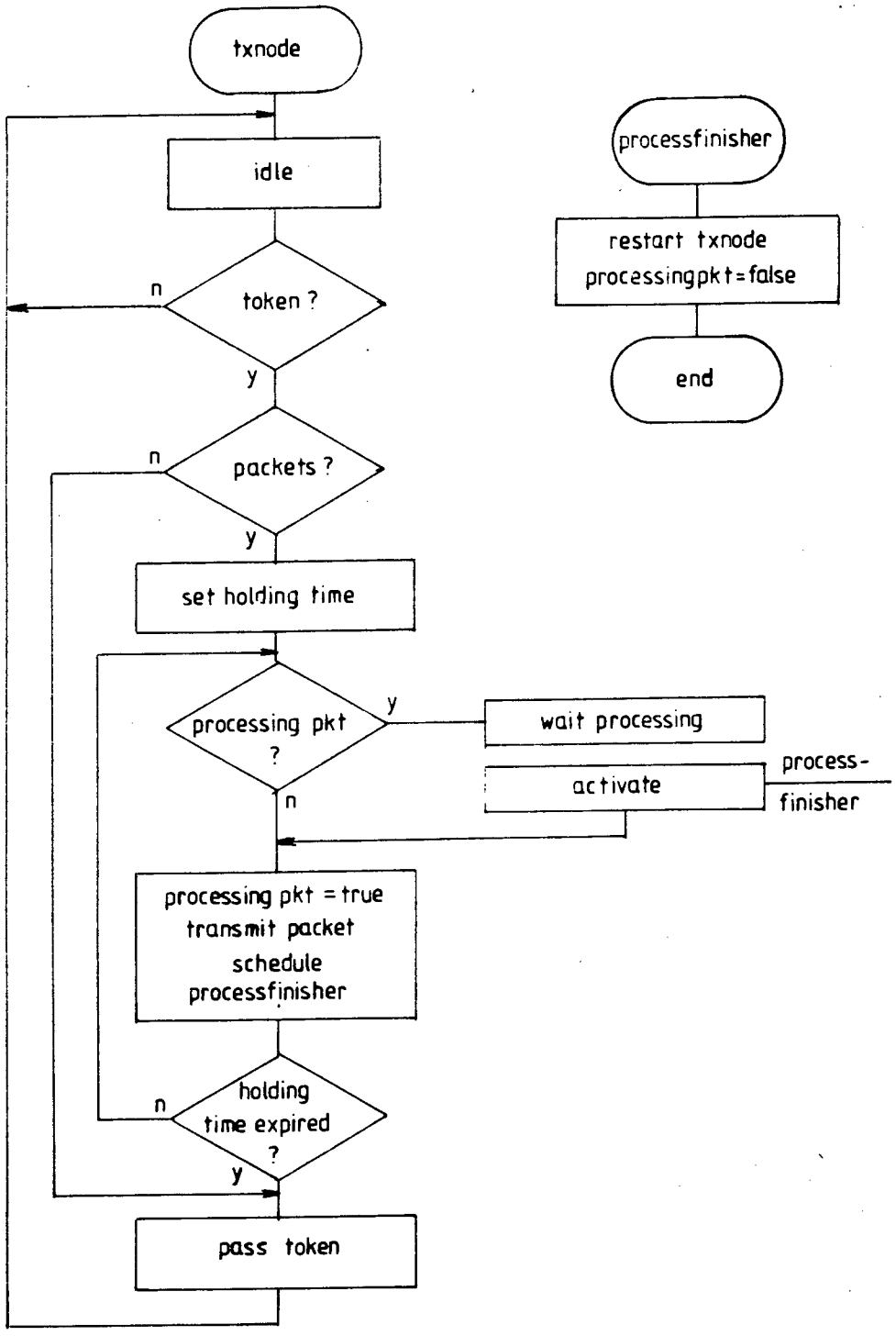

Figure C.1 Flow chart of the node activity

```
ACTIVITY txnoad(n); INTEGER n;                                    118
  BEGIN
    INTEGER nextstation;
    BOOLEAN token, processingpkt, txfirstpkt;
    finishprocessing(n) = NEW processfinisher(n);
    ACTIVATE finishprocessing((n);

    FOR dum = 1 WHILE TRUE DO
      BEGIN
      IF token THEN                                               125
        BEGIN
        FOR dum = 1 WHILE NOT EMPTY(outbuf(n))
            AND (TIME LSS timeout) DO
          INSPECT FIRST(outbuf(n)) WHEN packet DO
            BEGIN                                                 130
            IF processingpkt THEN WAIT(processing);               133
            processingpkt = TRUE;                                 134
            HOLD(trtime);                                         137
            INSPECT FIRST(outbuf(n)) WHEN packet DO
              BEGIN
              TRANSFER(pkt(no),inbuf(dest));                      140
              IF CARDINAL(inbuf(dest)) EQL 1 THEN
                ACTIVATE rxnode(dest);
              ACTIVATE finishprocessing(n) AT TIME +
                                              processingtime;
              END;                                                144
            notrans(n) = notrans(n) + 1;                          147
```

```
        END;
      HOLD(toktrtime);
      INSPECT txnode(nextstation) WHEN txnoad DO          150
        token = TRUE;
      ACTIVATE txnode(nextstation);
      token = FALSE;
      END of token true;
    PASSIVATE;
    timeout = TIME + holdingtime;
    END of true loop;
  END of txnoad;                                          158
```

The above transmitter node entity was originally developed in a simpler form which performed adequately with the initial simulations when the token holding time was considerably longer than the node processing time. When it was necessary to make the holding time equal to the processing time the model was no longer adequate. The node entity shown above is that developed for the short token holding time. It works equally well with long token holding times. All the results shown in this dissertation are obtained using the above entity. When the token holding time was large relative to the processing time the results obtained from the two models were nearly identical.

After declaring a few local variables, the node entity starts by creating another entity related to it that is used to determine the end of the processing interval (processfinisher). After this the usual 'true' loop is entered which defines the action for the

entity. The first thing that is done inside this loop is to check if the node has the token. If it does not possess it then all the node functions are bypassed and the node passivates itself.

When it possesses the token the activity continues (line 125) until the transmitter queue is empty or the token holding time is exceeded. The following line is a conditional statement that checks to see if the node processing time has elapsed. If it has not elapsed the activity gets suspended in the 'processing' queue. At the end of the processing time a separate activity flushes the node activity from the queue and execution continues. The 'processing pkt' parameter is set to indicate that another packet is being processed(line 134).

A further delay occurs equivalent to the transmission time of the packet, after which the packet is transferred to the input queue of the receiving node. In line 141 the length of the input queue is checked and if it was empty prior to the transfer then that node is activated.

After this the activity that flushes this activity from the 'processing' queue (see line 143) is activated at a time equal to the processing time from current. The next line performs a housekeeping function of recording the number of packets transmitted from this node.

This is the end of the loop that is active when the buffer is not

empty and the node is in possession of the token. The next section from line 149 deals with the passing on of the token to the next node. Firstly a delay occurs that is equivalent to the passing on of the token. That node is then activated after its token is set to 'true'. The local token status is set to false and the activity passivates itself.

On being reactivated when the token is next received the token holding time is set and activity continues if all the conditions are correct.

The activity that receives the packets from the generator and buffers them before being placed in the transmitting section of the node is given below:

```
ACTIVITY nodelink(n); INTEGER n;                                    159
  BEGIN
  FOR dum = 1 WHILE TRUE DO
    BEGIN
    FOR dum = 1  WHILE NOT EMPTY(nodebuf(n)) DO
      BEGIN
      FOR dum = 1 WHILE CARDINAL(outbuf((n)) GEQ txbufsize DO 165
        HOLD(trtime + processingtime);                              169
      INSPECT FIRST(nodebuf((n) WHEN packet DO                      171
        BEGIN
        life(no) = TIME;                                            173
        TRANSFER(pkt(no),outbuf(n));                                176
        END;
```

```
      IF CARDINAL(outbuf(n)) EQL 1 THEN
        BEGIN
        INSPECT txnode(n) WHEN txnoad DO                          180
          processingpkt = TRUE;
        REACTIVATE finishprocessing((n) AT TIME + processingtime;
        END;
      END of not empty loop;
    PASSIVATE;
    END of true loop;
  END of nodelink;                                                187
```

This activity is largely the same as that of the Ethernet model excepting in lines 180 to 182 . This deals with the setting up of the processing delay of the transmitting node. It is no longer necessary for this activity to activate the node as this is taken care of by the token passing procedure. All this activity has to do is set the processing delay correctly. Before the latter topic is discussed the last activity dealing with the determination of the processing delay is given:

```
LOCAL PROCEDURE setend;                                           45
ACTIVITY processfinisher(n); INTEGER n;                           105
  BEGIN
  FOR dum = 1 WHILE TRUE DO
    BEGIN
    setend(n);
    IF MEMBER(txnode(n),processing) NEQ NONE THEN
```

```
      BEGIN
      REMOVE(txnode(n));
      ACTIVATE txnode(n);
      END;
    PASSIVATE;
    END;
  END of processfinisher;                                    117

  PROCEDURE setend(n); INTEGER n;                            188
    BEGIN
    INSPECT txnode(n) WHEN txnoad DO
      processingpkt = FALSE;
    END;                                                     192
```

This activity is anomalous as a result of the limitations of the compiler used. It is a one pass compiler with the result that any parameter that is used must have already been defined. The above activity must therefore be declared before it can be referenced in the node activity. However the former must reference the 'processingpkt' flag of the node activity. This would require the former to appear after the node activity resulting in an impossible situation. It is resolved by placing the statement 'LOCAL PROCEDURE' at the beginning of the program, writing a separate procedure which references the attributes of the node activity and placing it after the node activity. This procedure can then be called from the 'processfinisher' activity because it has been partially declared.

The above approach to the handling of the processing delay seems complex compared with having a simple delay in the node activity equivalent to the processing time. In fact, when the program was originally written the simple delay was used. It introduces an error into the simulation process because this delay can only start when the node has the token. This error is small when the holding time is large compared with the processing time. As the two times become comparable, the error increases to as much as 100%.

This scheduling scheme uses a separate activity called 'nodefinisher' to reactivate the node process after the processing delay time has elapsed. It is scheduled for execution after this delay. This activity then checks to see if the relevent node has placed itself into the 'processing' queue(line 110). If it has then it removes it from the queue, resets the 'processing' flag and re-activates the node activity.

So far the method of operation is the same as would occur if a simple 'hold' function were used. The first of the two situations where this method differs, occurs when a packet is placed in the transmitter queue of the node. If the node does not have the token, the communications processor is still capable of preparing a packet for transmission before the token arrives. This is simulated in the current model by the processing time commencing as soon as the packet is placed in the buffer(lines 169 to 182). In a suitably large system the token circuit time could be longer

than the processing time. Thus it is possible for the processing time to have elapsed before the token is received by the node. In this case the node activity would not have been placed in the queue 'processing' but the 'processingpkt' flag would have been cleared by the 'processfinisher' activity and the node can proceed straight to the transmission of the packet.

The second situation occurs when a node has completed transmission of a packet at the time that the token holding time expires. While the node waits to receive the token again, the communications processor can prepare another packet for transmission and be ready to transmit as described above.

This method of simulating the processing delay replicates what is most likely to happen in an actual system. This sounds somewhat vague but no detailed hardware descriptions are available to the author. The description above assumes that the token handling algorithm will be implemented in dedicated hardware, most likely VLSI to gain the maximum performance from the system. Other implementations are possible but there will be a performance degradation. The simulation will then provide an upper bound on the performance obtainable from a practical system.

This completes the discussion on those parts of the token passing model that are different from the Ethernet model. For completeness the full listing of the token passing model is given below. The statements used for debugging the program and checking

its validity are once again included.

```
BEGIN                                                              1
INTEGER nonodes, nopkts;
FORMAT f1(S43, I4, S7, A1);
INTEGER mllim, mulim, mtxbufsize, low, incr, high, rate;
REAL msimperiod, mtrtime, mtoktrtime, mrxanaltime, min, max,
  mprocessfactor, mtimeout, mholdingtime;
READ(nonodes, nopokts);
WRITE(F1, 'GM MAP Token Passing Single Bus System with', nonodes,
                                          ' nodes');
READ(msimperiod, mtxbufsize, mtrtime, mtimeout);                  1Ø
READ(mtoktrtime, mprocessfactor, mholdingtime);
READ(mrxanaltime, low, incr, high);
READ(macknowledge);
READ(mllim, mulim);
FOR rate =(low,incr,high) DO
  BEGIN
  max = 1/rate;
  min = Ø;
    SIMULA BEGIN                                                   19
    INTEGER dum, seedr, seq, llim, ulim, txbufsize, nofterm,
      timed, retrytotl, noretries, start, finish, e, s, t, k, r;
    REAL simperiod, trtime, toktrtime, rxanaltime, processfactor,
      endofintvl, throughput, maxproptime, minproptime, timeout,
      holdingtime,  processingtime;
    SET processing;
    ELEMENT throughputlog, intvldatalog;
```

```
INTEGER ARRAY notrans(1:nonodes);
REAL ARRAY life(1:nopkts), tax(1:1:10), bufutil((1:10),
  periodlifeavge(1:10);
SET ARRAY comsbuf(1:nonodes), outbuf(1:nonodes),                    30
  inbuf(1:nonodes), nodebuf(1:nonodes),tolist(1:nonodes);
ELEMENT ARRAY txnode(1:nonodes), rxnode(1:nonodes),
  pkt((1:nopkts), pktserver(1:nonodes), pktimer(1:nopkts),
  finishprocessing(1:nonodes);
FORMAT f1(I5, S28, I5, S6, A1);
       f2(S26, R10.3, S6, A1),
       f3(I5, S29, A1),
       f4(:nonodes:(x2, I4), A1),
       f5(S12, I4, S9, A1),
       f6(S17, I6, A1),                                             40
       f7(S22, R10.3, S4, R10.3, S6, A1),
       f8(S31, I4, A1);
LIST l1(FOR r = (1,1,nonodes) DO notrans(r));

LOCAL PROCEDURE setend;

PROCEDURE report;
  BEGIN
  INTEGER i;
  REAL sum, average;
  WRITE(f3, seq-1, 'packets were generated.');
  WRITE(f3, nofterm, 'packets reached destination.');               50
  WRITE('transmissions from nodes:');
```

```
    WRITE(11,f4);
    FOR i = (nofterm//2,1,nofterm) DO
      sum = sum + life(i);
    average = sum/(nofterm - nofterm//2 + 1);
    WRITE(f2,'average transmission time:', average,'secs.');
    WRITE(f2,'min pkt propagation time:', minproptime,'secs.');
    WRITE(f2,'max pkt propagation time:', maxprpotime,'secs.');
    WRITE(f5,'applied load:', (seq-1)/endofintvl,
        pkts/sec.');                                              60
    WRITE(f5,'throughput  :', throughput,'pkts/sec.');
    WRITE(f6,'no of pkts timed:', 'pkts/sec.');
    WRITE(f8,'average no of retries/pkt:',retrytotl/nofterm);
    HPRINT(bufutil, tax);
    HPRINT(periodlifeavge, tax);
    END of report;

  ACTIVITY periodatalog;
    BEGIN
    INTEGER 1, total, last, number;
    REAL sumlife;                                                 70
    FOR dum = 1 WHILE TRUE DO
      BEGIN
      total = sumlife = 0;
      FOR 1 = (1,1,nonodes) DO total = total +
        CARDINAL(outbuf(1));
      bufutil(k) = total/nonodes;
      FOR 1 = (last+1,1,nofterm) DO
        sumlife = sumlife + life(1);
```

```
      number = nofterm - last - 1;
      periodlifeavge(k) = sumlife/number;                    80
      last = nofterm;
      tax(k) = TIME;
      k = k + 1;
      PASSIVATE;
      END;
    END of periodatalog;

  ACTIVITY throughputlog;
    BEGIN
    INTEGER startno;
    REAL startofintvl;                                       90

    startno = nofterm;
    startofintvl = TIME;
    PASSIVATE;
    timed = nofterm - startno;
    endofintvl = TIME;
    throughput = timed/(endofintvl - startofintvl);
    END of throughputlog;

  ACTIVITY packet(no); INTEGER no;
    BEGIN
    INTEGER srce, dest, rcount, dcount;                     100
    REAL endtime;
    REAL ARRAY timoftx(1:nonodes);
```

```
    BOOLEAN ack, timofl;
    END of packet;

  ACTIVITY processfinisher(n); INTEGER n;
    BEGIN
    FOR dum = 1 WHILE TRUE DO
      BEGIN
      setend(n);
      IF MEMBER(txnode(n), processing) NEQ NONE THEN          110
        BEGIN
        remove(txnode((n));
        ACTIVATE txnode(n);
        END;
      PASSIVATE;
      END;
    END of processfinisher;

  ACTIVITY txnoad(n); INTEGER n;
    BEGIN
    INTEGER nextstation;                                      120
    BOOLEAN token, processingpkt, txfirstpkt;

    finishprocessing(n) = NEW processfinisher(n);
    ACTIVATE finishprocessing((n);

    FOR dum = 1 WHILE TRUE DO
      BEGIN
      IF token THEN
```

```
BEGIN
FOR dum = 1 WHILE NOT EMPTY(outbuf(n))
                AND (TIME LSS timeout) DO
  INSPECT FIRST(outbuf(n)) WHEN packet DO
    BEGIN                                                    130
    IF seq GEQ llim AND seq LSS ulim THEN
      WRITE('txnode entered', n, no, TIME);
    IF processingpkt THEN WAIT(processing);
    processingpkt = TRUE;
    IF seq GEQ llim AND seq LSS ulim THEN
      WRITE('tr pkt', n, no, TIME);
    HOLD(trtime);
    INSPECT FIRST(outbuf(n)) WHEN packet DO
      BEGIN
      TRANSFER(pkt(no),inbuf(dest));                         140
      IF CARDINAL(inbuf(dest)) EQL 1 THEN
        ACTIVATE rxnode(dest);
      ACTIVATE finishprocessing(n) AT TIME +
                                        processingtime;
      END;
    IF seq GTR llim AND seq LSS ulim THEN
      WRITE('pkt transmitted', n, no, dest, TIME);
    notrans(n) = notrans(n) + 1;
    END;
  HOLD(toktrtime);
  INSPECT txnode(nextstation) WHEN txnoad DO               150
    token = TRUE;
```

```
          ACTIVATE txnode(nextstation);
          token = FALSE;
          END of token true;
        PASSIVATE;
        timeout = TIME + holdingtime;
        END of true loop;
      END of txnoad;

ACTIVITY nodelink(n); INTEGER n;
  BEGIN                                                          160
  FOR dum = 1 WHILE TRUE DO
    BEGIN
    FOR dum = 1 WHILE NOT EMPTY(nodebuf(n)) DO
      BEGIN
      FOR dum = 1 WHILE CARDINAL(outbuf(n)) GEQ txbufsize DO
        BEGIN
        IF seq GEQ llim AND seq LSS ulim THEN
          WRITE('pkt serve hold', n, TIME);
        HOLD(trtime + processingtime);
        END;                                                     170
      INSPECT FIRST(nodebuf(n)) WHEN packet DO
        BEGIN
        life(no) = TIME;
        IF seq GEQ llim AND seq LSS ulim THEN
          WRITE('pkt server xfer', n, no, TIME);
        TRANSFER(pkt(no),outbuf(n));
        END;
      IF CARDINAL(outbuf(n)) EQL 1 THEN
```

```
            BEGIN
            INSPECT txnode(n) WHEN txnoad DO                    180
              processingpkt = TRUE;
            REACTIVATE  finishprocessing(n) AT TIME +
                                              processingtime;
            END;
          END of not empty loop;
        PASSIVATE;
        END of true loop;
      END of nodelink;

    PROCEDURE setend(n); INTEGER n;
      BEGIN
      INSPECT txnode(n) WHEN txnoad DO                          190
        processingpkt = FALSE;
      END;

    ACTIVITY rxnoad(n); INTEGER n;
      BEGIN
    INTEGER temp;
    FOR dum = 1 WHILE TRUE DO
      BEGIN
      FOR dum = 1 WHILE NOT EMPTY(inbuf(n)) DO
        BEGIN
        HOLD(rxanaltime);                                       200
        INSPECT FIRST(inbuf(n)) WHEN packet DO
          IF ack THEN
```

```
        BEGIN
        REMOVE(pkt(no));
        IF acknowledge THEN rmvepktimer(no);
        pkt(no) = NONE;
        life(no) = TIME - life(no);
        IF life(no) GTR maxproptime THEN
          maxproptime = life(no);
        IF life(no) LSS minproptime THEN                    210
          minproptime = life(no);
        retrytotl = retrytotl + rcount;
        IF rcount GTR 0 THEN

          noretries = noretries + 1;
        nofterm = nofterm + 1;
        IF seq GEQ llim AND seq LSS ulim THEN
          WRITE('pkt removed', n,no,rcount,TIME);
        END;
      ELSE
        BEGIN                                               220
        IF acknowledge THEN
          BEGIN
          rmvepktimer(no);
          ack = TRUE;
          temp = srce;
          srce = dest;
          dest = temp;
          TRANSFER8pkt(no) outbuf(n));
          IF CARDINAL(outbuf(n)) EQL  1 THEN
```

```
            ACTIVATE txnode(n);                                    230
          IF seq GEQ llim AND seq LSS ulim THEN
          WRITE('ack pkt sent',n,no,srce,dest TIME);
          END;
        END;
      END of not empty loop;
      PASSIVATE;
      END of true loop;
    END of rxnoad;

  k = 1;
  seedr = 5;                                                       240
  simperiod = msimperiod; txbufsize = mtxbufsize;
  trtime = mtrtime; timeout = mtimeout;
  toktrtime = mtoktrtime; processfactor = mprocessfactor;
  holdingtime = mholdingtime; rxanaltime = mrxanaltime;
  acknowledge = macknowledge; llim = mllim; ulim = mulim;
  IF acknowledge THEN
    WRITE('reply to datagram included in simulation');
  ELSE
    WRITE('simulation of datagram service only');
  WRITE(f1,nopkts, 'pkts assigned. Simperiod is ', simperiod,
                                        ' secs.');               251
  WRITE(f2,'transmit buffer size:', txbufsize);
  WRITE(f2,'pkt transmission time:', trtime, ' secs.');
  WRITE(f2,'token transmission time:', toktrtime,' secs.');
  WRITE(f2,'station processing factor: 'processfactor,' p u');
  processingtime = (1.978*trtime + 0.001407)*processfactor;
```

```
WRITE(f2,'processing time:', processingtime,' secs.');
WRITE(f2,'station holding time:', holdingtime, ' secs.');
WRITE(f2,'receive analysis time:', rxanaltime,' secs.');
WRITE(f7,'packet intervals: max:', max,' min:',' secs.');

minproptime = simperiod;                                    260
FOR r = (1,1,nonodes) DO
  BEGIN
  txnode(r) = NEW txnoad(r);
  INSPECT txnode(r) WHEN txnoad DO
    BEGIN
    nextstation = r - 1;
    IF nextstation EQL 0 THEN
      nextstation = nonodes;
    END;
  rxnode(r) = NEW rxnoad(r);                                270
  pktserver(r) = NEW nodelink(r);
  ACTIVATE txnode(r);
  ACTIVATE rxnode(r);
  END;
intvldatalog = NEW periodatalog;

thruputlog = NEW throughputlog;
INSPECT txnode(nonodes) WHEN txnoad DO
  token = TRUE;
ACTIVATE txnode(nonodes);

s = t = 1;
```

```
    e = 3;                                                          280
    FOR seq = (1,1,nopkts) DO
      BEGIN
      pkt(seq) = NEW packet(seq);
    try1: start = RANDINT(1,nonodes,s);
      finish = RANDINT(1,nonodes,e);
      IF start EQL finish THEN GO TO try1;
      INSPECT pkt(seq) WHEN packet DO
        BEGIN srce = start;
        dest = finish;
        IF NOT ACKNOWLEDGE THEN ack = TRUE;                         290
        END;
      INCLUDE(pkt(seq), nodebuf(start));
      IF seq GEQ llim AND seq LSS ulim THEN
        WRITE('main pkt generated', seq, start, finish, TIME);
      IF CARDINAL(nodebuf(start) EQL 1 THEN
        ACTIVATE pktserver(start);
      IF seq EQL nopkts//2 THEN ACTIVATE thruputlog;
      IF seq EQL (k*nopkts//10) THEN ACTIVATE intvldatalog;
      HOLD(negexp(rate,t);
      END;                                                          300
    ACTIVATE thruputlog;
    HOLD(1);
    ACTIVATE thruputlog;
    report;
    END of simula block;
  END of rate loop;
```

```
END of program
```