Spring 1990

Token bus LAN performance : modeling and simulation

Ramesh Kurnool
New Jersey Institute of Technology

TOKEN BUS LAN PERFORMANCE: MODELING AND SIMULATION

by

RAMESH KURNOOL

Thesis submitted to the Faculty of the Graduate school of the

New Jersey Institute Of Technology.

in partial fulfillment of the requirement for the degree of

Master of Science in Electrical Engineering.

January 1990

APPROVAL SHEET

Title of Thesis:	Token Bus LAN Performance: Modeling and Simulation.
Name of the Candidate:	Ramesh Kurnool Master of Science in Electrical Engineering, 1990.

Thesis Approved :

Dr. Anthony Robbi Date
Associate Professor
Electrical Engineering

Dr. C. N. Manikopoulos Date
Associate Professor
Electrical Engineering

Dr. Irving Wang Date
Associate Professor
Electrical Engineering

VITA

Name : Ramesh Kurnool

Degree and Date to be Conferred : M.S.E.E., 1990.

Secondary Education : Government Junior College, Kurnool, A.P., India.

Collegiate Institutions Attended :	Date	Degree
New Jersey Institute of Technology.	09/87-12/89	M.S.E.E
College of Engg., J.N.T.University.	08/82-11/86	B.E.

Major : Electrical Engineering.

Minor : Computer Systems.

ABSTRACT

Title of Thesis: Token Bus LAN Performance: Modeling and Simulation

Ramesh Kurnool, Master of Science, 1990

Thesis directed by: Professor Anthony Robbi

A simulation model based on CSIM, a process oriented simulated language, to analyze the performance of the Token Bus protocol is developed. Performance measures such as throughput, average delay and maximum delay per packet are presented. System performance is analyzed for different loads, number of stations, network lengths, different physical and logical distribution of the stations with packet length as a parameter. Previous studies were based on the delay-throughput analysis with no discussion on the effect of variation of the logical and physical distribution of the stations on the performance of the model which is done in the present thesis. The load is offered to the network in the form of a stream of data packets with uniformly distributed inter-arrival times. A comparison of the Token Bus model with that of a CSMA/CD model shows that the physical distribution of the stations has a minimum effect on the performance of the model in the case of the Token Bus model but has a considerable effect on that of the CSMA/CD model.

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to Dr. Anthony Robbi for his valuable guidance, inspiration and encouragement during the entire course of this thesis. I am deeply indebted to Mrs. Perkins, Miss. Sandra Huffman, Miss. Susana Saldarriaga and Mr. Bill Bowman at Document Preparation Centre for all their willful help and support. Furthermore, I would like to acknowledge the Computer Aided Engineering Laboratory and the staff for providing the resources vital for the thesis work. Last, but certainly not the least I am indebted to my room partners Mr. Pratap Pasupuleti and Mr.Nanda Kanuri and my parents for their constant moral support, valuable guidance and encouragement.

TABLE OF CONTENTS

CHAPTER I
INTRODUCTION

The design, installation, and operation of computer networks is vital to the functioning of modern computerized organizations. Over the past decade, complex and diverse networks have been established, trying together mainframes, minicomputers, personal computers, terminals, and other devices, such as communications controllers and cluster controllers [1].

Many of today's networks use public telecommunications facilities to provide users with access to the processing capabilities and data storage facilities associated with the mainframes and to permit fast interchange of information among the users of the network. As the cost of micro electronic devices has dropped, the intelligence in the various devices that are attached to the network has increased. Intelligent terminals, minicomputers, personal computers, and other programmable devices are all part of these large networks. These networks are called wide area networks (WANS).

In parallel with the growth of wide area networks, there has been another area of expansion in the use of computing facilities. Personal computers have spread rapidly and widely throughout organizations. As the use of personal computers has grown, so has grown a need for these personal computers to communicate-with each other and with the larger, centralized data processing facilities of the organization. In order to make this feasible a type of networking technology known as local area network (LAN) has been developed.

A local area network is a communications network that provides interconnection of a variety of devices within a small area. These data communicating devices are often called as Nodes or Stations [5]. At the turn of this decade, the installed base worldwide, should be in excess of two million LANs. It is therefore becoming more and more important for network designers to evaluate the ability of LANs that use various communication protocols to handle the desired message loads, number of stations, and desired length of the network. It is further important to have a general scheme for modeling and simulation of these protocols which can be used to assess the performance of new protocols.

A. Local Area Network Considerations:

The present trend towards distributed data processing, coupled with the increased emphasis on office automation and the explosive growth rate in the use of automated office terminals, displays, and computers, has created a demand for more flexibility in terminal installation and interconnection systems[1]. Personal workstations are emerging as a means for increasing overall productivity by allowing the individual office workers,both clerical and professional, to share access to host systems, common data bases, peripheral printers, and remote computer networks. The workstation itself may range anywhere from a low cost keyboard/display device to a small computer[1].

The industry response to this need has resulted in several proposals and offerings of communication systems specifically designed for localized data transfer. These LANs are intended to provide a transport mechanism for digital encoded data and text, as well as for non-coded data(facsimile), voice, and fullmotion video. A LAN can offer a solution that addresses these present and future needs of industry when based on a communications architecture and physical wiring strategy that allows for growth and migration into the 21st century.

A LAN can be defined then as an information transport system for data transfer among office system terminals, clustered controllers, or host systems, via an interconnecting medium, within the bounds of a single office building, building complex, or campus. The geographical constrains eliminate the need to use common-carrier facilities, thus increasing the data transfer capacity of the LAN by allowing economical data transmission rates of many million bits per second. Today's network transmission technology permits the transfer of large blocks of data at these rates with simple error management procedures and control protocols. These transmission rates also permit a large number of data terminals to share the common physical interconnection link with minimum performance degradation resulting from resource contention.

B. Standards for LANs

The IEEE's 802 LAN standard is the key standard for local area networks. This project has a relationship to the ISO OSI (Open systems Interconnection) reference model as shown in Fig 1.1. The architectural model is a set of interfaces. The lower portion of the ISO OSI model has been sub structured by the 802 project [2] as shown in Fig 1.2. In particular the functions performed by the Data Link layer have been distributed over two sublayers ; a “Logical Link Control(LLC),” an upper sublayer, and “Medium Access Control(MAC)," the lower sublayer. The LLC layer serves to generate and interpret the link control commands while the complementary, lower layer, the

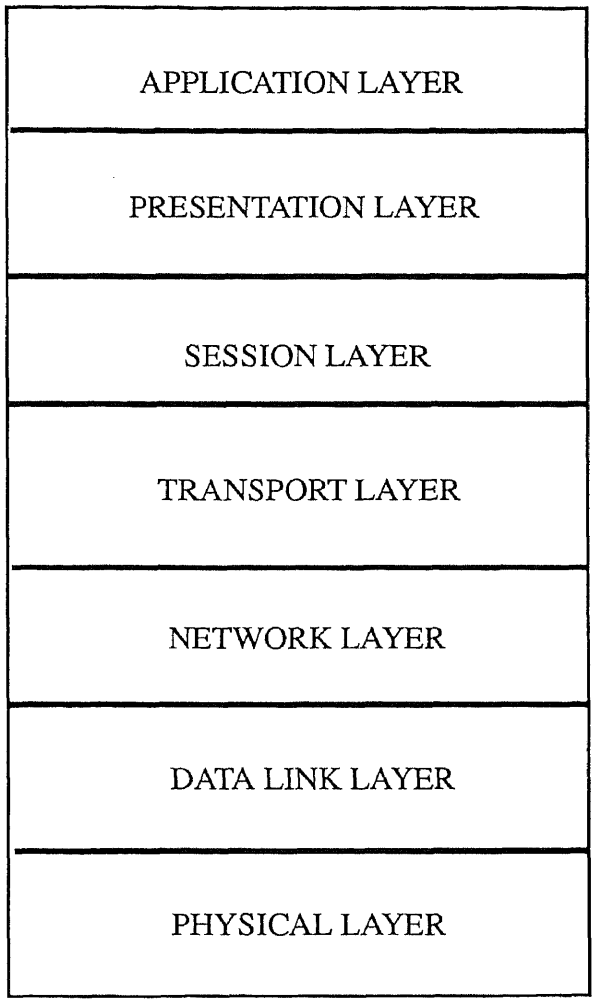


Fig 1.1
ISO OSI Model

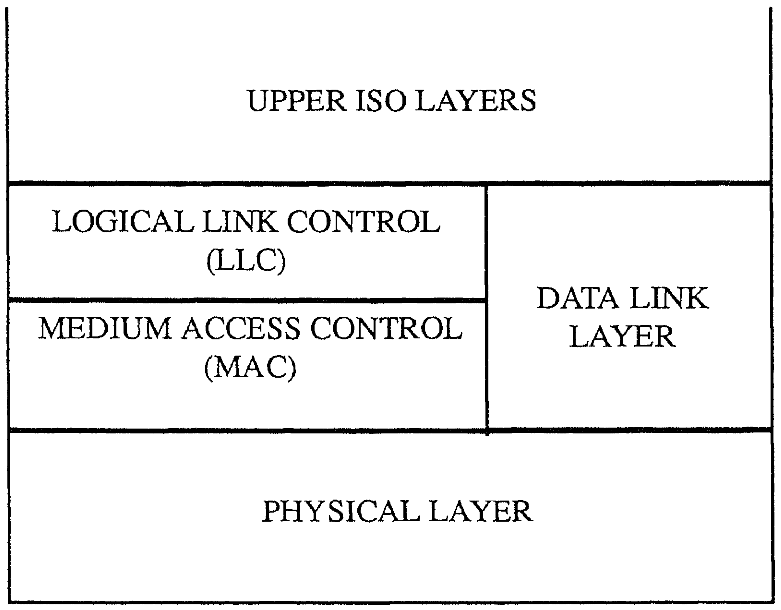


Fig 1.2
IEEE 802 Project Sublayer Model

MAC serves to frame the data units and acquire the right to access the medium. Above the bottom two layers, the five layers specified by the ISO model should operate independently.

Within this structure, the 802 project has defined four standards; a LLC standard and three standards for medium access control, together with their associated physical layers for use with the LLC standard. Three such standards are:

(1) Carrier sense multiple access with collision detection.(CSMA/CD).

(2) Token-passing bus access.

(3) Token ring access.

The "Token-Passing Bus" access method, the subject of this thesis, is one of the medium access control techniques which gives the right to the stations connected to the medium to transmit. Like CSMA/CD, the token passing bus access method communicates by broadcasting the signals generated at each station to all stations attached to the medium . The medium serves as a common bus through which the attached stations can transmit. Although CSMA/CD is widely used in offices, during the development of 802 standard people from the General Motors and other companies interested in factory automation had serious reservations about it. Due to the probabilistic behavior of the MAC protocol, there is a slight chance that a station might have to wait arbitrarily long to send a frame (i.e, the worst case is unbounded).

In the token bus method there are 'n' stations and each station takes 'T' seconds to send a frame. So, no frame will ever have to wait more than nT seconds to get a chance to be transmitted [5]. The factory automation people in

802 liked the conceptual idea of a ring, but did not like its physical implementation because a break in the ring cable will bring the whole network down. Further they noted that a ring is a poor fit to the linear topology of most assembly lines. As a result, a new standard was developed having the robustness of the cable used by the CSMA/CD method 802.3, but the known worst case behavior of a ring.

B.1. Discussion of the Physical layer

For the physical layer the token bus uses a broadband coaxial cable used for cable television. Three different analog modulation schemes are permitted in the token bus method :

(1) Phase continuous Frequency Shift Keying.

(2) Phase coherent Frequency Shift Keying.

(3) Multilevel duobinary Amplitude modulated Phase Shift Keying.

Data rates of 5 and 10 Mbps are possible using the phase coherent frequency shift keying or the multilevel duobinary AM/PSK.

Two important aspects of the network size are overall length of the cable, and total number of drops. In typical situations phase-continuous-FSK type of analog modulation is used. In such practical networks the cable length is usually limited by cable attenuation. The maximum network lengths achieved with commercially-available cables vary from 1280 meters to 7600 meters [2]. Although the Physical layer specification includes the use of repeaters to extend the trunk cable beyond the permitted length, this has rarely been needed in practice, and such use may compromise the simplicity otherwise offered b y the phase-continuous-FSK type of service. The number of drops in the above networks has typically ranged from 2 to approximately 30. This specification does not constrain explicitly the maximum number of drops, nor is that number explicitly defined cable parameters.

The preferred network topology is a long unbranched trunk requiring a single trunk cable to be routed to every station site in turn. This specification does cover the use of active, regenerative repeaters for branching. The data is transferred between the stations in the form of packets. Each packet is a collection of a finite number of bytes. (for ex., 512, 128 etc.,). The efficiency of the network is largely dependent on the size of the packets being sent and is high when large packets are used. This is due to the larger utilization of the bus by the data packets at larger packet lengths.

An IEEE Standards Committee has published its ANSI/IEEE std 802.4 Token-Passing Bus Access Method in 1985. It has seen widespread implementation throughout industry.

C. Performance of a network

The performance of a LAN, measured by the throughput achieved and the delay suffered by the packets, is a function of several parameters of the protocol and the network. These parameters include the number of stations, network topology, network length, packet length, and the distribution of the stations [12]. The performance of a netowork is very much influenced by the length of the network, which is the distance between the two farthest stations. In the present Token Bus model as the stations form a logical ring, the performance of the model is observed for various physical and logical distribution of the stations on the bus.

Simulation studies that have been done on the Token Bus performance so far were concentrated on the delay-throughput relationship and the effect of the offered load on the performance of the model [3][11]. Some studies were concentrated on the acknowledgement and priority mechanisms in Token Bus networks [9]. Some other studies were based on the queueing analysis of the data packets in the buffers of the stations. All the above mentioned studies have assumed one particular case of the physical and logical distribution of the stations on the bus. No effort has been made so far to test the effect of variation of the physical and logical distribution of the stations on the performance of the model. Further in most of the above mentioned simulation studies every station is modeled by a pair of processes. The first being able to send and receive data packets using the bus and the second being able to act as a traffic generator[10]. Further a station queues the data packets for transmission in the output buffer until it receives the token.

The Token Bus model developed in this thesis considers the traffic generating conditions which are different from those assumed in the above mentioned studies. The traffic is offered to the model by a traffic generator in the form of a stream of data packets. Each data packet is assigned a source and a destination station at random [12]. The source can not be same as the destination station. Every station has a buffer with a capacity of storing exactly one data packet. If all the stations are found busy i.e., their buffers are full, the traffic generator waits for a random time till one of the stations is free to accept a data packet. In this process of waiting the queuing delay of the data packets at the traffic generator is not considered.

In the present model in addition to the delay-throughput analysis different physical and the logical distribution of the stations have been considered. The model assumes two different types of physical distribution namely:

(1) Uniform

(2) Clustered form

In the uniform distribution the stations are spaced uniformly along the network length, whereas in the clustered form the stations are divided into a number of equal or unequal sized clusters.

Two cases of logical distribution are considered in this model. The first being the best case in which the physical and logical neighbors coincide each other and the worst case in which the logical neighbors are separated by a large distance. The effect variation of the physical and the logical distribution of the stations on the performance of the model is observed in all the experiments the details of which are explained later in Chapter IV. The details of the physical and logical distribution of the stations are given in Chapter.III.

CSIM, a process oriented simulation language which is implemented as a superset of the C programming language is used as a simulation tool. CSIM has a number of features which ease the task of building simulation models. They include modeling systems resources, message passing, data collection, and debugging [4].

CSIM is copyrighted by Microelectronics and Computer Technology Corporation, 1985

CHAPTER II
PROTOCOL AND MODEL DESCRIPTION

This chapter presents the description of the model on which the protocol is tested and an overview of description of the protocol that is tested. Further,this chapter also defines the parameters used in the simulation of the Token-passing bus protocol.

A. Protocol Description.

The IEEE 802 standards include two access control methods based on the token passing technique. One of these methods is used on a network with bus topology and the other on a network with ring topology. The token passing standard employing the bus topology, known as token bus, is defined in IEEE standard 802.4. This is the basis of the General Motors MAP architecture that is used extensively in factory automation [6]. With the token bus approach, control of the transmission medium is determined by possession of a token, which is passed from station to a station in a logical ring.

A.1. Overview of Token bus method:

The right to access the physical medium is given by the token. The token is a control frame that gives momentary control over the medium. The token is passed from one station to another thereby forming a logical ring. Normal steady state operation consists of two phases [2]; namely:

(1) Data transfer phase.

(2) Token transfer phase.

In the first phase the station which is the token holder transfers data to the bus. After transferring the data, the token holder transfers the token to its successor in its token transfer phase thereby relinquishing its control over the bus and the process repeats.

The maintenance functions of the logical ring so formed within the stations provide for the following:

(1) Ring initialization.

(2) Lost token recovery.

(3) New stations addition to the ring.

(4) General housekeeping of the logical ring.

The physical ordering of the stations on the bus is independent of the logical ordering [2], although it may be related to it as shown in the example of Fig 2.1. The token medium access method is always sequential in a logical sense. That is, during normal, steady state operation, the right to access the medium passes from station to station. The medium access control(MAC) sublayer provides sequential access to the shared bus medium in a logically circular fashion. This MAC sublayer determines when a station has the right to access the shared medium by recognizing and accepting the token from its predecessor station.

A.2. MAC INTERNAL STRUCTURE.

The MAC layer performs several functions which are loosely coupled.

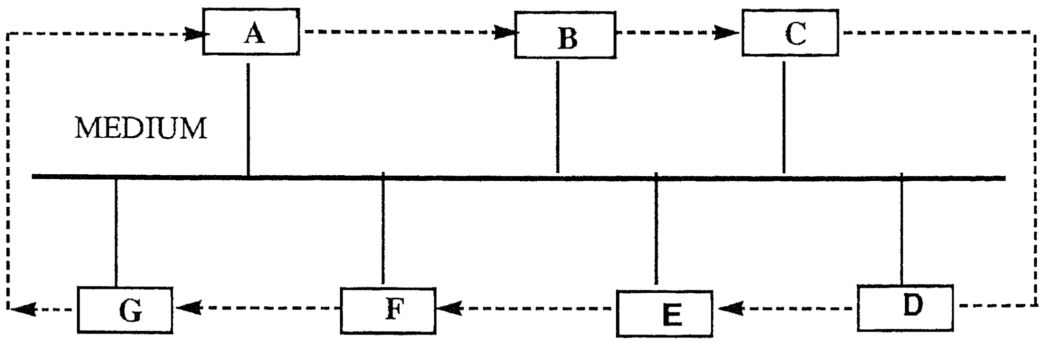


Fig 2.1
Logical Ring on Physical Bus

The descriptions and specifications of the MAC layer in this standard are organized in terms of one of the several possible partitioning of these functions. The partitioning of the MAC layer is illustrated in the Fig 2.2, which shows five asynchronous logical "machines", each of which handles some of the MAC functions. These partitions are as follows:

(1) Interface Machine(IFM)

(2) Access Control Machine(ACM)

(3) Receive Machine(RCM)

(4) Transmit Machine(TxM)

(5) Regenerative Repeater Machine(RRM)

The purpose of each machine is explained below:

A.2.1. Interface Machine: This machine acts as a buffer and interface between the LLC and MAC layers. It interprets all incoming data and other service primitives and generates appropriate outgoing service primitives. Further it handles queuing of service requests. It also performs the address recognition function on received LLC frames, accepting only those addressed to its station.

A.2.2. Access Control Machine: This machine cooperates with the ACMs of its fellow stations in handling the token to control the transmission access to the shared bus. The ACM is responsible for the following:

i. Initialization and maintenance of the ring.

ii. Taking care of newly admitted stations.

iii. Detection and recovery from faults and failures in the network.

A.2.3. Receive Machine: This machine accepts inputs from the physical layer, assembles them into frames which it validates and passes them to the IFM and

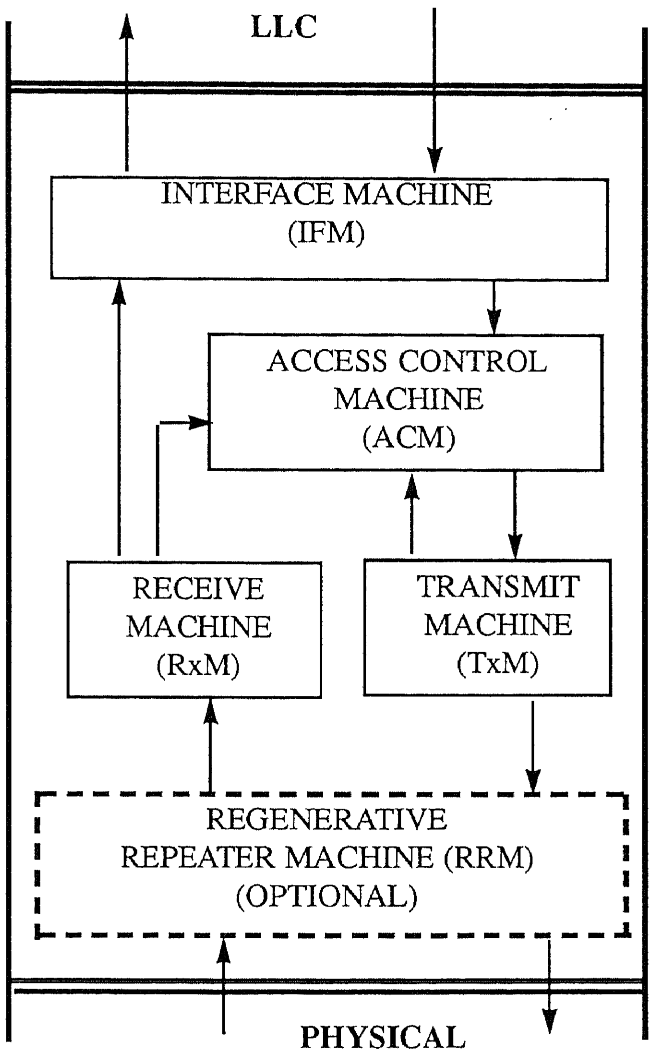


Fig 2.2
MAC Layer Functional Partitioning

ACM. The RxM accomplishes this by recognizing the frame start and end delimiters, checking the frame check sequence and validating the frame's structure. RxM is also responsible for reception of noise burst and quiet conditions. This noise burst condition is prevalent when the station is listening to the bus for some response.

A.2.4. Transmit Machine: This machine generally accepts a data frame from the ACM and transmits it as a sequence of atomic symbols in a proper format to the physical layer. The TxM builds a MAC protocol-data-unit by prefacing each frame with the required preamble and starting delimiter, and appending the Frame control sequence and end delimiter.

A.2.5. Regenerative Repeater Machine: This machine is an optional MAC component present only in special "Repeater stations", e.g., in a broadband or a head-end demodulator. In such stations the RRM repeats the incoming atomic symbol stream, from the physical layer, back to the physical layer for retransmission.

Of all these five machines the ACM is both the most critical and the most complex. It is the key control mechanism for the token-bus method. The IFM and RxM participate heavily in the operation of the MAC layer protocol.

A.3. MAC Definitions: Critical MAC parameters which are constrained by this specification are defined as follows:

A.3.1. MAC-symbols: The smallest unit of information exchanged between two MAC sublayer entities [2]. The six MAC- symbols are defined in Table 2.1.

NAME	ABBREVIATION
zero	0
one	1
non_data	N
pad_idle	P
silence	S
bad_signal	B

Table 2.1

MAC symbols.

Conventional binary 0 and 1 data bits are sent and received as zero and one MAC-symbols respectively.

A.3.2. MAC_symbol_time: The time required to send a single MAC_symbol, identical for all symbols. This is the inverse of the LAN data rate as shown in Table 2.2.

Nominal data rate	Nominal MAC_symbol_time
1 Mb/s	1.0 micro second
5 Mb/s	0.2 micro seconds
10 Mb/s	0.1 micro seconds

Table 2.2

MAC symbol times.

A.3.3. Octet_time: Corresponds to the time interval required to transmit eight MAC-symbols.

A.4. Elements of MAC Sublayer Operation:

The MAC protocol as developed is intended to be robust, in sense that it should tolerate and survive multiple concurrent errors [3]. These errors may be caused by communication errors or station failures. These errors may include multiple tokens, lost token, token-pass failure, Deaf station(station with inoperative receiver) and duplicate station address . The present model under study is assumed to be free of errors.

A.4.1. Basic Operation: Steady state operation (the network condition where a logical ring has been established and no error conditions are present) simply requires the sending of the token to a specific successor station as each station is finished transmitting.

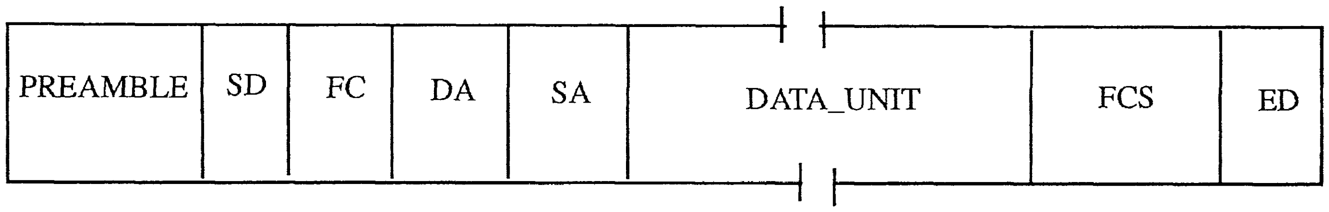
All the stations are connected to a common medium which is a bus made of a broadband coaxial cable. Physically, the token bus is a linear or tree-shaped cable onto which the stations are attached. Logically, stations are organized into a ring with each station knowing the address of the station to its “left” (PS) and “right” (NS) apart from knowing its own address (TS). When the logical ring is initialized, the highest numbered station holds the token and may send the first data frame. After it is done with transmitting, it passes the permission to its immediate neighbor by sending the neighbor a special control frame called a token. The token propagates around the logical ring, with only the token holder being permitted to transmit frames. Since only one station is allowed to transmit data frames at a time collisions will not occur.

An important point to realize is that since the cable is inherently a broadcast medium, each station receives every frame, discarding those not addressed to it. When a station passes the token, it sends a token frame specifically addressed to its logical neighbor in the ring, irrespective of where the station is physically located on the cable. It is also important to note that when stations are first powered on, they will not be in the ring, so the MAC protocol has provisions for adding to, and deleting stations from the ring.

A.5. Frame Format of IEEE 802.4 Std:

The token bus frame format is shown in the Fig.2.3. The various components of the frame are described next.

A.5.1. Preamble: The preamble pattern precedes every transmitted frame. Preamble is sent by a MAC layer as an appropriate number of pad_idle symbols. This will be decoded by the receiver modem as arbitrary data symbols that occur outside the frame delimiters. Preamble is primarily used by the receiving modem to acquire signal level and phase lock by using a known pattern agreed upon by the all the stations. A secondary purpose for the preamble is to guarantee a minimum end delimiter(ED) to start delimiter(SD) time period to allow stations to process the frame previously received. The minimum amount of preamble transmitted is dependent on the data rate of the medium and the modulation scheme implemented. The standard requires that the duration of the preamble shall be at least 2 micro seconds, regardless of the data rate [2].



PREAMBLE	=	pattern sent to set receiver's modem clock level (1 or more octets)
SD	=	start delimiter (1 or more octets)
FC	=	frame control (1 octet)
DA	=	Destination address (2 octets)
SA	=	source address (2 octets)
DATA_UNIT	=	data bits (0 or more octets)
FCS	=	frame check sequence (4 octets)
ED	=	end delimeter (1 octet)

Fig 2.3
Frame format of token bus

A.5.2. Start Delimiter(SD): The frame structure requires a start delimiter, which begins the frame. The start delimiter consists of signalling patterns that are always distinguishable from data.

A.5.3. Frame Control Field: The frame octet(FC) determines what class of frame is being sent. This field is of one octet. Various types of classes include MAC control, LLC data, Station management data and Special purpose data(reserved for future use).

A.5.4. Destination Address Field(DA): The destination address identifies the station to which the frame is destined. This may be 2 or 6 octets depending on the number of bits used for addressing. Addresses shall be either 16 bits or 48 bits in length. All addresses on a given LAN shall be of the same length.

A.5.5. Source Address Field(SA): The source address identifies the station originating the frame and has the same format and length as the destination address in a given frame.

A.5.6. MAC Data Unit Field: Depending on the bit pattern specified in the frame's frame control octet, the MAC data unit field can contain a LLC protocol data unit, which is used to exchange LLC information between LLC entities or a MAC management data frame which is used to exchange management information between MAC management entities or a value specific to one of the MAC control frames.

A.5.7. Frame Check Sequence (FCS) field: The FCS is a 32 bit frame checking sequence.

A.5.8. End Delimiter(ED): The frame structure requires an end delimiter, which ends the frame and determines the position of the frame check sequence. The data between the SD and the ED shall be an integral number of octets. All bits between the start and end delimiters are covered by the frame check sequence.

A.5.9. Token frame: The token frame is a special MAC control frame which gives a station the right to transmit. This frame has a structure shown in the Fig 2.3. This frame has the DA = the address of the station's successor in the logical ring. The token frame has a null data_unit.

A.6. The Token Bus MAC Sublayer Protocol.

When the ring is initialized stations are inserted into it in the descending order of their addresses. Token passing is also done from high to low addresses. Each time a station possesses the token, it can transmit frames for a certain amount of time called the token hold time, then it must pass the token on. If the frames are short enough, several consecutive frames may be sent. If a station has no data, it passes the token immediately upon receiving it.

The token bus defines four priority classes 0,2,4 and 6 for traffic, with 0 the lowest and 6 the highest. Each station is internally being divided into four substaions, one at each priority level. As input comes into the MAC sublayer from above, the data are checked for priority and routed to one of the four substations. Thus each substation maintains its timer with stipulated time known as Target rotation counter(TRTC). When a station receives the token it begins transmitting the frames of priority class 6. When it is done, the token is passed internally to the priority 4 substation, which may then transmit until its timer expires, at which point the token is passed internally to priority 2 substation. This process is repeated until all its frames have been sent or its timer has expired.

Again when the station does not have any frames to send it simply passes the token to its successor. The priority scheme is beyond the scope of the present model. The model assumes a single priority for all the data packets.

A.7. Logical Ring Maintenance.

From time to time stations are powered up and want to join the ring. Other stations stations turned off and leave the ring. The MAC sublayer protocol provides a detailed specification of exactly how this is done while maintaining the known worst case bound on token rotation. Below a brief sketch of this mechanism is given.

Once the ring has been established, each station's interface maintains the addresses of the predecessor and successor stations internally. Periodically, the token holder solicits bids from stations not currently in the ring that wish to join by sending one of the *solicit_successor* frames. The solicitation frame gives the sender's address and its successor's address. Stations inside that range may bid to enter(to keep the ring sorted in descending order of their addresses). After the station sends the *solicit_successor* frame, it opens a response window, which is a controlled interval of time (equal to one slot time). If no station bids to enter within a slot time(2T, as in Ethernet), the response window is closed and the token holder continues with its normal business [4]. If exactly one station bids to enter, it is inserted into the ring, and becomes the token holder's successor. If more stations bid to enter at the same time the frames will collide and be garbled, as in Ethernet. The token holder will then run an arbitration algorithm, starting with the broadcast of a *resolve_contention* frame.

The solicitation of new stations will not interfere with the guaranteed worst case for the token rotation. Each station has a timer that is reset whenever it acquires the token. When the station receives the token the old value of this timer is inspected just before the timer is reset. If it exceeds a threshold value, there has been too much traffic recently, so not bids may be solicited this time around. No guarantee is provided for how long a station may have to wait to join the ring when traffic is heavy, but in practice it should not be more than a few seconds.

Leaving the ring is easy. Consider a station A, with successor B, and predecessor C. To leave the ring, A sends C a *set_successor frame* telling it that henceforth its successor is B instead of A. Then A stops transmitting.

Due to transmission errors or hardware failures, problems can arise with the logical ring or the token. For example, if a station tries to pass the token to a station that has gone down, what happens? The solution is straight forward. After passing the token, a station listens to see if the successor either transmits a valid frame passes the token. If it does neither, the token is passed a second time. Even if that fails the station transmits a *who_follows* frame specifying the address of its successor. When the failed station's successor sees a *who_follows frame*, it responds by sending a *set_successor frame* to the the station whose successor failed, naming itself as the new successor. In this way the failed station is removed from the ring. Thus the logical ring is maintained. The model assumes a stable steady-state as far as the membership of the logical ring is concerned. Thus the performance figures computed are the upper bounds.

B. Model Description.

We are interested in evaluating the performance of token-bus network which implements the IEEE 802.4 standard and offers an unacknowledged datagram service to the data-link users. This means the datagram service offered here is not responsible for managing any acknowledgement mechanisms between remote stations. The following general assumptions have been made for the model:

The network architecture includes a number N of stations connected to a shared communication channel based on the IEEE 802.4 specifications.

Each station is modeled by a pair of processes. The first is to send and receive the packets and the second to pass the token to its successor, after it is done with the transmission of packets. Additionally each station can accept a data packet from a server which generates the simulated network load.

The token is passed by the stations residing on the medium according to a logical ring; the token gives control of the transmission channel.

Each station has a buffer to store the packets. For the present model the capacity of the buffer is limited to one packet.

The logical ring is assumed to be static i.e. no 'windows' are opened to allow for passive stations to be included in the logical ring nor shall any station shall leave the logical ring [11].

A station is active if it has a packet to transmit in its buffer and is inactive if has no packets. The station holding the token transmits exactly one packet if it is active, and immediately passes the token, if it is not.

CHAPTER III
SIMULATION DESIGN AND EXPERIMENTS

CSIM, a process oriented simulation language (implemented on top of the C programming language) is used for implementation of the model and the simulation. In this approach, processes are defined which are 'independent' programs or procedures which can execute 'in parallel' (pseudo) with other processes [3]. The processes appear to execute in parallel. In the CSIM model, the communications medium is declared as *facility* , and transmissions are declared as *events*. Simulated time is passed by using a *hold* function. The description of all these functions is given in Appendix A.

A. Description of the software model

The software model comprises 'N' stations coupled together in a logical ring. In a logical ring the stations are assigned logical positions in an ordered sequence, with the last member of the sequence followed by the first. The physical ordering of the stations, on the bus is independent of the logical ordering. In other words, the logical successor of a station may not be its immediate physical neighbor. Each station has a buffer to store a single outgoing data packet of finite length (bits). Each station knows the address of its predecessor (PS) and its successor (NS), apart from knowing its own address (TS). In each experiment, the number of stations, the network length, the station distribution, and the packet length are used as parameters.

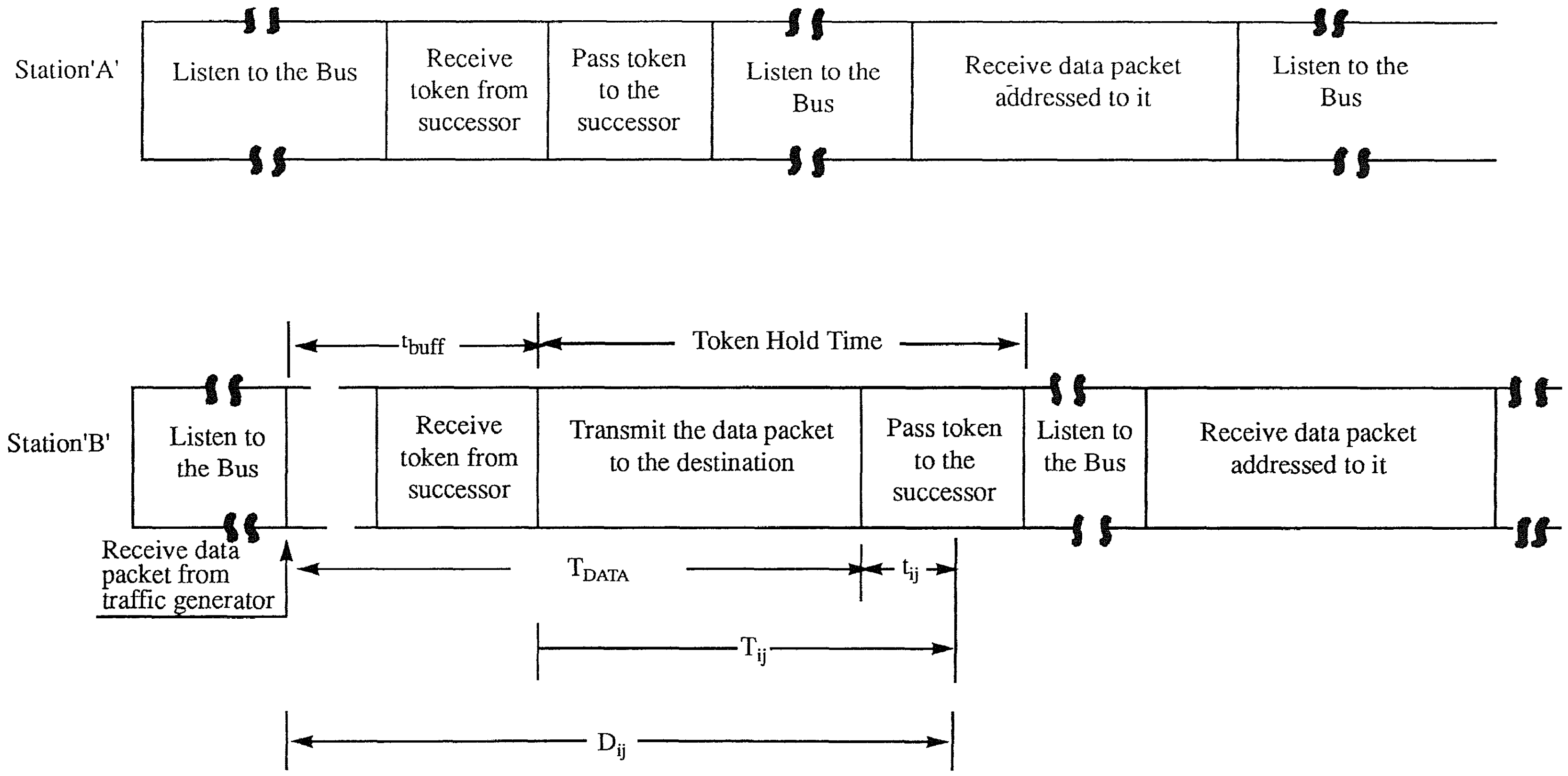
Each experiment consists of running the model for 1000 packets which are given to the overall network as loa d. These packets are offered to the model by a traffic generator which generates a stream of packets. For every packet, a source station and destination station are selected randomly [12]. Care was taken in the random selection of the source station, that it is neither in the transmission process nor its buffer is full. Obviously, the source station cannot be the destination. If the buffers of all the stations are found full, then the traffic generator waits for a random time until one of the stations is found ready to receive a packet. During this process the delay experienced by a data packet at the traffic generator before being assigned to any station is not considered. Further the time taken by traffic generator to assign a data packet to a station is assumed to be zero.

Any station can transmit a data packet if it has the token and pass the token to its successor station after transmitting the packet. A station holds the token for a certain time given by the "token hold time" if it has a data packet to transmit. This token hold time is the maximum time that a station can hold the token to transmit the data packet. In the present model the token hold time is set to a time large enough for a station to transmit one packet of variable length. The token hold time is changed according to the packet length and is same for all the stations in the model. The length of the token is same for all the experiments. Until a station receives the token, a data packet waits in the buffer. Thereby the token enables the station to transmit the packet to its destination. A station passes the token to its successor immediately if it has no packet to transmit.

A.1.Time-domain discussion This section gives the time domain features from the view point of the transmitting station. The various timings involved during the data and token transmissions are shown in the Fig 3.2. All the timings encountered during the simulation are defined in the Fig 3.1.

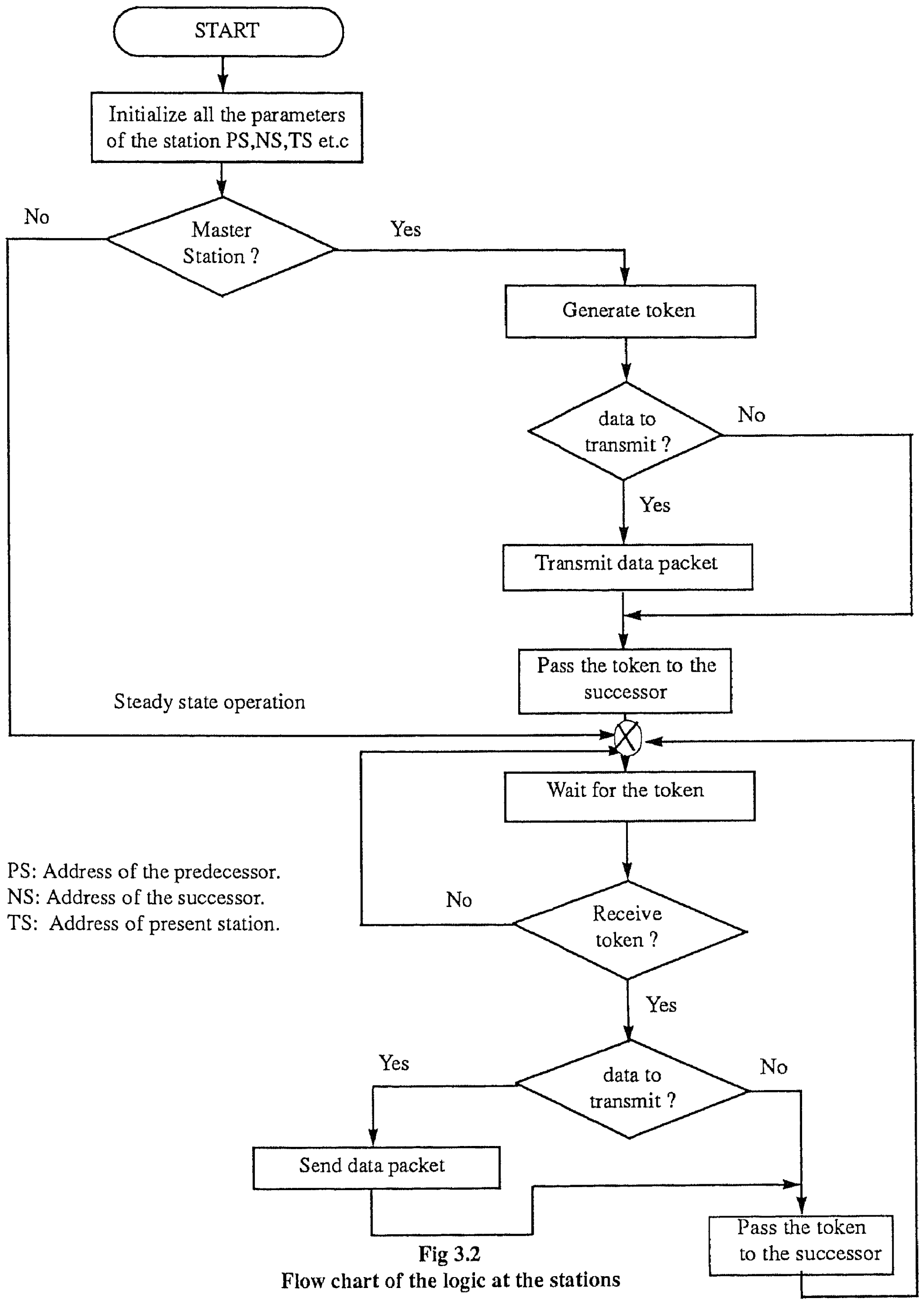
The general logic implemented at any station is explained clearly in the flow chart given in the Fig 3.2. The part of the flow chart above the circle marked in the Fig 3.2 is not simulated. The lower part of the flow chart, which is simulated explains the steady state operation of the model. To start with the highest addressed station which is the master station in the logical ring generates the token and starts transmitting the data packet if its buffer has one. Otherwise, it just passes the token to its successor and so on.

This present model does not deal with the reception side of communication and handling of transmission errors. These issues are treated at the higher levels of the protocol. The offered load to the network is controlled by the time gap between packet transmissions of the traffic generator. After every transmission initiation, the time gap for the next transmission is selected randomly within a range given by two parameters R1 and R2 [12]. R1 and R2 are two integer values. By varying the values of R1 and R2 the inter arrival rate of the packets at the stations is changed using a random function which generates an integer between R1 and R2. The values of R1 and R2, control the offered load in each experiment. The details of calculation of the offered load is given later in this chapter. For each experiment, the packet length and the token length are kept constant. IEEE Std 802.4 standards are used for the model. As per the standards channel capacities of 1,5 and 10 Mbps(mega bits per second) [4] are possible. For the present model the channel capacity is assumed to be 10 Mbps.


Station'A'
Listen to the Bus
Receive token from successor
Pass token to the successor
Listen to the Bus
Receive data packet addressed to it
Listen to the Bus
Station'B'
Listen to the Bus
Receive token from successor
Transmit the data packet to the destination
Pass token to the successor
Listen to the Bus
Receive data packet addressed to it
t_{buff}
Token Hold Time
Receive data packet from traffic generator
T_{DATA}
t_{ij}
T_{ij}
D_{ij}

T_{DATA}	:	Time elapsed from the instance a station receives a data packet from the generator until the end of transmission of the last bit of the data packet.
T_{ij}	:	Time taken to transmit the data packet from station 'i' to the destination 'j'.
t_{ij}	:	Propagation delay of the data packet from station 'i' to the station 'j'.
D_{ij}	:	Total delay experienced by the data packet.
tbuff	:	Time spent by the data packet in the buffer until the station receives the token.

Fig 3.1
Time domain picture from the view point of the transmitting station.



PS: Address of the predecessor.
NS: Address of the successor.
TS: Address of present station.

Fig 3.2
Flow chart of the logic at the stations

B. Physical Distribution of the stations

The station distribution is varied for certain experiments in the following configurations:

B.1.Uniform: 'N' Stations are spaced uniformly along the length of the network(L), with spacing between every pair of adjacent stations being L / N-1 as shown in the Fig 3.3.

```
*___*___*___*___*___*___*___*___*___*___*___*___*___*___*___*___*
```

Fig 3.3

Station distribution. Uniform spacing.

B.2.Equal-sized Clusters: The 'N' stations are divided into Nc clusters, with each cluster having N / Nc number of stations. The centers of the clusters are uniformly spaced along the network. The stations in each cluster are spaced uniformly. (Fig 3.4).

```
*_*_*_*_*__________*_*_*_*_*_____*_*_*_*_*_____________*_*_*_*_*
```

Fig 3.4

Station distribution. Equal sized clusters.

B.3.Unequal-sized clusters: The stations are divided into Nc clusters, with each cluster having a different number of stations. (fig 3.5)

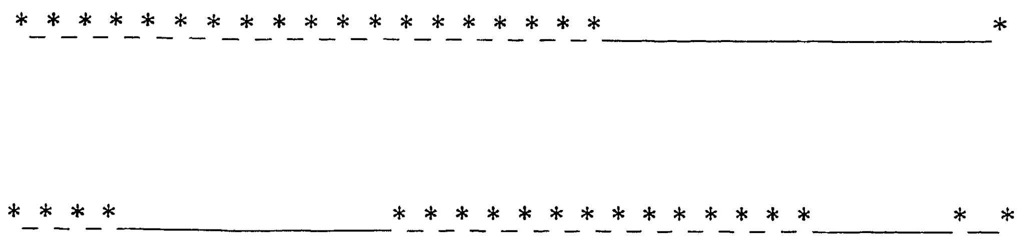


Fig 3.5

Station distribution. Unequal-sized clusters

C. Logical Distribution of the stations

In the token bus accessing method the stations form a logical ring as already discussed. The logical neighbor of any station may not be its physical neighbor. In the present model we consider two different logical distributions. In the first distribution, which is the best case, the logical neighbors are same as the physical neighbors. In the other distribution, the worst case, the logical neighbours are physically apart by a maximum possible distance.

D. Statistical Measurements

The statistical measurements, collected during the simulation runs are offered load, throughput, and average packet delay. Further the maximum delay of a data packet is also recorded.

D.1. Offered Load (G): Offered load is the total number of bits offered to the network. It includes the bits due to the control packets, tokens. The offered load (normalized) is calculated by dividing the total packet bit rate, in bits per second (total number of packets offered to the network, including the tokens) by the channel capacity 10 Mbs. The formula used for computation of the offered load in the present model is given as:

$$G = \left(\left(\text{NO\_OF-PACKETS} * \text{PACKET\_LENGTH} + \text{TOTAL\_TOKENS} * \text{TOKEN\_LENGTH} \right) / \text{sim\_time} \right) / \text{capacity} \qquad (1)$$

where NO_0F_PACKETS is the total number of data packets offered to the network.

PACKET_LENGTH is the length of the data packet in bits (512, 1024 etc)

TOTAL_TOKENS is the total number of token transmissions by the stations during the sim_time.

TOKEN_LENGTH is the length of the token frame which is assumed to be 112 bits.

sim_time is the total time simulated by the model.

capacity is the channel capacity, which is assumed to be10Mbps.

In the token bus access method when a station has no packet to transmit it has to pass the token to its neighbor. Even if there are no packets offered to the network the stations which are idle have to pass the token among themselves. Since the offered load involves the tokens, there always exists a non-zero offered load even in the absence of data packets.

D.2. Input Load (G'):

We define another load called as "input load"(G') defined as the number of data bits offered to the network per second, normalized to system capacity. The input load is given by the formula

$$G' = (\text{NO\_OF\_PACKETS} * \text{PACKET\_LENGTH} / \text{sim\_time}) / \text{capacity} \tag{2}$$

The input load does not take the token transmissions into consideration.

D.3. Throughput (S):

The throughput of the model is calculated by the formula

$$S = \left(\sum_{\substack{i=1 \\ j=1}}^{\substack{j=NP \\ i=NP}} T_{ij} \Big/ \text{sim\_time} \right) \Big/ \text{capacity} \tag{3}$$

Where T_{ij} is the time taken to transmit and propagate a data packet from a station i to a station j. Looking back to the Fig 3.2, T_{ij} includes the propagation delay t_{ij} which is the time taken to propagate the data packet from the station i to the station j. sim_time is the total simulated time and capacity is the capacity of the network (10 Mbps)

D.4. Average delay (D): The average delay is the average packet delay calculated by the formula

$$D = \sum_{\substack{i=1 \\ j=1}}^{\substack{j=NP \\ i=NP}} D_{ij} \Big/ \text{NO\_OF\_PACKETS} \tag{4}$$

where NP is the total number of packets offered to the model, D_{ij} is the time taken for a data packet to transmit successfully from the station i to the station j plus the time spent by the packet in the buffer (t_{buff} from Fig 3.2) until the station receives the token, enabling transmission. Thus D_{ij} defines the total delay for one packet.

D.5. Maximum Delay (D_{max}) : This is maximum delay experienced by a data packet. Of the 1000 data packets offered to the network as the load one data packet will experience the maximum delay than other data packets. D_{max} will be the maximum of all D_{ij}'s.

$$D_{max} = Max(D_{ij}) \tag{5}$$

Thus the simulation parameters have been explained and the results of these experiments are given in the following chapter.

CHAPTER IV
SIMULATION RESULTS

A. Overview of the experiments:

The performance of the LAN is modeled with various parameters like packet length, station distribution, logical distribution of the stations, number of stations and offered load which is controlled by the values of the packet arrival rates R1 and R2. Each experiment consists of running the model for 1000 packets. In each experiment the load offered to the network is varied by varying the arrival rate of the packets at the stations by varying the values of R1 and R2. In some experiments the packet length is varied for all other constant parameters. In some experiments the physical station distribution is varied from uniform to clusters for all other constant parameters. In some other experiments the number of stations is varied from 5 to 50 for all other constant parameters. Finally in some experiments the performance of the model is tested for different network lengths with all other parameters constant. In most of the experiments the network length is assumed to be of 1000 meters with 20 stations distributed uniformly. In all the experiments the token hold time is same for all the stations, time enough for a station to transmit one data packet and pass the token to its successor.

B. Discussion of the results

The results of various experiments are discussed in this section.

B.1. Throughput Versus Offered load:

Figure Fig 4.1 shows the variation of the throughput with the offered load for a packet length of 512 bytes. It is observed that at lower loads the throughput increases nearly linearly with the offered load. As the offered load becomes higher

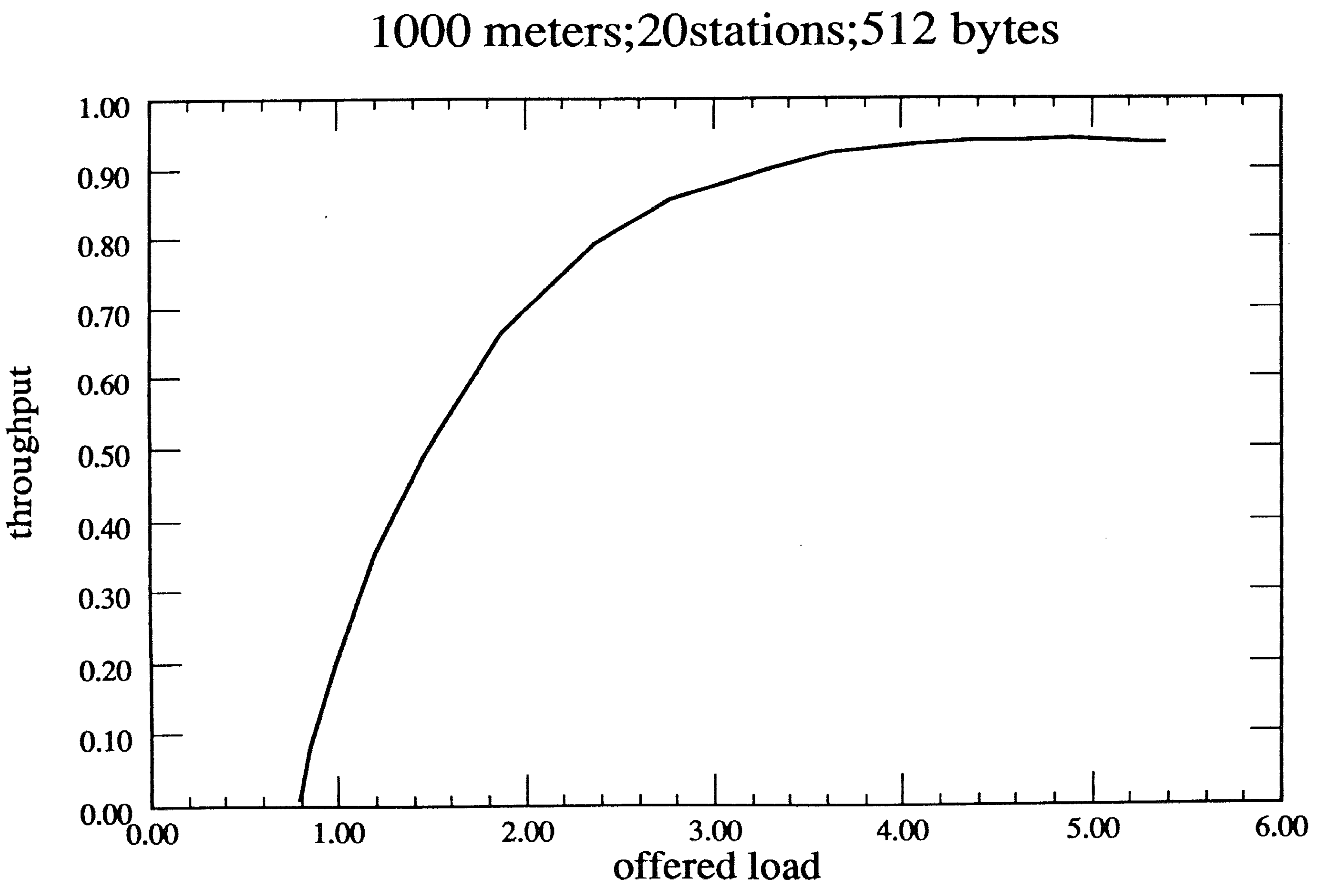

1000 meters;20stations;512 bytes
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
throughput
0.00
1.00
2.00
3.00
4.00
5.00
6.00
offered load

Fig 4.1

the throughput is stabilized at a maximum value. It can also be observed that the offered load will not be reduced to zero. This is due to the token passing between the stations even when none of the stations have data packets to transmit. This prevents the offered load from reducing to zero in the absence of data packet transmission.

The throughput of an ideal network (a network that has no overhead or token passing) increases to accommodate the load up to an offered load equal to the full capacity of the system; then remains at 1.0, which means that the network is fully utilized [3]. In the present experiment, the overhead due to the token passing always causes the performance to fall short of this ideal. Fig 4.2 shows the offered load-throughput relation for a shorter packet length. The slope of this curve is less than that of the curve with a higher packet length. This is because there is relatively more token overhead due to the token passing at smaller packet lengths. The difference of the slopes of both these curves can be better seen from Fig 4.3, where they are plotted together. It can also be observed that the maximum throughput value is higher for longer packets. This is because,the channel capacity being the same, the proportion of the number of useful data bits carried by the medium at any instance decreases as the packet length decreases.

B.2. Effect of the total length of the network on the throughput

The total length of the network is the distance between the farthest stations . The following Table 4.1 gives the values of the throughput achieved for various lengths with a packet length of 4096 bits ,token length of 112 bits for 20 stations.

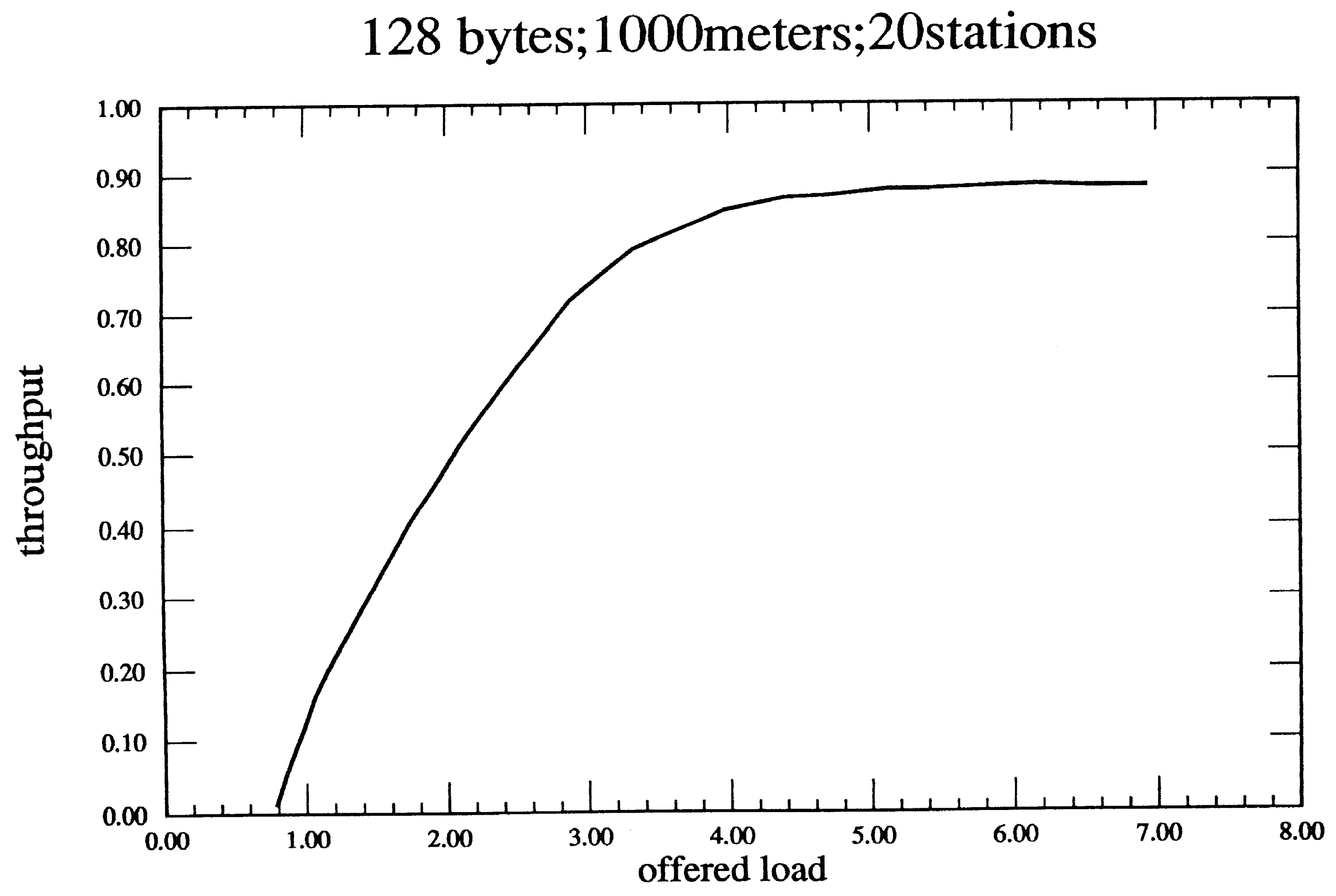

128 bytes;1000meters;20stations
throughput
offered load
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
0.00
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00

Fig 4.2

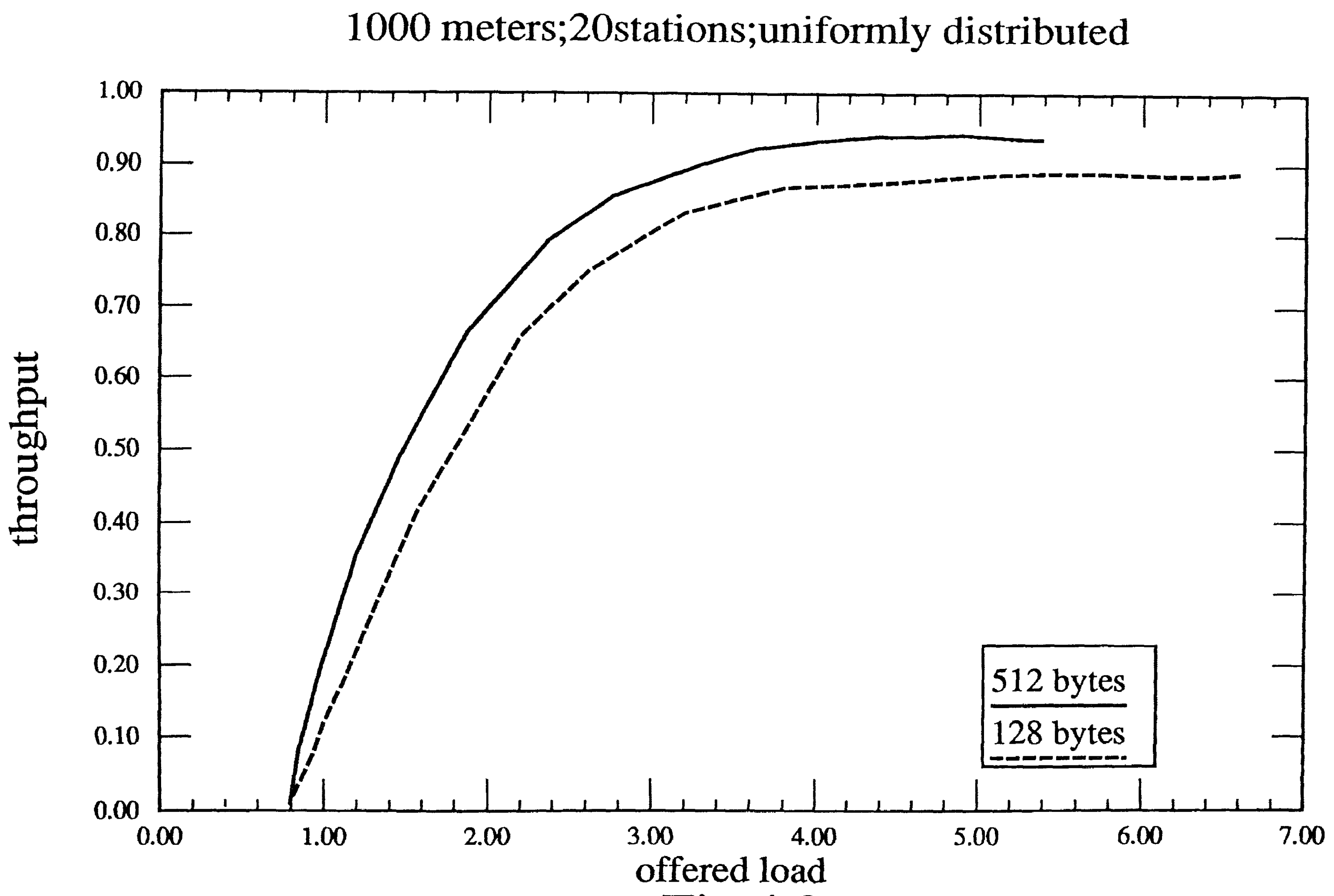

1000 meters;20stations;uniformly distributed
throughput
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
0.00
1.00
2.00
3.00
4.00
5.00
6.00
7.00
offered load
512 bytes
128 bytes

Fig 4.3

R1	R2	Throughput	Total length (meters)
1	10	0.980	100
1	10	0.974	500
1	10	0.971	1,000
1	10	0.968	2,000
1	10	0.957	4,000
1	10	0.928	8,000
1	10	0.919	10,000
100	300	0.968	100
100	300	0.963	500
100	300	0.959	1,000
100	300	0.954	2,000
100	300	0.937	4,000
100	300	0.909	8,000
100	300	0.895	10,000
300	400	0.963	100
300	400	0.959	500
300	400	0.955	1,000
300	400	0.948	2,000
300	400	0.933	4,000
300	400	0.906	8,000
300	400	0.892	10,000

Table 4.1

Throughput for different network lengths

From Table 4.1 it can be observed that for a given packet length (4096 bits) as the length of the network is increased, the throughput of the network is decreased.

As the length of the network is increased, the distance travelled by the token is increased. This increases the token delay, which is part of the time that a packet has to wait before it is transmitted by the source station to its destination. The increase in the token delay has the effect of increasing the total simulated time, which brings down the throughput.

B.3. Throughput Versus Input load:

Fig 4.4 shows the variation of the throughput with input load defined in the last chapter. It can be seen from the figure that the throughput increases linearly with the input load and stabilizes to a maximum throughput at higher loads. A striking difference between Fig 4.3 and 4.4 is that the the curve in the Fig 4.3 passes through the zero origin where as the curve in Fig 4.4 does not pass through the origin. This implies that the input load is zero when no data packets are offered to the network, where as the offered load will never be zero, owing to the token passing between the idle stations when they have no data packets to transmit. Further the offered load to the network is greater than the input load. This is due to the fact that the offered load includes not only the data packets offered to the network but also the token passing between the stations ,whereas the input load includes only the data packets.

B.4. Average Delay Versus Throughput

Fig 4.5 and Fig 4.6 show the variation of the average packet delay with the throughput for two different packet lengths. As the arrival rate of the packets at the stations increases, the offered load increases. As a result of the increased

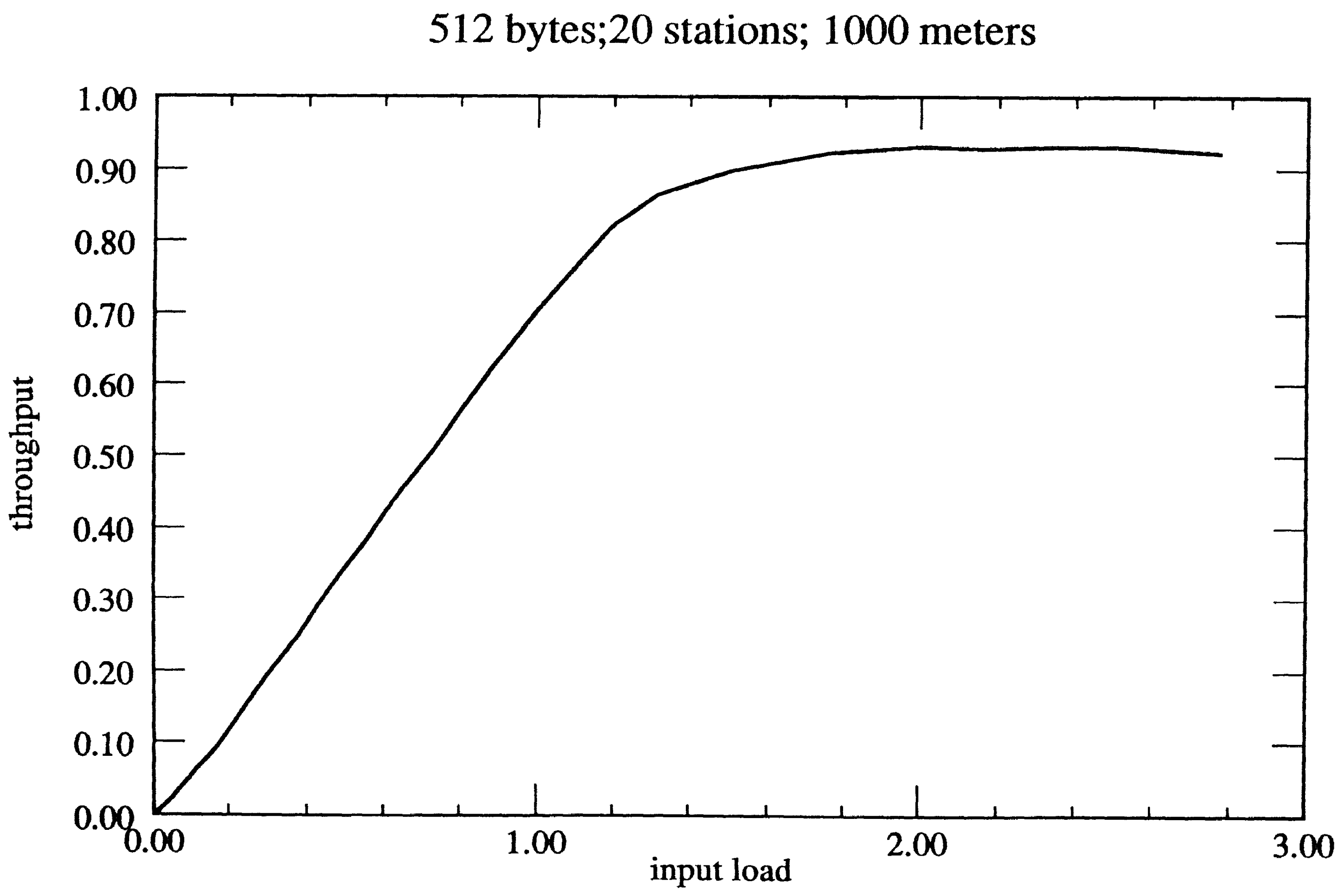

512 bytes;20 stations; 1000 meters
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
throughput
0.00
1.00
2.00
3.00
input load

Fig 4.4

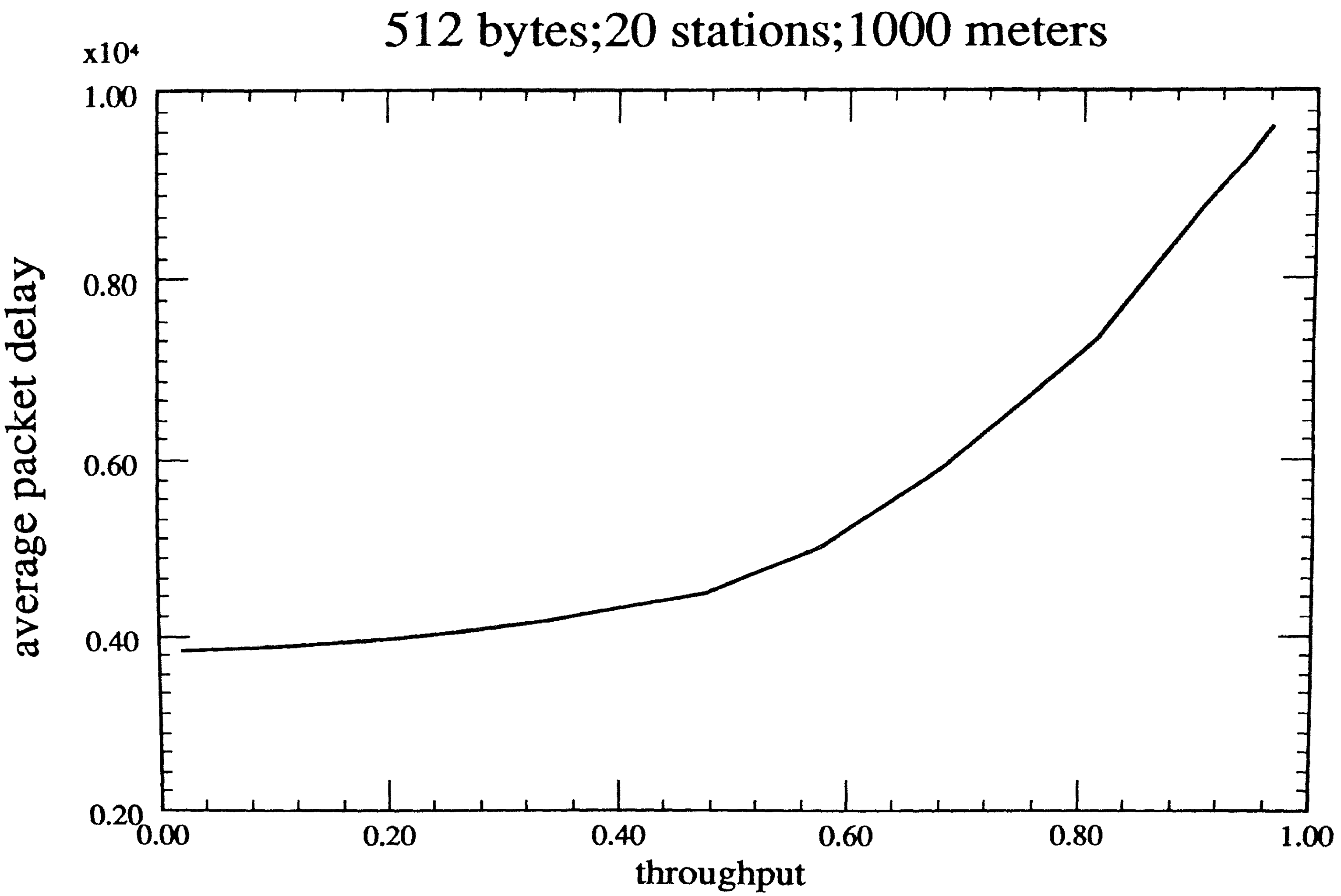

512 bytes;20 stations;1000 meters
x10^4
1.00
0.80
0.60
0.40
0.20
average packet delay
0.00
0.20
0.40
0.60
0.80
1.00
throughput

Fig 4.5

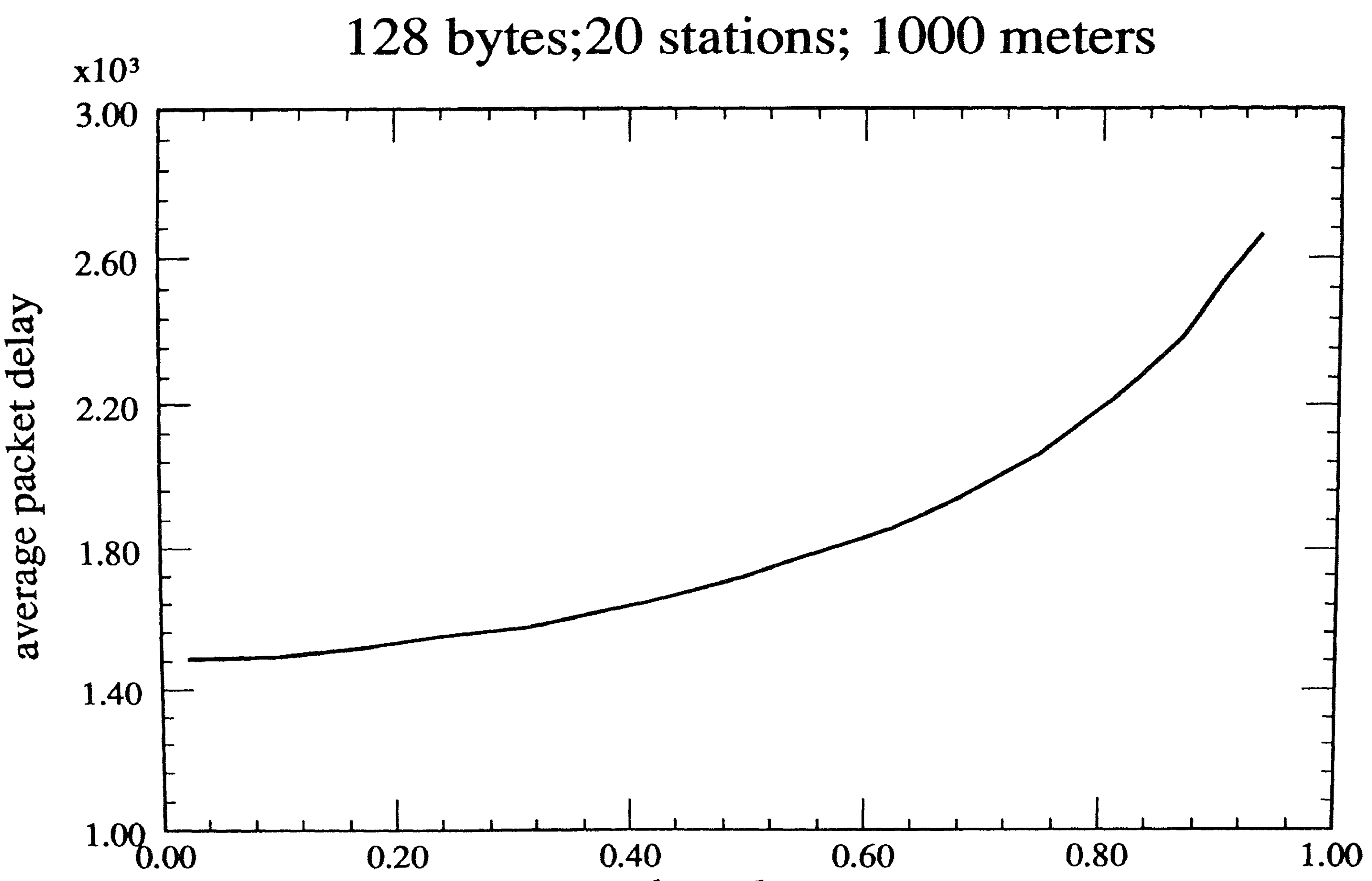

128 bytes;20 stations; 1000 meters
x10^3
3.00
2.60
2.20
1.80
1.40
1.00
average packet delay
0.00
0.20
0.40
0.60
0.80
1.00
throughput

Fig 4.6

offered load, the throughput of the model increases. This has an effect of increasing the average token rotation period. Average token rotation period is the average time taken by the token to complete one round of passing through all the stations in the logical ring. A station has to wait for the token before it transmits a packet. So, as the average token rotation period increases the average packet delay increases. It can be observed from the figures that the average packet delay increases gradually with the throughput and attains a maximum at the highest throughput. Delay-throughput analysis of two different packet lengths (in bytes) is shown in the figure Fig 4.7. The use of longer packets yields better throughput , but at the cost of increase in the time delay. The sensitivity of the model to varying packet lengths can be seen from the figure Fig 4.7. In this delay analysis the additional overhead to the data packet transmission caused by the token transmission is considered. The maximum delay experienced by a data packet is recorded for each experiment. A situation arises where the traffic generator supplies a data packet to a station that has just passed the token to its successor. In such a case the data packet has to wait in the buffer until the station receives the token in the next round. Thus the data packet experiences the maximum delay under the above mentioned conditions. The maximum delay experienced by a data packet is recorded as 8,945.55 micro seconds.

B.5. Throughput Versus Number of stations

Results of experiments with four different packet lengths (64, 128, 256 and 512 bytes) as a parameter, uniformly distributed in 1000 meters are shown in the Fig 4.8. It can be seen that as the packet length increases the maximum throughput increases. It can also be seen that as the number of stations increases for any packet length the performance of the LAN is enhanced. This is because as the number of stations is increased within the same length, the stations will be located close to one another . Thus, less time is spent in token passing and the

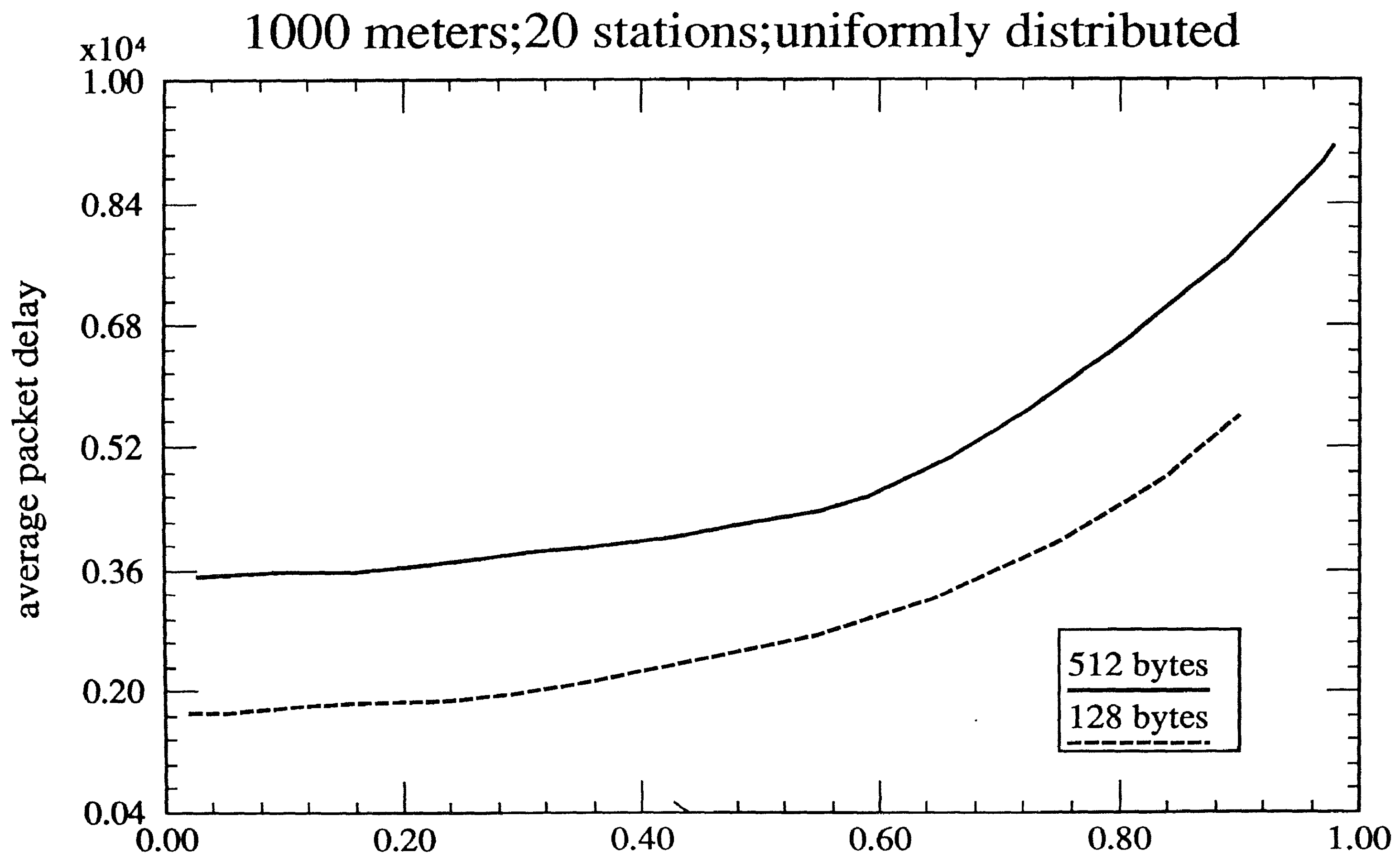
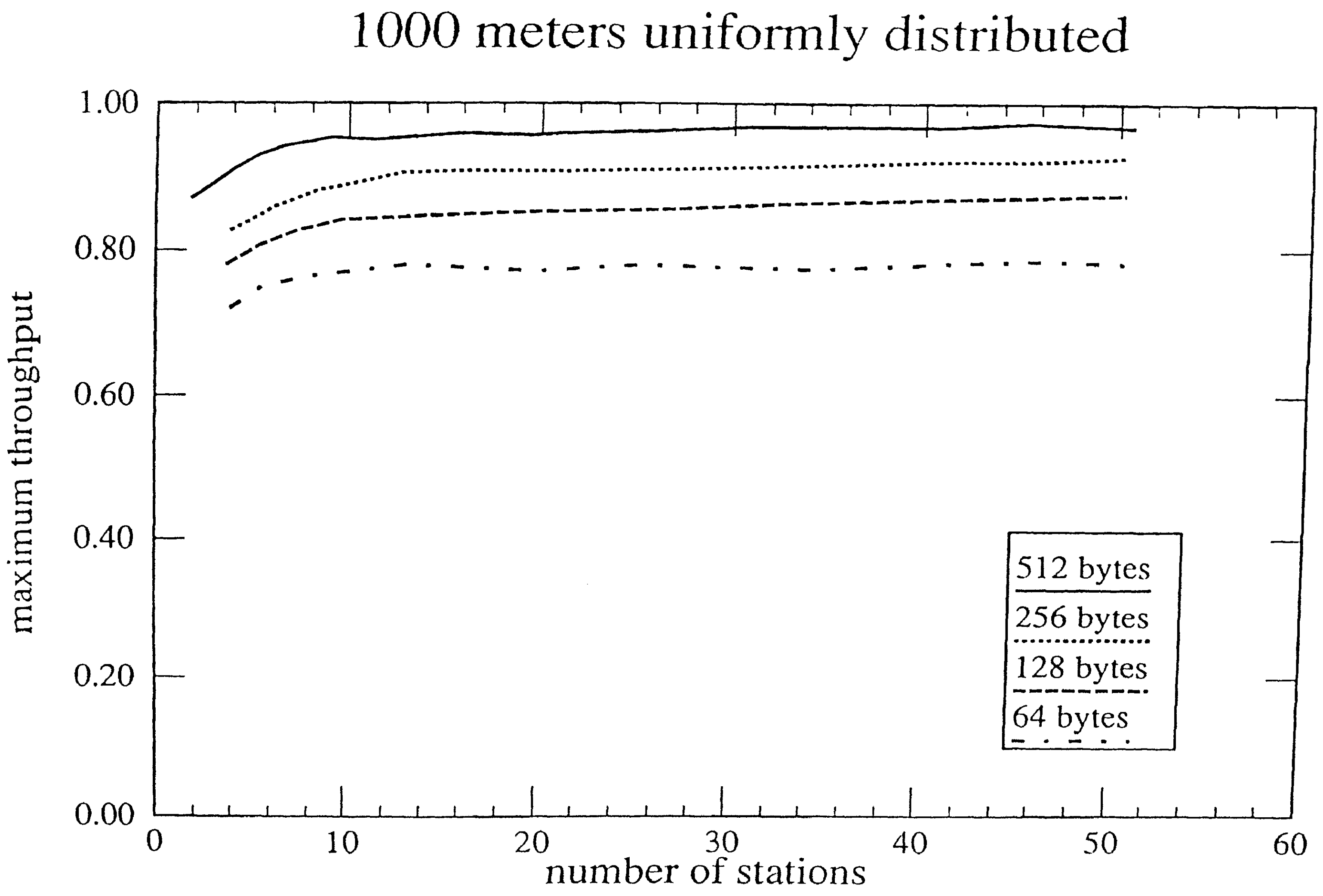

1000 meters;20 stations;uniformly distributed
x10^4
1.00
0.84
0.68
0.52
0.36
0.20
0.04
average packet delay
0.00
0.20
0.40
0.60
0.80
1.00
throughput
512 bytes
128 bytes

Fig 4.7



1000 meters uniformly distributed
maximum throughput
1.00
0.80
0.60
0.40
0.20
0.00
0
10
20
30
40
50
60
number of stations
512 bytes
256 bytes
128 bytes
64 bytes

Fig 4.8

overhead due to the token passing is decreased.

B.6. Throughput Versus Station Distribution

Results of experiments with equal sized clusters and unequal sized clusters, for the best case with different intercluster spacing are as shown in the Table 4.2, Table 4.3. The model has a 1000 meters length and a packet length of 512 bytes. All equal sized clusters are symmetrical about the physical center of the network.

Table 4.2 shows the maximum throughput achieved when the 20 stations are in equal sized clusters with intracluster spacing being 10 meters. It can be seen that there is only a slight change in the throughput for a change in the number of clusters, all else being equal

Configuration	Inter-cluster separation	Maximum throughput
2 clusters	410.0 meters	0.976
4 clusters	280.0 meters	0.973
10 clusters	100.0 meters	0.974
Uniform	55.0 meters	0.973

Table 4.2

Equal sized clusters

Table 4.3 shows the maximum throughput achieved when the 20 stations are in unequal clusters with the intercluster space being 10 meters. Even in this case there is a negligible change in the throughput for the change in the number of clusters

Configuration	Inter-cluster separation	Maximum throughput
3 clusters (10,5,5)	415.0 meters	0.977
2 clusters (15,5)	820.0 meters	0.976
2 clusters (19,1)	820.0 meters	0.975

Table 4.3

Unequal sized clusters

These results confirm that physical configuration of the workstaions has a minimal effect on the throughput. This is due to the fact that the total distances travelled by the token in all the above mentioned physical configurations is the same. Therefore the token overhead is the same.

C. Effect of logical distribution of stations on the performance of the model

As explained in the previous chapter the model is tested for two different logical distributions of the stations. The first case of distribution being the best case where the logical neighbours are same as the physical neighbours and the other case being the worst case where logical neighbours are physically apart by a maximum distance. In this experiment the stations are uniformly distributed on a length of 1000 meters. Results of this experiment are as shown in the Table 4.4.

R1	R2	Throughput (Best case)	Throughput (worst case)	Total length (meters)
1	10	0.980	0.977	100
1	10	0.974	0.971	500
1	10	0.971	0.969	1,000
1	10	0.968	0.965	2,000
1	10	0.957	0.954	4,000
1	10	0.928	0.925	8,000
1	10	0.919	0.899	10,000
100	300	0.968	0.965	100
100	300	0.964	0.961	500
100	300	0.960	0.959	1,000
100	300	0.954	0.952	2,000
100	300	0.937	0.934	4,000
100	300	0.909	0.890	8,000
100	300	0.880	0.875	10,000
300	400	0.961	0.959	100
300	400	0.957	0.954	500
300	400	0.949	0.945	1,000
300	400	0.939	0.935	2,000
300	400	0.927	0.925	4,000
300	400	0.906	0.890	8,000
300	400	0.889	0.879	10,000

Table 4.4

Comparison of throughput for the best and worst cases
(Uniform distribution)

It can be observed from the results that there is not an appreciable difference in the throughput of the LAN between the two cases. We can infer from this that the logical distribution of the stations does not have an effect on the performance of the model when the stations are distributed uniformly. This is primarily because, the token overhead is independent of the logical distribution of the stations. The token has to pass through all the stations irrespective of their locations on the medium, as such the token overhead will be the same.

On the other hand if the stations are distributed in the form of clusters as explained in the previous chapter the throughput for the worst case of the logical distribution is less than that of the best case. This is because when the stations are distributed in the form of clusters, the distance travelled by the token between the stations in the worst case is increased. This increases the token delay, which is part of the time that a packet has to wait before it gets transmitted by the source station to its destination. This increase in the token delay has the effect of increasing the total simulated time, which has the effect of decreasing the throughput. A close look at Table 4.5 shows that the throughput in the case of the worst case decreases as the inter-cluster distance is increased. This is because as the inter-cluster distance is increased the distance travelled by the token in one rotation will be increased, increasing the token delay, thereby decreasing the throughput.

Configuration	Inter-cluster separation	Throughput (best case)	Throughput (worst case)
2 clusters	810.0 meters	0.976	0.912
4 clusters	280.0 meters	0.973	0.934
10 clusters	100.0 meters	0.974	0.945

Table 4.5

Comparison of throughput for best and worst cases

(non-uniform distribution)

An implication of the results from Table 4.4 and Table 4.5 is that when the stations are uniformly distributed, the logical distribution of the stations does not have an appreciable effect on the throughput of the network. This is because, the token overhead will be the same in both the cases as the token has to pass through all the stations irrespective of their locations on the medium. But, when the stations are distributed in the form of clusters, there will be a fall in the throughput. This is due to the fact that the distance travelled by the token will be large wdhen compared to the best case, which decreases the throughput.

CHAPTER V
COMPARATIVE PERFORMANCE WITH CSMA/CD MODEL

The purpose of this chapter is to give some insight into the relative performance of the present model employing Token Bus access method with that of another model employing CSMA/CD access method.

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) media access method is a means by which two or more stations share a common bus transmission medium. It is a LWT (listen while talk) network [5]. In CSMA/CD there is no central station controlling the transmission activities of all the stations. In this access method when a station has a data packet to send, it first listens to the medium to see if anyone else is transmitting. If the medium is found busy, the station waits until the medium becomes idle. When a station detects an idle medium, it transmits a data frame. There is a small chance that just after a station begins sending, another station will become ready to send and sense the channel. If the first station's signal has not yet reached the second one, the latter will sense an idle channel and will also begin sending, resulting in a collision. If a collision occurs, the station waits a random amount of time and retransmits the data frame that is collided.

For comparison, we consider the two models with identical load generating conditions. Both the models have 20 stations in a network length of 1000 meters. The channel capacity in both the models is 10 Mbps.

Comparison of throughput

For both models the throughput is dependent on the packet length and the number of stations. For a fixed number of stations and identical traffic conditions the throughput of both models increase with the increase in the packet length. But the maximum throughput achieved in the CSMA/CD model is less than that of the token bus model. This is due the high collision rate at high loads in CSMA/CD model. In the token bus, there is no chance for the packets to collide. The performance of the token bus improves very much at higher loads. The difference in the throughput of both models can be seen from the Table 5.1

No. of stations	Packet length in bytes							
	64		128		256		512	
	TB	CS	TB	CS	TB	CS	TB	CS
2	0.77	0.55	0.86	0.62	0.91	0.72	0.95	0.77
5	0.78	0.54	0.87	0.61	0.92	0.71	0.96	0.76
10	0.78	0.50	0.87	0.60	0.93	0.68	0.96	0.75
20	0.79	0.43	0.88	0.56	0.93	0.63	0.97	0.70
25	0.80	0.41	0.89	0.54	0.94	0.68	0.98	0.68

Table 5.1

Maximum Throughput

TB: Token Bus, CS: CSMA/CD

From a close look at the results in the above table it can be seen that the throughput of a CSMA/CD model decreases as the number of stations within a given length is increased. This is due to greater likelihood of collisions. But the performance of the token bus model improves as the interstation distance decreases, because less time is spent in token passing.

The physical configuration of the stations on the bus has a marked effect on the throughput of the CSMA/CD model. As seen earlier the token bus model is quite insensitive to the change in the physical configuration of the model because the overhead due to the token passing will be the same irrespective of the physical configuration of the stations as already discussed in the previous chapter. The change of the throughput with a change in the physical configuration can be observed from the Table 5.2 and 5.3

Configuration	Average separation, m	Maximum Throughput	
		Token Bus	CSMA/CD
2 clusters	518.83	0.97	0.57
4 clusters	435.26	0.97	0.58
10 clusters	385.12	0.97	0.60
Uniform	368.41	0.97	0.61

Table 5.2

Equal sized clusters

Configuration	Average separation, m	Maximum Throughput	
		Token Bus	CSMA/CD
3 clusters(10,5,5)	455.98	0.97	0.57
2 clusters(15,5)	393.48	0.97	0.59
2 clusters(19,1)	112.75	0.97	0.69

Table 5.3

Unequal sized clusters

Comparison of average packet delay

The average packet delays for various values of the throughput can be observed from the Table 5.4. It can be seen that at lower loads the CSMA/CD method offers the shortest delay (milli seconds). This is because the stations do not have to wait for a token; they can transmit data immediately. In the token passing method the delay suffered by a packet at lower loads is higher than that of the CSMA/CD. This is because, the overhead due to token-passing is relatively severe at lower loads. But at higher loads the delay suffered by a data packet in the token bus model is very much less than the delay in CSMA/CD model. The higher delay in the CSMA/CD model is due to an increase in frequency of collisions. The variation of the average packet delay with the throughput for both the models can be seen from Fig 5.1.

Throughput	Average Delay (milli sec) (CSMA/CD)	Average Delay (milli sec) (TOKEN BUS)
0.80	80.0	6.90
0.78	60.0	6.10
0.68	1.50	5.97
0.63	0.90	5.96
0.54	0.70	5.70
0.48	0.60	5.30
0.40	0.55	4.70
0.32	0.52	4.4
0.21	0.50	4.10

Table 5.4

Average delay Versus Throughput

Comparing the offered load of the two models, it is seen that the offered load in the case of the CSMA/CD model includes the originally transmitted data packets and the retransmissions when portions of some of the data packets are lost in collision. The offered load will be zero when no packets are offered to the

1000 meters; 20 stations; uniformly distributed; 512 bytes

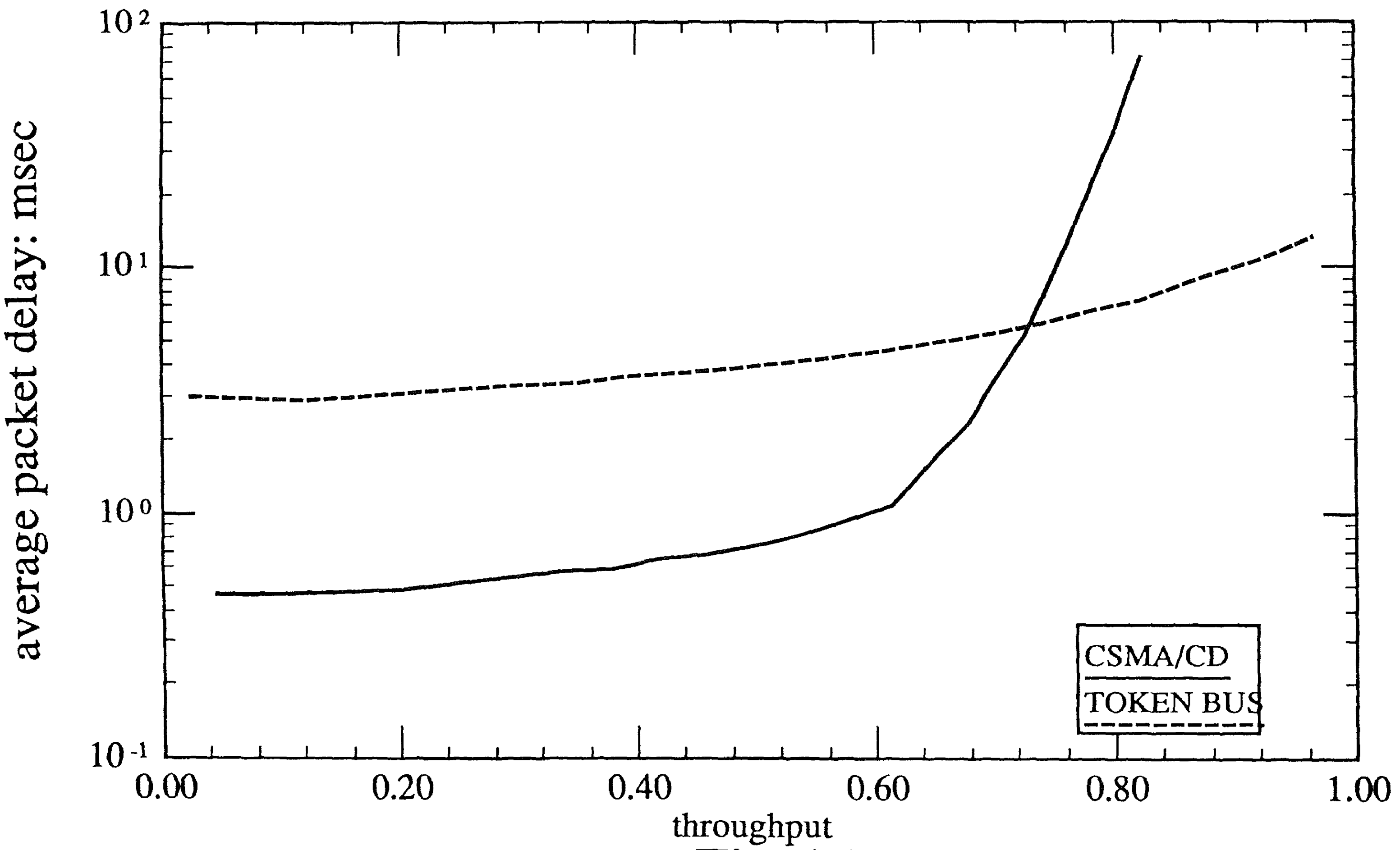


Fig 5.1

network. But in the case of the token bus model, since the access to the medium is controlled by the token, the offered load includes both the tokens and the data packets. There are no collisions or retransmissions in the case of the token bus. In the token bus model the offered load will never reduce to zero because the token will always be passing among the stations.

Finally ,the CSMA/CD method is more nondeterministic [5], which is often inappropriate for real time work. But the token bus accessing method is more deterministic than the CSMA/CD. The token passing bus method has excellent throughput and efficiency at high load.

Discussion of the MAP protocol:

A group of people working under the auspices of IEEE was set up to write official standards for the LANs. Many people were invited to join the committee and contribute to the work. Some of these people came from General Motors, which was carefully looking at LANs. GM wanted to set up a network for factories. An important application of the GM network was factory automation, in which all the robots working on the assembly lines would all be interconnected by a LAN [5]. Since the cars on the assembly lines move by at a fixed rate, whether the robots are ready or not, GM felt it essential to have a LAN in which the worst case transmission time had an upper bound that was known in advance. CSMA/CD does not have this property.

Essentially, CSMA/CD works by having all the machines listen to the cable. If the cable is idle, any machine can transmit. If two stations transmit at the same time, there is a collision, in which case they all stop, wait a random period of time, and try again later. In theory, there is no upper bound on the time a station might have to wait to send a message. GM came up with the LAN called **token bus** in which stations took turns, round-robin, thus giving a deterministic worst-case performance rather than a statistical one [5].

GM and other companies concerned with factory automation clearly saw the need to adopt specific protocols in each OSI layer. This work led to the **MAP (Manufacturing Automation Protocol)** using the token bus. MAP protocol follows very closely to the OSI model. MAP uses the token bus for the physical medium. It uses the IEEE 802.2 data link protocol LLC (Logical Link Control) in the data link layer in connectionless mode as the service available to the network layer.

CHAPTER VI

COMPARATIVE RESULTS FROM ANALYTICAL AND SIMULATION STUDIES

In the present chapter a comparison of the simulation results obtained in the previous chapters will be done with those of the analytical results.

For the purpose of analysis we consider a group of N active stations forming a logical ring. We consider one round of token passing assuming that every station gets the token only once. The number of stations transmitting in one token rotation depends upon whether or not a station has packets to transmit, which depends on the arrival rate of the packets at each station. An inactive station just passes the token to its logical neighbor.

In general the throughput of a network [10] can be defined as

$$S = F \Big/ F + Y \qquad (1)$$

where F is the time spent on the network transmitting messages and Y is the overhead associated with the message transfers. The overhead calculation depends upon the number of packets transmitted, because for each packet transfer the token and other associated overheads are also transmitted. If there are no packets waiting for transmission at any station, the token is passed without being used. For the purpose of analysis we assume that all the stations have data packets to transmit. This is the case for the maximum throughput.

We consider the best case of the token bus for our analytical model in which the logical neighbors coincide with the physical neighbors. The following assumptions are made in the evaluation of the throughput for the token bus:

(a) An explicit token scheme is considered; and

(b) The token is not an integral part of the packet, but is transmitted as a separate packet.

(c) The logical ring is assumed to be static, i.e., no addition or deletion of stations is possible.

(d) The total length of the network is assumed to be 1,000 meters.

The throughput for this model can be derived to the form as shown in the equation (2)

$$S = 1 / (1+2a/N + q) \tag{2}$$

where **a** is the normalized propagation delay defined as

$$a = Rd / V L \tag{3}$$

where **R** is the data rate(channel capacity), **d** is the length of the total length of the network, **V** is the propagation velocity, which is about two- thirds of the speed of light, and **L** is the length of the data frame transmitted.

and **q** is the normalized token overhead delay defined as

$$q = \text{token transmission time} \Big/ \text{message transmission time} \qquad (4)$$

Consider the model with 20 stations and with the following parameters:

Packet length = 4096 bits

Token length = 112 bits

Speed of the channel = $2 * 10^8$ m/sec.

Capacity of the channel = 10 Mbps.

Length of the network = 1,000 meters.

Substituting these parameters in the above formula for the throughput , we get the value of the throughput as S = 0.969.

For the same packet length of 4096 bits, the simulated model yields a throughput of 0.972.

Decreasing the packet length to 1024 bits, with the same token length of 112 bits the analytical model gives a throughput of 0.88, where as the simulated model yields a throughput of 0.87.

Delay analysis:

A system with a known worst case is a ring with the stations take turns sending data packets. We consider a logical ring with N stations in which the token is rotating. Each data packet has to wait in the buffer until the station gets the token to transmit the data packet. We define the worst case token rotation time as the time taken by the token to go a round the logical ring once. This includes the holding time of the token at each station. The average packet delay is the average time taken by the data packet to transmit from the source station to the destination plus the time spent by the data packet in the source buffer until the station receives the token. Thus it can be seen that the average packet delay depends on the token rotation time.

The general expression for the worst case token rotation time for the best case of the logical distribution of the stations is given by the equation (5).

$$T_d = T(N + Nq+2a) \tag{5}$$

where T is the data packet transmission time.

For the model with 20 stations and a packet length of 4096 bits, computation of the worst case token rotation time gives a value of 8.42 milli seconds.

For the same parameters the simulated model gives a worst case token rotation time of 8.37 milli seconds.

For a packet length of 1024 bits, the analytical model gives the worst case token rotation as 2.28 milli seconds. The simulated model gives a worst case token rotation of 2.25 milli seconds.

Thus it is observed that the simulated model gave the results very close to those of analytical results.

CHAPTER VII
CONCLUSIONS

Simulation study of several aspects of the Token Bus performance is presented. At lower loads the overhead due to the token-passing among the stations causes low channel utilization yielding low throughput. But at higher loads Token Bus yields an excellent throughput without showing any degradation. The use of longer packets yields better utilization of the channel but with an increase in the average packet delay. The offered load to the network is never reduced to zero due to the token passings even if none of the stations have data packets to transmit. The performance of the Token Bus improves as the number of stations within a given length of the network is increased. For a given number of stations and a given packet length the throughput of the network decreases as the network length increases.

The physical distribution of the stations on the bus has a least effect on the throughput of the network. When the stations are distributed uniformly the difference of the throughput between the best case and the worst case of the logical distribution of the stations is negligible. On the other hand when the stations are distributed in the form of clusters there is a slight decrease in the throughput in the case of the worst case. Finally a comparison of the performance of the token bus model is compared with that of another model using CSMA/CD access method.

The simulated model in this thesis does not consider the prio
mechanism offered by the token bus protocol. The token bus protocol enable

priority mechanism in which the data packet s have different priorities. For future work, extension of the present model is possible so as to include the priority mechanism. An extensive study can be made to see how the bandwidth will be allocated for various prioritized data packets. Further the the traffic generating environment can be changed and the performance of the model can be studied under these conditions.

APPENDIX A

CSIM AND SOURCE LISTING OF SIMULATOR

CSIM is a process oriented simulation language which is implemented as a superset of the C programming language. In CSIM, a process is an 'independent' program or procedure which can execute in 'parallel' with other processes (the processes only appear to execute in parallel, CSIM simulates parallel or simultaneous sequences of activities). CSIM provides an extended set of features which facilitate the implementation of process oriented simulation models. These are implemented as a set of extensions to the C programming language. A CSIM program accesses these features via function or procedure calls from a C program constructed by the modeler.

CSIM applied to communication system: In the CSIM model the communication medium is declared as a *facility* , which is a data object and can be *reserved* and *released* by processes. A facility can be in one of the two states; BUSY or FREE. If a process *reserves* a FREE facility, the facility is assigned to that process and the process continues. If a process *reserves* a BUSY facility, the process is suspended until another process *releases* that facility. When this happens, the waiting process is given control of the facility and execution resumes.

Summary of the CSIM statements used:

CSIM is embedded in C programming language. In addition to C, the following statements of CSIM are used in the simulation.

Declarations and Initializers: Initializers are really executable CSIM functions which return pointers to CSIM data structures. Initializer statements must be executed after the *create* statement in the respective processes.

1. EVENT ev;

    ev = event("name");

The above declaration declares ev(a variable of type EVENT) to be a pointer to an event of name "name". Events declared in the sim(first) process are globally accessible. Events declared in another processes are local to the declaring process. These local events can be passed as parameters to other processes. Local events are deleted when the declaring process terminates.

The transmissions are declared as *events* in the CSIM model. An *event* is another kind of data object, has two states; *occured* and *not-occured.* When a process *waits* for an event which is in the*occured* state, the event is automatically placed in the *not-occured* state and the process continues. When a process *waits* for an event which is in the *not-occured* state, execution of the process is suspended. When some other process *sets* that event, the event is placed in the *occured* state.

2. FACILITY f;

    f = facility ("name");

The above declaration declares f (a variable of type FACILITY) to be a facility of name "name". Facilities should be declared with global variables and initialized in the sim(main) process, prior to the beginning of the simulation part of the model.

Execution Statements:

1. clear(ev);

This statement resets event ev to a *not-occured state.* The event must have been declared to be of type EVENT and initialized using an *event* statement.

2. hold(t);

This statement is used to suspend a process for a simulated time interval of length t. The variable t should be of type float.

3. release(f);

This statement is used to release a facility f. f must have been declared to be of type FACILITY and initialized with a *facility* statement.

4. reserve(f);

This statement is used to reserve a facility f. If the facility is already reserved by other processes, then the reserving process is suspended until the facility is released so that it can gain access to the facility.

5. set(ev);

This statement sets an event ev to the *occured* state.

6. t = simtime();

This statement sets t to the current simulated time. This is a function of type float.

7. terminate();

This function is used to end of terminate a process that is executing. A *terminat* statement is not required if the process exits normally.

8. rerun();

The rerun statement causes the simulation support system to be completely reinitialized (with the exception of the random number function). This feature is used to obtain several independent executions of the models.

Operation: The CSIM command file, csim should be copied to the present working directory or a path is to be set properly so as to access it. The CSIM programs are compiled by the command *csim filename.c* and executable module is placed in the file a.out. All the standard options to the C compiler can be used, including the -o option which specifies a name for the executable module. The command *a.out -T* will cause a CSIM debugging event trace to be created on the standard output file. The first executable procedure must be named *sim*. *Sim* must contain a *create* function. The header file *csim.h* must be included with proper path at the beginning of the program. The *create* function causes a procedure to be set up as a process, capable of executing in a pseudo-parallel manner with other processes.

DESCRIPTION OF THE CSIM CODE
FOLLOWS

```
************************************************************************************
               TOKEN BUS MODEL WITH TWENTY STATIONS IN A LENGTH OF 'L' METERS
************************************************************************************

    /* Header Files */

#include <stdio.h>
#include <math.h>
#include "/usr/mesuna/csim/lib/csim.h"
#include "/usr/mesunb/users/rxk8784/exper/tokens.h" /* contains the parameters like
        packet length,token length,capacity of the network, propagation speed etc.*/
************************************************************************************

        /* Global variable declarations */
int total_dist;
struct tkn find_token();
int stn_initialize();
static int stn_distance[] = {
50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50}; /*initializing the
                                                   inter station distances */
static int order[]={
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19 };  /* initializing the
                                              logical ordering of the staions */
EVENT ev0, ev1, ev2, ev3, ev4, ev5, ev6, ev7, ev8, ev9, ev10, ev10, ev11, ev12, ev13,
ev14, ev15, ev16, ev17, ev18, ev19;                      /* declaring the events */
int find_av_stn(),tot_phy_len();
extern float prob();
extern int random();
float TOKEN_HOLD_TIME;
float no_rotations;
float extra_time;
struct stn    /* structure that holds the details of the current data packet */
{
        float mess_arr_time;
        float mess_dep_time;
        int stn_dist;
}*station[NO_OF_STATIONS],*phy_stn[NO_OF_STATIONS];

struct tkn  /* structure holding the details of the station currently holding
                                                  the token */
{
        int stn_no;      /* number of the station holding the token currently */
        int token_dist;  /* distance travelled by the token */
        int rev_dir;     /* distance travelled by the token after it crossed the
                                        last station in the logical ring */
        float time_spent;    /* time spent by the token at the station */
}token;
static int check=0;
/* Beginning of the sim process */

sim()
{
        int i,pps1=0,failed=0,inter_gap,no_tokens=0,ttokens=0, data_present;
        float C = 0.0,tot_token_time 0.0,maxdelay = 0.0, Message_delay = 0.0,
        packet_delay = 0.0,max_time=0.0,sim_time=0.0,current_time,token_delay1=0.0,
token_delay2=0.0,message_delay=0.0,average_delay=0.0,total_delay=0.0,
av_token_delay=0.0,
av_packet_delay=0.0,sum_packet_delay=0.0,S=0.0,D=0.0,av_tk_delay = 0.0,
tot_tk_delay = 0.0;
int tot_phy_dist, rand_source_no,rand_dest_no,total_dist,count = 0,;
int ret_stn,path1,path2,token_pos;

create("sim");
/* initializing all the events */
ev0 = event("ev0");
ev1 = event("ev1");
ev2 = event("ev2");
ev3 = event("ev3");
ev4 = event("ev4");
ev5 = event("ev5");
ev6 = event("ev6");
ev7 = event("ev7");
ev8 = event("ev8");
ev9 = event("ev9");
ev10 = event("ev10");
ev11 = event("ev11");
ev12 = event("ev12");
ev13 = event("ev13");
ev14 = event("ev14");
ev15 = event("ev15");
ev16 = event("ev16");
ev17 = event("ev17");
ev18 = event("ev18");
ev19 = event("ev19");
token = facility ("token");

set(ev0);       /* setting all the events*/
set(ev1);
set(ev2);
set(ev3);
set(ev4);
set(ev5);
set(ev6);
set(ev7);
set(ev8);
set(ev9);
set(ev10);
set(ev11);
set(ev12);
set(ev13);
set(ev14);
set(ev15);
set(ev16);
set(ev17);
set(ev18);
set(ev19);
TOKEN_HOLD_TIME = TOKEN_TIME + (tot_phy_dist/TOKEN_PROP_SPEED);
tot_token_time = (NO_OF_STATIONS)*TOKEN_HOLD_TIME;
total_dist = stn_initialize();
while(count < NO_OF_PACKETS)
{
```

```
                no_tokens = 0;
                current_time =prob();
                inter_gap = random(R1,R2);
if (ppsl==0)
{
 inter_gap=0;
ppsl=1;}
                current_time += inter_gap;
                rand_source_no = random(0,NO_OF_STATIONS -1);/*random function to
                                                           pick the source station */

                rand_dest_no =  random(0,NO_OF_STATIONS - 1); /*random function to
                                                          pick up the destination*/
                sim_time += current_time; /* sim_time keeps track of
                                                  the current simulated time */
                if(station[rand_source_no]->mess_dep_time > sim_time)
                {
                        ret_stn=find_av_stn(rand_source_no,sim_time);
                        if(ret_stn == -1)
                                {
                                token = find_token(token,current_time);
                                continue;
                                }
                        else
                                rand_source_no=ret_stn;
                }
                while(rand_source_no==rand_dest_no)
                {
                        rand_dest_no=random(0,NO_OF_STATIONS-1);
                }
        count++;
        printf("%2d,%9.3f, ",count,sim_time);  /*  used for debugging */
        printf("%2d,%2d,",rand_source_no,rand_dest_no); /* used for debugging  */
                if(sim_time == current_time)
                {
                        token.stn_no = order[0];
                        token.token_dist = phy_stn[order[0]]->stn_dist;
                        token.time_spent = 0;
                        token.rev_dir=0;
                }

                else
                        {
                        token = find_token(token,current_time);
                        }
      /* this printf is for debugging */
        printf ("%4d, %4d, %8.3f, ",token.stn_no,token.token_dist,token.time_spent);
                token_pos=token.token_dist+token.rev_dir;
                if(token_pos <= station[rand_source_no]->stn_dist)
token_delay1 =(station[rand_source_no]-> stn_dist - token.token_dist)/TOKEN_PROP_SPEED
                else
token_delay1 = (station[rand_source_no]-> stn_dist+ total_dist-token.rev_dir-
                                                   token.token_dist)/TOKEN_PROP_SPEED;

                if(token_delay1 < 0)
token_delay1 = (total_dist-token.token_dist +
                        station[rand_source_no]->stn_dist)/TOKEN_PROP_SPEED;

                token_delay2 = (rand_source_no - token.stn_no - 1) * TOKEN_HOL
                if(token_delay2 < 0)
                token_delay2 = (NO_OF_STATIONS - token.stn_no-1 +
                                                rand_source_no) * TOKEN_HOLI
                no_tokens += token_delay2/TOKEN_HOLD_TIME;
                if(sim_time==current_time)
                        token_delay2 += TOKEN_HOLD_TIME;
                if(token.time_spent > 0)
{
                        token_delay2 += TOKEN_HOLD_TIME - token.time_spent;
                        no_tokens++;
}
if(token.stn_no==rand_source_no && (token.token_dist==0 || token.time_spent > 0))
                {
                        token_delay2 = (tot_token_time-token.time_spent);
                        token_delay1 = (total_dist/200.0);
                        no_tokens = NO_OF_STATIONS-1;
                }
                tot_tk_delay += (token_delay1 + token_delay2);
if ( data_present == 1)   /*if the buffer of the station has the packet to send
                                                when the token is received */
        {
        reserve(token);
        hold (TOKEN_HOLD_TIME);  /*holding the token for the hold time when the
                                        station has a data packet to transfer  */
        release(token); /*release the token and pass it to the successor */
        }
else            /* if the buffer of the staion does not have a
                                                data packet to transmit */
        {
                release (token); /* pass the token to its successor */
        }
/* message_delay gives the propagation delay of the data packet from the source t
                                                        destination station */

message_delay =(phy_stn[rand_dest_no]->stn_dist - phy_stn[rand_source_no]->stn_di
                                                        MESSAGE_PROP_SPEED;

                if (message_delay < 0)
                   message_delay *= -1;

/* Message_delay is the sum of the transmission and propagation delays
                                                        of the data packet */

Message_delay = message_delay +(float) PACKET_LENGTH/CAPACITY;

/*packet_delay actual delay experienced by a data packet which includes the waiting
 time  of the data packet in the buffer of the source station */

                packet_delay = Message_delay + token_delay1+token_delay2;

if(packet_delay > maxdelay)

        maxdelay = packet_delay;  /*calculates the maximum delay experienced
```

```
                                                by a data packet */
       sum_packet_delay += packet_delay; /* total delay of all the data packets */
if(sim_time+token_delay1+token_delay2+Message_delay > max_time)

max_time = sim_time+token_delay1+token_delay2+Message_delay; /* max_time is the total
                                                              simulated time */
station[rand_source_no]->mess_dep_time = sim_time + token_delay1 +token_delay2;

total_delay += Message_delay;

/*total_delay is the sum of the transmission and
                                        propagation delays of all data packets */
       ttokens += no_tokens; /* keeps track of the total token transmissions during
                                                the entire simulation */

       }/*for while loop*/

av_packet_delay = sum_packet_delay/NO_OF_PACKETS;/*computes the average packet delay*/

S = ((total_delay/max_time))/10.0;
/*S computes the normalized throughput of the model */

D =(((PACKET_LENGTH*NO_OF_PACKETS) +(ttokens*TOKEN_LENGTH))/max_time)/10.0;
/* D computes the normalized offered load */

G = ((PACKET_LENGTH*NO_OF_PACKETS)/max_time)/10.0;
/* G computes the normalized input load */

printf(" %.3f,  %.3f  %.3f,  %.3f , %.3f\n",D,G,S,av_packet_delay,maxdelay);
rerun();
}

/* the structure below is used to find the position of the token
                         at a given instance of time */

struct tkn find_token(present,present_time)
struct tkn present;
float present_time;
{
       float cal_time = 0.0,prop_delay;
       int next_stn,token_stn_dist,cond=0,pps=0;
       char c;
if(present.rev_dir > 0)
{
prop_delay = (total_dist-station[NO_OF_STATIONS-1]->stn_dist
                               - present.rev_dir)/TOKEN_PROP_SPEED;
               if (prop_delay <= present_time)
               {
                       cal_time = prop_delay;
                       check=0;
                       present.stn_no=order[0];
                       present.token_dist=station[order[0]]->stn_dist;
                       present.rev_dir=0;
                       pps=1;
               }
               else
```

```
                {
                        cal_time = 100000;
                        present.rev_dir += present_time*TOKEN_PROP_SPEED;
                }
        }
        if(present.time_spent > 0)
        {
                if(TOKEN_HOLD_TIME - present.time_spent <=  present_time)
                {
                        cal_time = TOKEN_HOLD_TIME - present.time_spent;
                        present.time_spent = 0;

                }

                else
                {
                        cal_time = 100000;
                        present.time_spent += present_time;
                }
        }

        if(cal_time != 100000)
                {
                while(cal_time <= present_time)
                {
                if(cond==1 || check==0)
                {
                        check=1;
                        if(cal_time + TOKEN_HOLD_TIME > present_time)
                        {
                                present.time_spent = present_time - cal_time;
                                cal_time=100000;
                        }
                        else
                                        cal_time += TOKEN_HOLD_TIME;
                        }
                        if(cal_time !=100000)
                        {
                                next_stn = (present.stn_no) + 1;
                                if((next_stn == NO_OF_STATIONS) && pps==0)
                                {
                        token_stn_dist=total_dist - phy_stn[next_stn - 1]->stn_dist;
                        prop_delay = token_stn_dist/TOKEN_PROP_SPEED;
                if(prop_delay > present_time - cal_time)
 {
        present.rev_dir += (present_time - cal_time)*TOKEN_PROP_SPEED;
        cal_time=100000;
}
                                else
                                {
                                present.stn_no = next_stn;
                                if (next_stn==NO_OF_STATIONS)
                                next_stn=order[0];
                                present.token_dist = station[next_stn]->stn_dist;
                                cal_time += prop_delay;
                                }
```

```
                                        next_stn=order[0];
                                        }
                                        if(next_stn == NO_OF_STATIONS)
                                        {
                                                next_stn=order[0];
                                        }
                                        if(cal_time != 100000)
                                {
                                token_stn_dist=station[next_stn]->stn_dist-present.token_dist;
                                if(token_stn_dist < 0)
                                {
                                token_stn_dist=station[next_stn]->stn_dist;
                                next_stn=1;
                                }
                                prop_delay = token_stn_dist/TOKEN_PROP_SPEED;
                                if(prop_delay > present_time - cal_time) {
                                present.token_dist += (present_time - cal_time)*TOKEN_PROP_SPE
                                if(present.token_dist > total_dist)
                                present.token_dist -= total_dist;
                                cal_time = 100000;
                                }
                                        else
                                        {
                                        present.stn_no = next_stn;
                                        present.token_dist = station[next_stn]->stn_dist;
                                        cal_time += prop_delay;
                                        }
                                        }
                                }

                                cond=1;
                        }
                }

                return(present);
}

/* the function below is used to initialize the distribution of the staions
   given inter station distances and the logical ordering of the staions */
stn_initialize()
{
        int total_distance;
        int i,dist=0;
        for(i=0;i<NO_OF_STATIONS;i++)
        {
                phy_stn[i] = (struct stn *) malloc(sizeof(struct stn));
                station[i] = (struct stn *) malloc(sizeof(struct stn));
                dist += stn_distance[i];
                phy_stn[i]->stn_dist = dist;
                phy_stn[i]->mess_arr_time = 0;
                total_dist += stn_distance[order[i]];
                station[i]-> stn_dist = total_dist;
                station[i]-> mess_arr_time = 0;
        }
        total_dist += phy_stn[i-1]->stn_dist;
        return(total_dist);
```

```
}

tot_phy_len()  /* this function is used to calculate the total length of the network w
                            inter station distances */
{
        int i,len=0;
        for(i=0;i<NO_OF_STATIONS;i++)
                len += stn_distance[i];
        return(len);
}
find_av_stn(busy_stn,at_time)  /* this function is used to pick up the source station
                                     generator when all the staions are busy */
int busy_stn;
long float at_time;
{
        int i,stn;
        for(i=1;i<=NO_OF_STATIONS;i++)
        {
                stn = i+busy_stn;
                if(stn > NO_OF_STATIONS-1)
                        stn -= NO_OF_STATIONS;
                if(station[stn]->mess_dep_time < at_time)
                        return(stn);
        }
        return(-1);
}
```

REFERENCES

[1] James Martin *Local Area Networks Architecture & Implementation*, Arben Group, Prentice Hall, 1984.

[2] IEEE Standards for Local Area Networks: Token Bus Accessing method ANSI/IEEE Std 802.4-1985 ISO/DIS 8802/4.

[3] Stallings, William, "Local Network Performance", *IEEE Communications Magazine, vol 22, pp. 3-41, Mr '84.*

[4] Herb Schwetman, "CSIM: A C-Based, Process-oriented simulation Language," *Research Report*, Microelectronics and Computer Technology Corporation, Austin, Texas.

[5] Tanenbaum A.S. , *Computer Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[6] Mischa Schwartz., *Telecommunication NETWORKS Protocols, Modeling and Analysis.* Addison-Wesley Publishing company, 1987.

[7] Kernighan, B.W. and D.M. Ritchie, *The C Programming Language*. Engle-Wood Cliffs, NJ: Prentice Hall, 1978.

[8] Herb Schwetman, *CSIM Reference Manual(Revision 12)*. Austin, Texas: Mcc,1987.

[9] Anura P. Jayasumana , "Throughput Analysis of the IEEE 802.4 Priority Scheme", *IEEE Transactions on Communications*, Vol 37, No.6, June 1989. pp.565-571.

[10] A.K.Sood, S.Akhtar and K.Y. Srinivasan, "An extended Token bus Protocol for embedded networks", *Computers & Elect Engg* Vol.14 No.3/4, pp.105-123, 1988.

[11] Peter MARTINI, Otto SPANIOL, "Token-passing in high speed backbone networks for campus-wide environments," "Modeling Techniques and Performance Evaluation, Elsevieer Science Publishers , 1987.

[12] Nanda K. Kanuri "CSMA/CD LAN Performance: Modeling and simulation in CSIM", Vol 20 Part 3, Proceedings of the Twentieth Annual Pittsburgh Conference held May 4-5, 1989. pp 1003-1007.