

# **SIMULATION OF CSMA/CD LAN WITH NETWORK PARTITIONING**

*Dissertation submitted to the Jawaharlal Nehru University  
in partial fulfilment of the requirements  
for the award of the Degree of*  
**MASTER OF TECHNOLOGY**

*in*  
**COMPUTER SCIENCE AND TECHNOLOGY**

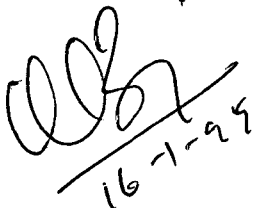
*by*  
**PANGULURI RAMARAO**

**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES  
JAWAHARLAL NEHRU UNIVERSITY  
NEW DELHI - 110067  
INDIA.**

## CERTIFICATE

This is to certify that the thesis entitled "**Simulation of CSMA/CD LAN with Network Partitioning**", being submitted by me to Jawaharlal Nehru University in partial fulfilment of the requirements for the award of Degree of Master of Technology, is a record of original work done by me under the supervision of Prof. Karmeshu, Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University and Mr. S. Madan, Asst. Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University, during the Monsoon Semester 1994.

The results reported in this thesis have not been submitted in part or full to any other university or Institution for the award of any degree.

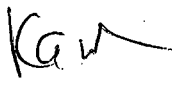


16-1-94

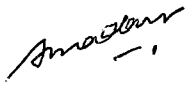
**Prof. K.K. Bharadwaj**  
Dean, SC & SS  
Jawaharlal Nehru University  
New Delhi  
Pin 110067.



**PANGULURI RAMA RAO**



**Prof. Karmeshu**  
Professor,  
SC & SS  
J.N.U., New Delhi.



**Mr. S. Madan**  
Asst. Professor  
SC & SS  
J.N.U., New Delhi.

## ACKNOWLEDGEMENTS

It is a great pleasure for me to sincerely express my deep sense of respect and gratitude to Prof. Karmeshu for his invaluable suggestions, guidance and inspiration.

I express my special gratitude to Mr.S.Madan for his invaluable suggestions and for his constant encouragement during the course of the project.

I take this opportunity to thank Prof. K.K. Bharadwaj for providing me the opportunity to undertake the project. I would also like to thank the authorities of our School for providing me the necessary facilities to complete my project.

I am thankful to Michael, Gopal and John for their support.

PANGULURI RAMARAO

## ABSTRACT

The performance of CSMA/CD LAN degrades under heavy load. In my thesis, by using the concept of network partitioning an upgraded CSMA/CD LAN is implemented which does not degrade the performance under heavy load.

Initially, the software model has been developed for the CSMA/CD LAN. The event scheduling approach has been used to model the protocol, the various parameters of the protocol can be user specified or the appropriate 802.3 standards may be used for simulation runs. The simulation result produced by the software module are validated by comparing them with the Schoch's measured results [6] and the analytical results. Finally, the software model was modified to simulate the CSMA/CD LAN with network partitioning. The simulation run provides the mean packet delay averaged on all the nodes in the LAN.

The simulation results of the LAN with and without network partitioning are presented the results show that the performance of CSMA/CD LAN improves with the network partitioning under heavy load.

# CONTENTS

## CHAPTER 1 INTRODUCTION

1.1 Local Area Networks (LAN)	1
1.2 Media Access Protocols in LANs	2
1.3 Performance parameters and Measures	6
1.4 Objectives	7

## CHAPTER 2 CSMA/CD WITH NETWORK PARTITIONING

2.1 Network Partitioning	9
2.2 Partition station and its functions	10
2.3 States of partition station	12
2.4 Advantages of dynamic partitioning	13

## CHAPTER 3 OVERVIEW OF DISCRETE EVENT SIMULATION

3.1 Definitions	14
3.2 Discrete Event Simulation techniques	14
3.3 Why not special purpose simulation languages	16
3.4 Validation and its importance	17
3.5 Random number generation	18
3.6 Simulation output Analysis	21

## CHAPTER 4 MODELLING

4.1 Ethernet Simulation Model	26
4.2 Modelling Assumptions	33
4.3 Input and Output Parameters of Model	34
4.4 Validation of Simulation Model	35

## CHAPTER 5 PROGRAMMING APPROACH FOR THE MODEL

5.1 Development of Ethernet Model	38
5.2 Development of Model with Network Partitioning	43
5.3 Major Modules defined in program	44

## CHAPTER 6 CONCLUSION

6.1 Validation of Ethernet Model by comparing with Schoch's Measured results and with Ethernet Analytical results	51
6.2 Simulation of two segment Ethernet with Network partitioning.	52
6.3 Conclusion	54

## PROGRAM LISTING

## BIBLIOGRAPHY

## CHAPTER 1

### INTRODUCTION

#### 1.1 LOCAL AREA NETWORKS (LAN)

Local Area Networks are widely used nowadays in the offices.

A Local Area Network is defined as:

"A Local Area Network is a communication network that provides interconnection of a variety of data communicating devices within a small area".

Some of the key characteristics of local area networks are:

- . high data rates (up to 100 Mbps)
- . Short distances (0.1-50 km)
- . low error rate ( $10^{-8}$ - $10^{-11}$ )
- . geographically confined to a small area
- . generally privately owned

Local networks are characterized in terms of their topology. Bus and ring topologies widely became popular. In Bus topology, all devices share a common communication medium, only one device can transmit at a time, and transmission employs a packet containing source and destination address fields and data. The ring topology consists of a closed loop, with each node attached to a repeating element.

## 1.2 MEDIUM ACCESS PROTOCOLS IN LANS

The most commonly used MAC protocols are CSMA/CD, tokenring and tokenbus.

In the CSMA/CD protocol, a terminal with a packet ready for transmission senses the channel and proceeds or follows:

(1) If the channel is sensed idle, the terminal initiates transmission of packet.

(2) If the channel is busy, then depending on the persistence algorithm the terminal transmits. There are mainly three persistence algorithms. They are nonpersistent, 1-persistent and p-persistent.

In the nonpersistence case, the station backoff a random amount of time and then senses the medium again. It is effective in avoiding collisions; two stations wishing to transmit when the medium is busy are likely to backoff for different amounts of time. The drawback is that there is likely to be a wasted idle time following each transmission.

In the 1-persistent algorithm, the station continues to sense the medium until it is idle, and then it transmits. This attempts to reduce the idle time by allowing a single waiting station to transmit immediately after another transmission. Unfortunately, if more than one station is waiting, a collision is guaranteed.

In the p-persistent algorithm, the station continues to

sense the medium until it is idle, then transmits with some pre assigned probability. Otherwise it backs off a fixed amount of time, then transmits with probability  $p$  or continues to backoff with probability  $(1-p)$ . This algorithm is a compromise that attempts to minimize both collisions and idle time.

(3). If a collision is detected during transmission, immediately cease transmitting the packet and transmit a brief jamming signal to ensure that all stations know that there has been a collision. After transmitting jamming signal, wait a random amount of time, then attempt to transmit again.

For the proper operation of the CSMA/CD protocol the packet size should be sufficient to permit collision detection in the worstcase. For a baseband system, with two stations that are as far apart as possible (worstcase), the time that it takes to detect a collision is twice the propagation delay.

The most common choice in the persistence algorithms is 1-persistent, used by the Ethernet and IEEE 802 standard. With the 1-persistent scheme, the wasted idle time is eliminated at the cost of wasted collision time.

The time wasted due to collisions is short if the packets are long relative to the propagation delay. With random backoff, two stations involved in a collision are



unlikely to collide on their next tries. To ensure stability of this backoff, a technique known as binary exponential backoff is used. A station attempt to transmit repeatedly in the face of repeated collisions, but the mean value of random delay is doubled after each collision. After a number of unsuccessful attempts the station gives up and reports an error.

Token Bus is a technique in which stations on the bus or tree form a logical ring; that is, the stations are assigned positions in an ordered sequence, with the last member of the sequence followed by the first. Each station knows the identity of the stations preceding and following it. When a station receives the token, it is granted control of the medium for a specified time, during which it may transmit. When the station is done or time has expired, it passes the token on to the next station in the logical sequence. Hence steadystate operation consists of alternating data transfer and token transfer phases.

In the case of token ring, a small token packet circulates around the ring: when all stations are idle, the token packet is labeled as 'free' token. A station wishing to transmit waits until it detects the token passing by, alters the bit pattern of the token from "free" token to "busy" token and transmits the packet immediately following busy token. Since there is no free token on the ring, all

other stations wishing to transmit must wait. The packet on the ring will make a round trip and be purged by the transmitting station. The transmitting station inserts a "freetoken" on the ring when the station has completed transmission of its packet and busytoken has returned to the station. When transmitting station releases a new free token, the next station downstream with data to send will be able to seize the token and transmit.

The comparison between tokenring, token bus and CSMA/CD is done by many. The analysis [3] of these comparisons yielded the following conclusions.

- (1) Token ring is least sensitive to workload.
- (2) CSMA/CD offers shortest delay under light load, whereas it is most sensitive under heavy load to the workload.

Under heavy load, the disparity between token passing and CSMA/CD is due to the instability of CSMA/CD. As offered load increases, so does throughput, until beyond some maximum value, throughput actually declines as offered load increases. This results from the fact that there is an increased frequency of collisions. More packets are offered but fewer successfully escape collision. Worse, those packets that do collide must be retransmitted, further increasing the load.

### 1.3 PERFORMANCE PARAMETERS AND MEASURES

In the performance analysis of Local Area Network the two most useful parameters are the Data Rate (R) of the medium and the average signal propagation delay (D) between the stations on the network. The product of these two terms  $R \times D$ , is the important parameter for determining the performance of a Local Area Network.

Various measures for the analysis of LANs are throughput and meanframe delay and utilization as a function of network load. Network load is characterized by the number of stations on the network, the interframe transmission intervals of these stations and mean frame length of transmissions.

Throughput can be defined interms of number of frames per and time or bytes or bits per unit time. It can also be defined as that part of channel utilisation used for successful data transmission and otherpart of channel utilization represents transmissions abandoned because of collisions. The definition which is in use is "the part of channel utilisation used for successful date transmission", which is referred to as Network Throughput(S).

The mean frame delay time  $T_d$  is defined from the time at which a station first attempts to transmit a frame to the time at which transmission of frame is successfully

completed. It includes the channel waiting time, collision recovery time, backoff time and frame transmission time.

If  $T_f$  be the mean transmission time of a frame in the absence of contention:  $T_f$  is the product of frame length in bits and channel transfer rate in bits per sec. The normalized mean delay (D) is defined as:

$$D = T_d / T_f$$

The Ratio of meanframe delay to the mean frame transmission time.

The load on the network is called the offered load, G. If  $T_i$  is the mean interval between the time at which a station completes transmission of one frame and the time at which it initiates transmission of next frame and if there are N stations in the network and if all are identical and if the network is assumed a closed system, the offered load G is defined as

$$G = \frac{NT_f}{T_i + T_f}$$

If the network is viewed as an open system, the offered load G, is

$$G = \lambda T_f$$

Where  $\lambda$  is the arrival rate of new request at the channel  $\lambda = N/T_i = N\lambda_i$

Where  $\lambda_i$  is the arrival rate of frames at a particular station.

One more parameter in the performance of CSMA/CD networks is the ratio of end to end propagation time of the channel to the mean frame transmission time and is denoted by  $\alpha$ .

$$\alpha = T/T_f$$

= propagation time/frame transmission time

For a given offered load, network with large  $\alpha$  will spend more time in the collision recovery, and so have lower throughput than a network with large  $\alpha$ .

#### 1.4 OBJECTIVE

The CSMA/CD protocol performs well under light network load but not well at heavy load. Network partitioning allows the network to be partitioned in to segments when under heavy load and the partition station acts as a bridge between segments. By using this concept the objective is to show that performance of CSMA/CD improves under heavy load by measuring the throughput and average packet delay for Ethernet.

...

Parameters	Values
Slottime	512 bit times
Jam size	32 bit
Inter frame gap	9.6 micro sec.
Maximum number of Retransmission attempts	16
Maximum backoff possible	10
Address size	48 bits
Minimum Frame size	512 bits
Maximum Frame Size	1518 bytes

Table 1.1 IEEE 802.3 standards

## CHAPTER 2

### CSMA/CD WITH NETWORK PARTITIONING

#### 2.1 NETWORK PARTITIONING

Consider a room of people who have the ability to send and receive data through the speech. The communication that can take place in the room can be increased by partitioning the room into compartments with each compartment isolated from the other. And also each compartment can pass speech across the partitions to adjacent compartments. Then it is possible to have the communication taking place simultaneously in each compartment. If the same is applied to a CSMA/CD LAN with the room being the physical network and the people, set of stations on the network, then it is possible to increase the performance.

Partitioning the LAN into separate segments with each segment isolated from the other segments and having the communication taking place simultaneously in each segment. Each segment has the capability to pass messages across partitions to adjacent segments. This is the concept of Network partitioning.

The following are the benefits of employing the Network partitioning for the CSMA/CD.

- 1) It gives maximum aggregate throughput.
- 2) It does not degrade performance under light load.

## 2.2 PARTITION STATION AND ITS FUNCTIONS

The partition station has two important functions.

- (1) At the place where partition station exists, putting and removing the partition.
- (2) When the partition is present i.e. when it is separated in to separate segments, the function of partition station is to store and forward packets from one side of the partition station to the otherside, in case packets destination is on the otherside.

Major components of the partition station are as shown in Fig. 2(a). When the LAN is not separated in to segments i.e. when the partition is not in place, the partition station uses the collision frames input to determine when to place the partition. When the network is under heavy load i.e. when the number of collision frames per second exceeds certain value, then the partition is to be put in place and when the number of collisions per second falls below a certain value, which indicates that the network load is light, the partition is to be removed. If the buffer of the partition station is full then also partition is to be removed.

Depending on the output of the partition station, the placement and removal of partition is controlled. The inputs to the partition station are collision frames input, pass through traffic and buffer full condition. Onto either



side of partition mechanism the partition station is connected to the network and the connection is equivalent to the connection of a station to the network.

The partition station is itself a bridge and is a Medium Access Control (MAC) layer bridge. The function of a bridge is as follows: A bridge accepts an entire frame and passes it up to the data link layer where the checksum is verified. Then the frame is sent down to the physical layer for forwarding on a different subnet. A MAC layer bridge operates below the network layer in the open system interconnection (OSI) reference model. It operates within the data link layer. Two IEEE approaches in the design of the bridges exist. One among them is transparent bridge. The transparent bridge operates in promiscuous mode, accepting every frame transmitted on all the LANs to which it is attached. The routing procedure for an incoming frame depends on the LAN it arrives on (the source LAN) and the LAN its destination is on (the destination LAN), as follows:

1. If the destination and source LAN's are the same, discard the frame.
2. If the destination and source LANs are different, forward the frame.
3. If the destination LAN is unknown, use flooding i.e. keeping the packet on all LANs.

### 2.3 STATES OF PARTITION STATION

The different states of a partition station are shown in Fig 2 (b).

Before partitioning, the partition station is said to be in the 'Waiting' state. While in the waiting state partition station is transparent to the network. When the network is under light load, partition station stays in 'Waiting' state. On this state, the partition station continuously senses the number of collisions per second, which is an indicator of load on the network. When it exceeds a certain value, the partition station puts the partition in place i.e. the partition station goes into 'bridge' state, where it acts as a MAC layer bridge between the two sides of the network. In this state, the partition station continues to monitor the number of collisions per unit time on both the sides of partition. If the total collisions per unit time drops below a certain value, indicating that network load is light, the partition station removes the partition and goes in to 'empty buffer' state. In this state initially the partition station transmits all packets in its buffer. After all the packets in partition station are transmitted, the partition station goes in to 'waiting' state. When the partition station is in 'bridge' state, if the buffer gets full, then partition station removes the partition and goes into 'empty buffer' state. Otherwise, the packets will overflow because of buffer full condition.

The implementation of network partitioning improves the performance of CSMA/CD LAN and the performance varies with the traffic patterns on the network.

#### 2.4 ADVANTAGES OF DYNAMIC PARTITIONING

Dynamic partitioning has some advantages over building permanent partition stations, they are :

(1) A permanent partition or bridge slightly degrades performance under light load. The partition station takes non zero processing time at the partition station to store packets and forward them. When you don't want to degrade the network performance under light load, the dynamic partitioning is the best choice.

(2) In case a failure occurs in a permanent station or bridge the network is at a minimum disconnected. In dynamic partitioning partition station could be constructed in such a way that it would handle failure by removing the partition. Unlike a permanent partition station, after removing partition, the network remains connected.

(3) Dynamic partitioning would be an advantage in situations where known traffic patterns exist on a network such that partitioning could be scheduled. For example, traffic patterns are known for check processing in a bank. During the busy hours of bank, partition station could be put in place and for other hours, partition can be removed.

...

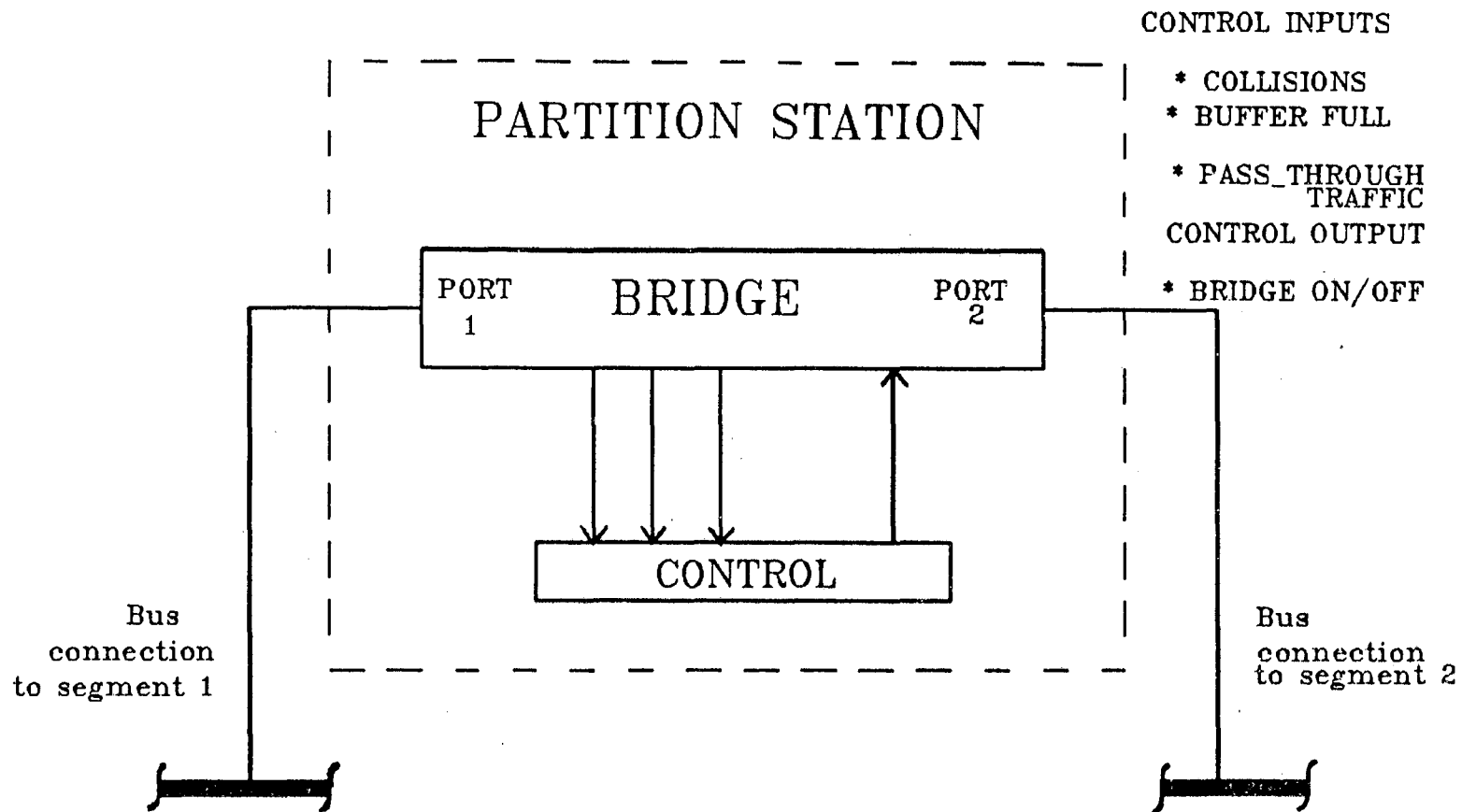


FIG. 2(a). COMPONENTS OF A PARTITION STATION

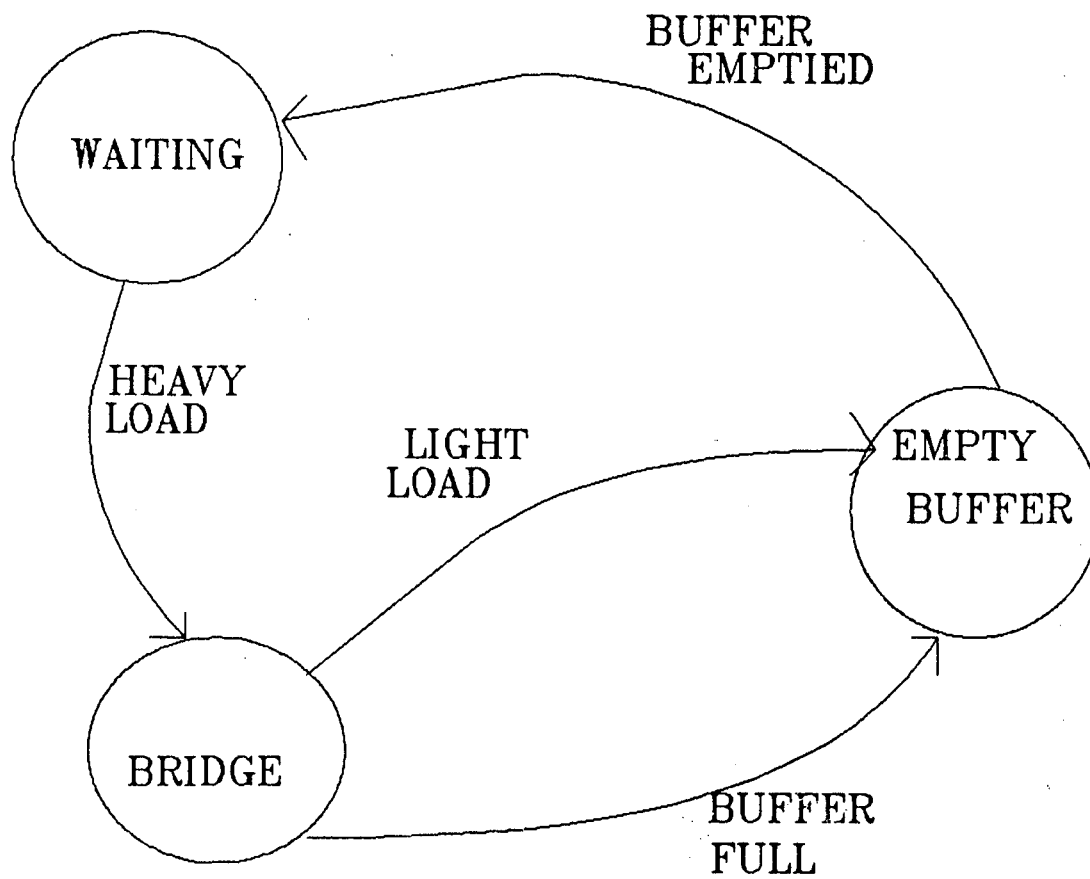


FIG. 2(b). STATE DIAGRAM OF PARTITION STATION

## CHAPTER 3

### OVERVIEW OF DISCRETE SIMULATION

#### 3.1 DEFINITIONS

A Model of a system is a collection of entities and their relationships. Entities are described by attributes. For example, customer in a queueing model is an entity, and customer is described by waiting time, service time, these are attributes. The description of values of all the attributes of an entity at a specified instant is called state of an entity. An event denotes the change in the state of the system entities. If the changes in the state of the system entities occur at discrete amounts of time then the system is called a discrete event system.

#### 3.2 DISCRETE EVENT SIMULATION TECHNIQUES

In Discrete Event Simulation (DES), the state of the simulated system is stored in a set of system state variables. Event routines cause state variables to be modified. An eventlist is used to control the execution sequence of these event routines; the list consists of events in chronological order. Event routines can add or delete items from the event list and pseudo random number generators in the event routines provide requisite randomness for modifying and scheduling of future events. Running a simulation is, in essence, the repeated execution of a loop, where at each iteration the most imminent event (the one with

the earliest schedule time) is executed in turn. A flow diagram for DES is as shown in Fig 3 (a).

There are three approaches for modelling a discrete event system. Namely, the event scheduling approach, the activity scanning approach and process interaction approach.

In the event scheduling approach, events are scheduled in advance. Whenever an event is scheduled, data identifying the type of the event and time at which to occur is placed in the special list called the event list. The simulation proceeds as follows:

- (1) The list of scheduled events is searched to find the event with the earliest scheduled time.
- (2) The simulation time is advanced to that earliest scheduled time.
- (3) State changes and scheduling of new events associated with the occurrence of the event occurred are performed.
- (4) Repeat the steps 1 to 4 again.

The activity scanning approach calls for modelling of the activities. An activity is a collection of operation that transform the state of an entity. Each system entity that changes state has a clock. The simulation progress as follows:

- (1) clocks of all system entities are checked to determine the time and type of next event.

- (2) The simulation time is advanced to the above time.
- (3) The step necessary with that event are performed.
- (4) Repeat steps (1) to (4).

The process interaction approach combines the event scheduling and activity scanning approaches. It maintains a list of future events but it also carries out, at each event time a scan of all activities and finds out which event can begin or end.

### **3.3 Why not special purpose simulation languages for the implementation of model:**

There are languages such as SIMSCRIPT, CSL and GPSS, which have been designed especially for the simulation. These languages are designed to optimize the features unique to simulation. These languages :

- (1) provide a convenient representation of elements that appear in simulation models.
- (2) generate automatically the pseudo random numbers for any statistical distribution.
- (3) facilitates collection of data and statistics of the simulated system.

Even though, these languages provide many features required for the simulation, they have not become popular. The main reasons for not using them widely are :

- (1) Users are not familiar with these languages.



- (2) These languages are not easily available to the users.
- (3) These languages are highly complex in dealing with the problem.

On the otherhand, highlevel languages such as PASCAL, C and C++ have the advantage of being familiar with users, easy availability, and the model can be chosen on the way you like where as special purpose simulation languages cannot. Therefore, general purpose high level languages are being used to simulate discrete event systems.

#### **3.4 MODEL VALIDATION AND ITS IMPORTANCE**

Model validation can be defined as the process of substantiating that the model is sufficiently accurate for the intended application. It is an important stage in a simulation experiment before simulation results can be accepted. The objective of the validation stage is to ensure that the simulation model is a proper representation of the system being studied. Since no simulation model exactly replicates the system under study in all particulars, the question of validation of model is a difficult one. However simulation studies are usually done with a model which represents the real system sufficiently for the purpose for which it is used. The efforts for validating a model can be grouped in to two parts. One is, validation of the abstract model it self and the other is the validation of the implementation. The validation of the abstract model is

1. Uniformly generated between 0 and 1.
2. Statistically independent.
3. Reproducible.
4. Non repeating for any required length.

### 3.5.1 Uniform random number generator

Several methods exist for generating the uniform random numbers. These methods are based on some recursive relation. Each new random number is generated from the previous value by applying some scrambling operation. Multiplicative congruential generator is one method for generating the random numbers. It consists of computing

$$X_{n+1} = (CX_n) \text{ mod } M$$

Where  $X_{n+1}$  is the (n+1)th random number,  $X_n$  is the previous random number and C is a constant multiplier. The value of  $X_0$  is called seed of the random number generator. With this generator, the maximum period  $2^{b-2}$  is obtained when

$$M = 2^b; \quad b > 4$$

$$c = 8K+5 \quad ; \quad K=0,1,2\dots$$

$$X_0 \text{ is odd}$$

The value of M is chosen to be equal to the largest prime number which is less than  $2^b$ . For a computer system the value m is equal to largest prime number that can be represented in it. If the multiplier C is a primitive root

modulo M, then the generator will have a maximal period of m-1.

### 3.5.2 Non uniform Random number generation

It is possible to generate random numbers for any distribution using the method of inverse transformation from the uniform random numbers.

If  $X_i$  is a sequence of random numbers which are uniformly distributed in the interval (0,1) and if Y has probability density function  $f(y)$  and cumulative distribution function  $F(y)$  then the sequence of random numbers  $Y_i$  are generated by the operation

$$Y_i = F^{-1}(X_i) \quad \text{----- 3.1}$$

For example, for an exponentially distributed sequence

$$f(y) = (1/\lambda) e^{-y/\lambda} \quad \text{where } \lambda \text{ is the arrival rate}$$

and  $F(y) = 1 - e^{-y/\lambda}$

From the inverse transformation method, if  $x_i$  's are the uniform random numbers then the exponentially distributed random sequence  $y_i$  are generated by  $y_i = -\lambda \log (1-x_i)$

$$Y_i = -\lambda \log (x_i) \quad \text{----- 3.2}$$

Thus equation (3.2) will generate the exponentially distributed random numbers.

### 3.6 SIMULATION OUTPUT ANALYSIS

In the simulation process the question which comes into mind is how long should the simulation run? Hence the approaches to estimating and controlling simulation output accuracy is needed. The subject which deals with this is called simulation output analysis.

If the simulation run length is determined by the problem itself, then this type of simulation is called terminating or transient simulation. For this type of simulation, the question becomes how many times the simulation must be repeated to achieve a specified accuracy.

In the steady state simulation, both the initial conditions and the length of the simulation are determined by the modeler, and the measure of the interest is the limiting value reached as the length of simulation run goes to infinity. Practically, run lengths are finite and our problem is to determine how close the mean estimated from the sample values is to the true mean of the distribution.

Most simulations in the computer system design environment are steady state simulations.

The performance measure of interest is the mean (average) value of simulation output variable.

For a discrete time random process, the simulation



TH-5601

produces a sequence of  $n$  sample values  $X_1, x_2 \dots x_n$  whose mean (denoted by  $\bar{X}$ ) is

$$\bar{X} = \sum_{i=1}^n x_i/n$$

As  $n$  approaches infinity,  $\bar{X}$  converges to a limiting value  $E[X]$  called expectation of  $X$ . Call  $E(x)$  as distribution mean ( $\mu$ ).

The variance is a measure of dispersion of a distribution. The variance of a set of values  $x_1, x_2 \dots, x_n$

$$s^2 = \sum_{i=1}^n (x_i - \bar{x})^2 / (n-1)$$

$s^2$  is called sample variance.

As  $n$  approaches infinity,  $s^2$  converges to limiting value  $E[s^2] = E[(x-\mu)^2]$  denoted by  $\sigma^2$ .

The square root of variance is standard deviation.

### 3.6.1 Confidence interval:

How close the sample mean obtained from finite length simulation is to the distribution mean  $\mu$  or how long run lengths have to be to obtain a sample mean arbitrarily close to  $\mu$  is to be estimated. This is the problem. The answer to this can be found by computing a measure called confidence interval.

Suppose we have a set of  $N$  sample values  $Y_1, Y_2, \dots, Y_n$  from a distribution with true (but unknown) mean  $\mu$ .

$$\text{The sample mean is } \bar{Y} = \frac{\sum_{i=1}^n y_i}{n}$$

By defining  $1-\alpha$  as the probability that the absolute value of the difference between sample mean and  $\mu$  is equal to or less than  $H$ .

$$P[ |\bar{Y}-\mu| \leq H ] = 1-\alpha$$

Then a confidence interval for the mean is defined as

$$P[ \bar{Y}-H \leq \mu \leq \bar{Y}+H ] = 1-\alpha$$

The interval  $\bar{Y}-H$  to  $\bar{Y}+H$  is called the confidence interval.  $H$  is called the confidence interval half width, and  $1-\alpha$  is called the confidence level or confidence coefficient. Typical values of which are 0.90 or 0.95. The confidence level  $1-\alpha$  is specified by the analyst.  $H$  then is determined by the sample values, number of samples, and the value of  $\alpha$ .

$H$  is a function of random variables and so is a random variable itself.

When  $Y_1, Y_2, \dots, Y_n$  are independent random variables from a normal distribution with mean  $\mu$ ,  $H$  is given by

$$H = t_{\alpha/2; n-1} s/n^{1/2}$$

where  $t_{\alpha/2; (n-1)}$  is the upper  $\alpha/2$  quantile of a distribution

with (n-1) degree of freedom and  $s^2$  is the sample variance

$$s^2 = \sum_{i=1}^n (y_i - \bar{y})^2 / (n-1)$$

Estimation of a confidence interval is what the simulation output analysis is all about.

A number of methods for estimating a confidence interval for the mean of simulation output variable have been described. These include methods called

1) Replication (2) Batch means (3) Regeneration (4) Auto regression (5) spectral analysis (6) standardized time series.

Approaches to confidence interval estimation can be classified as fixed sample size procedures or as sequential procedures. In a fixed sample size procedure a simulation experiment of total fixed length is performed and the confidence interval is estimated from the results of the experiment upon its completion. In a sequential procedure, the desired accuracy is specified, confidence interval estimates are computed at selected intervals and the experiment is continued until the desired accuracy is obtained.

In the batch means analysis divide one long run in to a set of K subruns of length m, called batches, computing a separate sample mean for each batch, and using these batch means to compute the grand mean and confidence interval.

Assuming deletion is used to achieve steady state initial conditions for the first batch, each subsequent batch begins with the system in the steady state. Since warmup effects have to be dealt with only once, rather than  $K$  times as in the case of replication, the batch means method is potentially more efficient i.e., fewer sample values are needed to achieve a given accuracy.

...



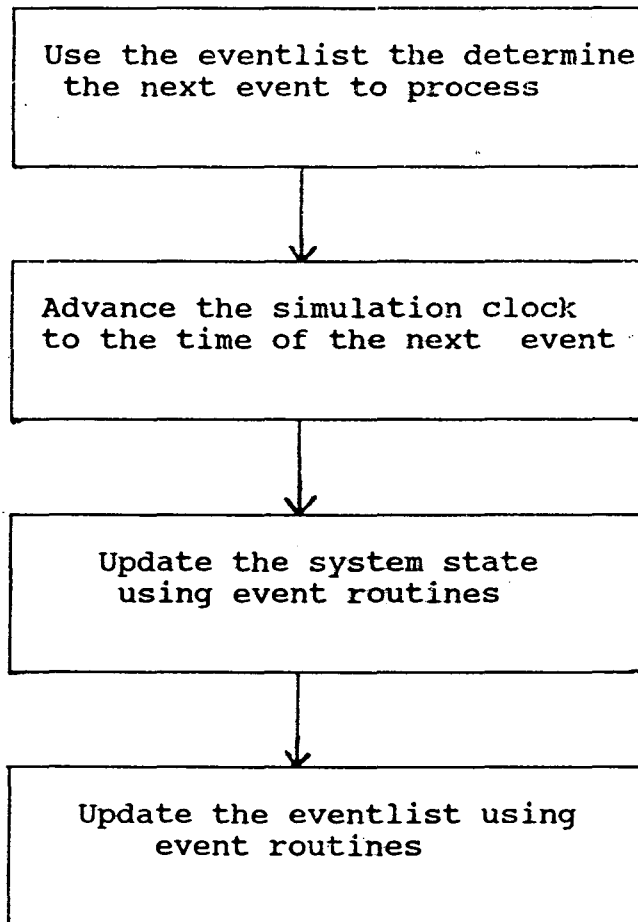


Fig 3(a) DES flow design

## CHAPTER 4

### MODELLING

#### 4.1 ETHERNET SIMULATION MODEL

The effect of Network partitioning on network performance, was shown by modelling the Ethernet and by simulating it.

The protocol used in Ethernet is 1-persistent CSMA/CD. The simulation was written based on the 1-persistent CSMA/CD protocol and the Ethernet specification. The model was developed using the C++ language.

In the Ethernet model, the network is assumed to be composed of a single bus and a set of stations on that bus. Each station has independent packet arrival and queue. Packets are transmitted and propagated through the bus to the both the sides. A station will attempt to transmit the first packet in the queue before attempting the next packet which is present in the queue.

It is assumed that the distance between stations is given. The model is developed by using the discrete event simulation. An event list has been formed by taking different events, which occur in the model. An event list is a list of nodes arranged in ascending order, with each node having the data relating to the type of the event, the time at which the event has occurred and the pointer to the next node in the

list. The type of the next event to occur is the type of the event in node to which the event list is currently pointed to and time at which it occurs is the event time stored in that particular node.

The packet arrival events at each station other than the partition station are generated from the exponentially distributed inter arrival times. Whenever an arrival occurs packets present count is incremented and its arrival time is stored in a buffer. If the buffer is full, then the packet is discarded. From the exponential distribution a random inter arrival time is generated by using the arrival rate of packets at a station, and the next packet arrival is scheduled at a time equal to the sum of the current time and randomly generated inter arrival time.

When a node has acquired the channel, and there is going to be no collision then the time to transmit a packet of length  $L$  is calculated and the packet transmission completion event is scheduled after that much time i.e. current time plus packet transmission time.

The eventlist of CSMA/CD consists of the following events.

- (1) channel idle at a upstream node event.
- (2) channel idle at a downstream node event.
- (3) collision backoff period completion events for the nodes which are backlogged. (A node is called backlogged if its

packet transmission attempt had resulted in a collision and it has not succeeded in transmitting that packet yet)

(4) Next packet arrival event for each node.

(5) Packet transmission completion event.

The channel idle events i.e. the events (1) and (2) are scheduled each time after a collision and a successful transmission. Whenever the events(1) or (2) occurs channel status as seen by the upstream node or downstream node is set to idle.If that node does not attempt to acquire the channel, then the channel idle events for the next upstream or downstream node is scheduled. When an arrival event occurs then the next arrival event for that node is generated and is scheduled in the eventlist and if that node is not backlogged and the channel status as seen by that node is idle, then attempt to acquire the channel. A node will attempt to acquire the channel in the following conditions :

(1) It has just sensed the channel idle i.e the channel status seen by the node is idle and if any of the following two is satisfied.

(a) It is not back logged and it has one or more packets to transmit in its queue.

(b) It is backlogged and its backlog completion event has completed already.

(2) If the channel is sensed idle, i.e. the channel status seen by the node is idle and an arrival has just taken place.

After a node tries to acquire the channel, then the question comes is whether the channel acquisition attempt will result in a successful transmission or a collision. This is simulated as follows:

Whenever a node  $i$  attempts to acquire the channel for its packet transmission, then the list of scheduled events associated with each node other than the node, which is currently acquiring the channel i.e. node  $i$ , and whether acquiring node is backlogged or not and number of packets present in it are checked to find the possibility of a collision. Let us define a collision window between node  $i$ , which is attempting to transmit and any other node  $j$  as the current simulation time plus the propagation delay between the two nodes.

A collision will result under the following conditions:

- (1) If any other node also simultaneously attempt to acquire the channel.
- (2) If any other node  $j$  will have a packet arrival event within the collision window and the node  $j$  is not backlogged.
- (3) If any other node  $j$  which is backlogged and whose collision backoff period will be completed within the collision window.

If the exhaustive checking at every node shows that the packet transmission attempt by the node  $i$  will result in a collision than the farthest upstream and downstream nodes

participating in the collision are found out. The collision detection time and subsequent transmission stop times for these nodes are calculated. From the transmission stop times at the downstream node and upstream node participating in the collisions, the time when the channel will again start appearing idle is calculated and channel idle events, i.e. events (1) and (2) are scheduled in the event list. For all those nodes, which have transmitted collided packets the backoff periods are generated using the binary exponential backoff algorithm and these times are scheduled in the event list.

In case, if no collision occurs after the exhaustive checking at every other node other than the node, which is currently acquired channel to transmit, then packet transmission completion event is scheduled in the event list.

When the packet transmission completion event occurs then the channel idle events are scheduled after a delay equal to interframe gap and the count of the number of successfully transmitted packets is incremented.

The backoff algorithm used in case of collision is as follows :

Backoff Algorithm : The algorithm used in delaying the retransmission attempts of the stations after the involvement of them in a collision is called truncated binary exponential backoff. The amount of delay due to backoff is calculated as

follows.

1. Retransmission attempt count is increased by one.
2. If the number of retransmission attempts is more than 16, then discard the packet.
3. Compute the minimum of (retransmission attempt, 10) say it  $k$ .
4. Generate random integer  $r$  in the range  $[0, 2^k - 1]$ .
5. Set the amount of backoff delay to  $(r * \text{slottime})$  i.e. product of ' $r$ ' and ' $\text{slottime}$ '.

After first collision of a packet, retransmission is done after a backoff delay of 0 or 1 slot times, and after the second collision of a packet, retransmission is after a delay of from 0 to 3 slot times, up to a delay of from 0 to 1023 slottimes for attempt to 10-16. After 16 unsuccessful attempts, discard the packet.

If the channel tends to become overloaded, then backoff algorithm stabilizes it. As the load on the channel increases, the collisions increases. Collision in effect increases the backoff delay, reducing the load on the channel.

By assuming that during the partition, each partition segment is independent of the other, the partitioned network is simulated. The traffic to each segment from the other segment is assumed to be the offered load to the partition

station. By assuming the offered load of the partition station to be equal to the passthrough traffic and then simulating the network is equivalent to simulating the partitioned network.

In the simulated two segment ethernet model developed, the length of segment was set at 500 meters. The packet size be 1000 bits per each packet and number of station to the 10.

Two segment Ethernet was shown as in Fig 4.1 this has got 20 stations and a partition station, separating 10 stations in to segment on each side. The model, which was described above was used to simulate the two segment Ethernet. The network is simulated initially without network partitioning and next with network partitioning and 0%, 50% and 100% pass-through traffic.

The transition from the partitioned network to the nonpartitioned network or vice versa is simulated as follows:

When the number of collisions per second exceeds a certain value, the event list is broken in to two event lists, and two networks are run independent of each other. For each packet, determine the destination. If the destination is on the other side of the partition station, then store the packet in the partition station. and also try to check the number of collisions per second on both the sides of partition station. If the sum of these falls below a



certain value, then join the two event lists in to the single event list. Try to transmit the packets in the partition station by truncating the back off algorithm to zero so that it is always the first station to transmit after a collision. The conversion from partition to nonpartition is done, even if the buffer of the partition station is full.

#### 4.2 MODELLING ASSUMPTIONS

The following assumptions are used in the modelling :

- (1) Medium of transmission (channel) is completely error free.
- (2) The processing time for the packets at the stations in the network is negligible.
- (3) There is a limit on the buffer size of the stations.
- (4) The packet arrival at the stations except at the partition station are assumed to be poisson.
- (5) The inter arrival times of packets at the stations is assumed to be exponentially distributed.
- (6) The length of the packets is assumed to be fixed.
- (7) For the packets at the stations, the first come first served basis is assumed.
- (8) The partition station is assumed to be a two way. The operation of partition station in each direction has been assumed to be independent of operation in the other direction.
- (9) The processing time for packets at the partition station in the network is negligible.

### 4.3 INPUT AND OUTPUT PARAMETERS OF MODEL

#### 4.3.1 Input parameters to the simulation model :

- (1) Number of stations on the network.
- (2) Specify the location of the partition station.
- (3) Mean inter arrival rate of packets : The average with which packets arrive at the channel.
- (4) The distance between the stations on the network in meters.
- (5) Packet size : Length of the frame in bits.
- (6) Channel speed : Rate at which data is transferred on channel and is given in Mbps.
- (7) Jam time : When a station recognizes that a collision has occurred, it transmits a jam signal by immediately abandoning, its transmission of data to insure that all other stations on the network know that collision has occurred. The jam is specified to be from 32 to 48 bit times in length; assuming the maximum.  $T_{jam} = 4.8$  microseconds for the Ethernet.
- (8) Interframe spacing : The delay between the time at which a station recognises that the channel is free and the time at which that station initiates a transmission. This gap between the two transmissions makes sure that the receiving station has time to prepare for a new transmission. From Ethernet this delay is specified as 9.6 microseconds.
- (9) Truncation of binary exponential backoff algorithm.
- (10) Slottime : It is rescheduling time used by the stations in backing off after a collision has occurred. For the

Ethernet, in is specified to be 512 bit times or 51.2 micro seconds.

#### **4.3.2. Output parameters from the simulation model**

1. Offered load : The offered load on the network is the sum of individual loads of all stations connected to the network.
2. Average packet delay : It is the difference between the time at which a station first attempt to transmit a frame to the time at which transmission of frame is completed.
3. Throughput : The ratio of number of frames which are successfully transmitted to the number of frames which are generated and a given in Mbits per sec.
4. Number of collisions per second : Total number of collisions / simulation period.

#### **4.4 VALIDATION OF SIMULATION MODEL**

In order to validate a simulation model a network with a known throughput must be simulated. The network chosen was Schoch's Experimental Ethernet [6].

##### **4.4.1 Schoch's Experiemental Ethernet:**

The experiemental Ethernet system, uses a coaxial cable running at 2.94 Mbps. The length of the cable is 550 meters and connects over 120 machines.

This uses the following parameters:

- (1) Bus bandwidth 2.95 Mbps.
- (2) The bus length is 550 meters with one way propagation delay of 2.75 micro seconds.
- (3) The slot time or round propagation delay is 5.5 micro seconds.
- (4) The system uses the binary exponential backoff algorithm.

#### 4.4.2 Performance under heavy load as found by Schoch [6].

In an idle case, the total channel utilization increases with the total offered load up to 100 percent. Beyond 100 percent load-under very heavy load the channel utilization remain at 100 percent, representing full use of available capacity.

Real systems can not perform this way. A pure Aloha channel gets only 18 percent maximum channel utilization and slotted aloha 37 percent. But for Ethernet system as found by Schoch as the total offered load increases from 0.20 percent channel utilisation matches is perfectly; all of the traffic gets out correctly. As the offered load moves above 90 percent, channel utilisation flatters out of a level above 96 percent for full size data packets (512 bytes). Ethernet system under light load shows no instability. The throughput curve doesnot decline as total offered load increases.

By assuming the offered load to be equally divided among all stations, the measurements of Schoch [6] give us the following important points:

(1) Under normal load, transmitting stations rarely have to defer and there are very few collisions. Thus the access time for any station attempting to transmit is virtually zero.

(2) Under heavy load there are more collisions, but the collision detection and resolution mechanism work well, and channel utilization remains very high approaching 98 percent. In addition, the utilization remains very stable.

(3) Even under extreme overload, the Ethernet channel does not become unstable.

...

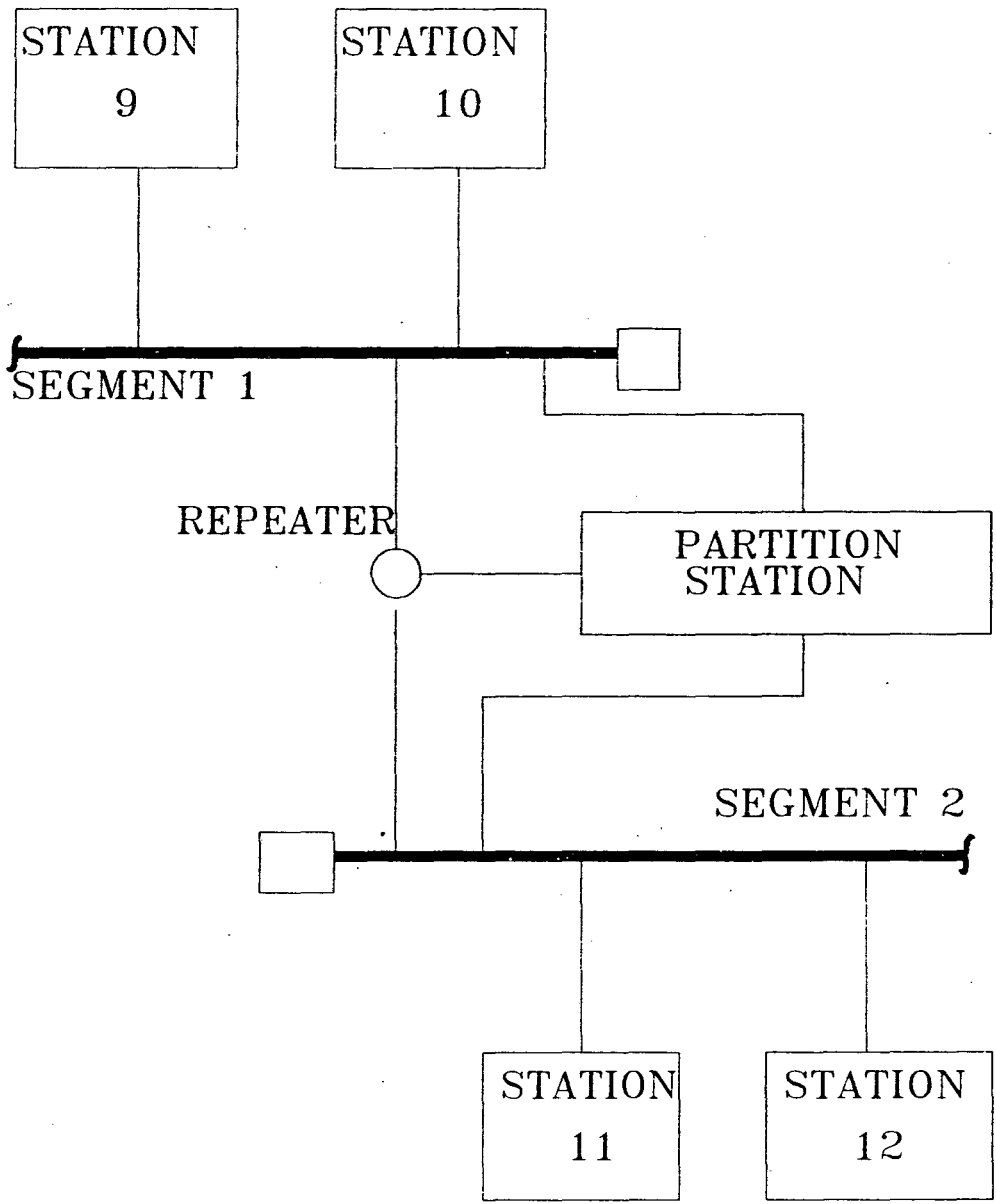


FIG. 4.1 TWO SEGMENT ETHERNET CONFIGURATION WITH A PARTITION STATION BETWEEN SEGMENTS

## CHAPTER 5

### PROGRAMMING APPROACH FOR THE MODEL

#### 5.1 DEVELOPMENT OF ETHERNET MODEL

The major steps in the development of the model constitutes :

- (1) Generation of the arrival times for the packets.
- (2) Preparation of the eventlist.
- (3) Depending on the topevent of the event list, performing the function associated with that event.
- (4) Trying to acquire the channel in case the channel is idle.
- (5) Determining whether a collision occurs or not, after a station tried to acquire channel.
- (6) In case a collision occurs, find the transmission stop times for the farthest upstream and farthest downstream nodes participating in collision.
- (7) Developing binary exponential backoff algorithm to back off the stations, which are colliding.
- (8) Developing discard algorithm in case the number of retransmission attempts are more than 16.
- (9) Developing an algorithm to determine the statistics.
- (10) Employing the batchmeans analysis to determine the confidence intervals.

### 5.1.1 Generation of Arrival times:

In the C language libraries, there is a built\_in function `rand()` which generates the random integers between the 0 and `RAND-MAX`. By using this function, a function generating a random number between 0 and 1 is generated.

Arrivals at a station are poisson and inter arrival times are exponentially distributed. From the exponential distribution with the specification of arrival rate inter arrival times are found.

### 5.1.2 Representation of Eventlist:

Eventlist is defined as an object, which is linked list. In the linked list, each node is defined to be having three fields, type of the event, time at which the event has occurred and the pointer to the next. The event list structure is defined as follows:

EVENT TYPE (K)	INDICATION
0	Channel idle at upstream node
1	Channel idle at downstream node
$2 \leq K \leq \text{nos}+1$	Arrival at station [K-1]
$\text{nos}+2 \leq K \leq 2*\text{nos}+1$	Backlog completion event of station [K-nos-2]
$2*\text{nos}+2$	packet completion event

(nos indicates the number of stations)



As shown in the event list structure, the type of the event indicates whether the event is channel idle or arrival or backlog completion or packet completion event. For each type of the event, a separate module is required to be run. In case the top event type in the event list is channel idle at upstream node then, perform the following steps:

- (1) If that station is not backlogged and if it contains one or more packets in its queue try to acquire the channel.
- (2) Otherwise, if backlogged and its backlog completion event completes then try to acquire the channel.
- (3) If not (1) and (2) then generate a channel idle event at the next upstream node and keep it in event list.

If the top event type the event list is channel idle at downstream node then perform :

- (1) If that downstream station is not backlogged and if it has got one or more packets in its queue then try to acquire the channel. Otherwise,
- (2) If that station is backlogged and its backlog completion event ends already, then also try to acquire the channel. Otherwise,
- (3) Generate a channel idle event for the next downstream node and add it to the event list.

If the backlog completion event at a particular station is the top event in the event list, then see whether the channel status as seen by that station is idle or not. If it

is idle, then try to acquire channel.

If arrival at a particular station, say  $i$ , is the event type occurred then keep this arrival in the process queue of that station, generate the next arrival and finally, if that station is not backlogged and the channel status as seen by that station is idle, then try to acquire the channel.

If the event is the packet completion type, then increment the number of successfully transmitted packets, update the statistics and generate the channel idle event at time equal to current simulation clock plus the interframe gap at the successfully transmitted station by making this station the current upstream node. Make the current downstream node to be the next of current upstream node, in case it is not the final station, and set the channel idle event at this station also.

### **5.1.3 Actions after channel acquisition attempt:**

After trying to acquiring the channel by a station, it sees to that no other station will send any packet within collision window, otherwise the collision will occur, it is implemented as follows:

1. Remove the channel idle events in case if they exists in the event list since the channel is going to be busy if any station is trying to acquire the channel.
2. Determine the extreme nodes from the acquiring station,

participated in collision and the transmission stop times at these nodes.

3. If the number of transmitting stations are more than one, then find for all the nodes which are participated in collision, the backoff completion times and set these times in the event list.

(4) If the number of transmitting stations is one, then set the packet completion event at a time equal to the current simulation time plus packet transmission time in the event list.

#### 5.1.4 Detection of a collision:

Cowindow is defined as the current simulation clock value plus the propagation delay between node  $i$ , which attempts to transmit and any other node  $j$ . For every other node  $j$  the following checks are made to find whether it participates in collision or not.

1. If the node  $j$  is backlogged and its backlog completion event ends within the cowindow.

2. If the node  $j$  is not backlogged and if it has got one or more packets in its queue.

3. If the node  $j$  is not backlogged and if it has any arrival within the cowindow.

if (1) or (2) or (3) happens, then the collision flag is set to one, which means collision has occurred.

### **5.1.5 Calculating and Displaying the statistics:**

Whenever an arrival event has occurred, that packet arrival event time is stored in the queue of the station at which the arrival has occurred. This arrival time is useful for finding the delay experienced by that packet. After that packet is successfully transmitted, the difference between the packet completion time and the packet arrival time is fed to the procedure `obs()`. This function does the batch means analysis of the delay. This determines the average packet delay experienced in the network and the confidence intervals within which this average packet delay remains. After getting the average packet delay, the normalized packet delay is found by dividing it with the packet transmission time.

### **5.2 IMPLEMENTATION OF MODEL WITH NETWORK PARTITIONING**

In case of partitioned network, the passthrough traffic to each segment is offered load to the partition stations on that segment. If the partition flag is 1, which indicates that the network is partitioned, then determine the offered load from the value of passthrough traffic given. This offered load gives the value of  $\lambda$  at the partition station. With this  $\lambda$ , generate the arrivals at the partition station and run the simulation.

To implement the transient behaviour of network partitioning, the transition from a partitioned network in to

a non partitioned network, generate the two simulation clocks and run both the segments independently. Each time find the sum of number of collisions per second on both the sides and if this goes below a value found from the number of collisions vs.offered load curve obtained from the simulation nonpartitioned network, then join the simulation clocks to a single simulation clock. Even if the buffer of partition station is filled, then also go to nonpartition network. Now, transmit the packets in the partition station with a backoff of zero and until the partition station buffer is empty. Now, if in the non-partitioned network the number of collisions per second is greater than value found from the curve of Ethernet Simulation, then partition. This way the network transforms between a partitioned network to nonpartitioned network and vice versa.

### 5.3 MAJOR MODULES DEFINED IN PROGRAM

#### 5.3.1 CLASSES

To maintain the value of the current simulation clock, a class called float\_counter is defined.

```
Class float_counter
{
private      : double value;
public      :
              float_counter();
              void increment (double x);
              double get (void);
```

```
void set (double x);  
  
};
```

The object of this class is defined by the statement  
float\_counter simclk;

When the simclk object is created, the constructor float\_counter causes the value to be initialized to zero. The member function increment() is used to increment the value of the private variable by x and get() function returns the value and set(double x) function sets the variable value to the specified value x.

The channel status as seen by each node is maintained in a bus class.

```
class bus  
{  
private      : int flag [nos];  
public      :  
              bus();  
              void makeidle (int i);  
              void makebusy (int i);  
              int getstatus (int i);  
};
```

The constructor bus() initializes of the flags of all stations to idle, when an object of class bus is created. The

member function makeidle(int i) sets the flag of the station i to idle and makebusy(int i) sets the flag of station i to busy. The channel status as seen by the node i is returned by the member function getstatus(int i).

The eventlist is defined as a class and is as follows :

```
struct link
{
    double time;
    int type;
    link * next;
};
class linklist
{
private      : link * first;
public      :
            linklist ();
            double gettime ();
            double gettime (int t);
            double gettype ();
            void assendadd (double d, int t);
            void remove ();
            void remove (int t);
            void display ();
};
```

The object of class linklist is created with the statement,

```
linklist evlist;
```

When the object evlist is created, thus constructor link list() assigns the first to NULL.

The member function ascendadd(double d, int t) is used to add a node having the event time equal to d and event type t to the event list and makes the eventlist in the ascending order, with first always pointing to the node with a earliest event time.

The member function gettime() returns the event time of the next imminent event, which is going to occur, and the member function gettime(int t) return the event time of node for which the type of the event is t.

The function gettype() returns the type of next imminent event and the remove functions are used to remove an event from the event list. Remove() removes the top event in the event list and remove(int t) removes the event for which type of the event is t. Display() function is used to display the event list.

### 5.3.2 FUNCTIONS

FRAND : Generates a floating point random number between [0,1).

ARRIVAL\_TIME : Generates inter arrival times.



SET\_PARAMETERS : Take input parameters, takes with Ethernet's standard parameters or Schoch's experimental Ethernet parameters or user specified parameters.

SETSTANDARD1 : Sets the 802.3 standard Ethernet parameters.

SETSTANDARD2 : Sets the schoch's experiemental Ethernet parameters

START\_SIMULATE : Intialize the buffers in the queue for each station to empty and generate a single arrival for each station and place all these arrivals in the event list.

CSMACD : This is the main procedure which calls other functions depending on the top event type in the event list.

CHANNEL\_ACQUIRE : If a station is ready to transmit, checks whether it can do it successfully, or not. If it can transmit successfully, call function to schedule transmission of a packet else call CHANNELIDLE to schedule retransmission intervals of colliding stations and channel idle events after collision.

COLLISION\_DETECT : It checks whenever a station j can also attempt to acquire the channel before transmission is heard by station j. In case of collision, it updates the number of backoffs and number of attempts.

UPDATE\_STAT : This function used to update the statistics. This finds the differences between a packet completion time and that packet arrival time and with this

difference it calls the OBS(double) function.

INIT\_BM(int x,int y): This is used to initialize the batch means analysis. The values as specified by x are deleted to reduce the warm\_up effects and the value of y specifies the batch size.

OBS (double) : It computes the confidence interval and the average packet delay. At the end of the 10 batches, the program return 1, which is used to stop the simulation run.

### 5.3.3. GLOBAL VARIABLES

Par\_flag : Partition flag, this is set to 1 in the case of partition, otherwise it is set to 0.

Lambda : Arrival rate per second per station.

Flag\_backlog[i] : It is set to 1 if the station i is backlogged otherwise, it is set to 0.

Retx\_flag[i] : It is set to retransmitted otherwise to 0.

Pkttxttime : Packet transmission time, and is equal to packetsize / transmission speed.

Overfpkts : Gives the number of overflowed packets.

Atbuff[nos][BUFFSIZE]: This is used to store all the packet arrival times for those packets which are in queue.

Topevtime : Gives the topevent time of the event list.

Packets : Total number of packets which are generated.

...

## CHAPTER 6

### CONCLUSION

This chapter provides some of the sample results produced by running the simulation program to demonstrate the effect of network partitioning on the CSMA/CD LAN. The simulation of 1-persistent CSMA/CD LAN is done and its results are compared with the analytical results produced by [5]. Almost all the results have been obtained after simulating more than 2000 packets on an average per each node. The effects of transients on the mean is taken in to consideration by leaving the initial 2000 packets generated and the batchmeans analysis is used with a batchsize of 2000. It is assumed that there is no correlation among successive output sample while generating confidence intervals.

#### 6.1 VALIDATION OF 1-PERSISTENT SIMULATION MODEL

The utilization is found for the model developed for a packet size of 512 bytes and the results are compared with the Schoch's results[6] and are shown in table 6.1.

The performance characteristic of average packet delay Vs the offeredload of the CSMA/CD network with the 20 nodes operating at 1Mbps is as shown in the figure 6.1. The average packet delays, which are obtained from the approximate analytical calculations are also shown in the same Figure. From this figure it is clear that the delay

performance obtained from the simulation results are better than those found by the approximate analytical calculations. The results are given table 6.2 .

In order to show the effect that the confidence interval reduces as the length of the simulation run increases, the graph showing the upper and lower confidence limits is drawn with respect to number of packets simulated. It is shown in Fig. 6.2 and the results in table 6.3 .

To demonstrate the effect of initial transients, the different values of the average packet delay are found after discarding the different percentage number of initial observations. At the time of finding these values the simulation length is fixed at 2000 packets per node. This was done at the offered loads of 0.1 and 0.5, and the results are given in table 6.4. From the table it was found that the transients have effect at high offered load values. To reduce this effect at higher offered load, the longer simulation length should be chosen.

## **6.2 SIMULATION OF TWO SEGMENT ETHERNET WITH NETWORK PARTITIONING**

Figure 6.3 shows the results of the simulation with the network partitioning assuming that the pass through traffic to each segment is offered load to the partition station and each segment is independent after partitioning. For three different passthrough traffics of 0%, 50% and 100% and its

comparison without the network partitioning for the network having the 20 nodes and the for a packet size of 1000 bits. From this figure, it is evident that without network partitioning the network is saturated at 0.85 Megabits per second offered load and with network partitioning the maximum throughput achievable is 1.7 Megabits per second for no passthrough traffic through the partition station. Since the two segments act independently after partition, the throughput has to be double. The results have shown this, and also the results show that for the 50% pass through traffic with network partitioning the maximum throughput achievable is 1.1 Megabits per second. From Figure 6.3, it is known that after the throughput crosses the 0.7 Mbps, it is better to keep the partition in place. During the simulation run of the Ethernet without network partitioning, the number of collisions per second are found for each offered load, and these results are produced in figure 6.4. From this figure, the number of collisions per second for throughput of 0.7 Mbps is found and it is kept as a limit for keeping the partition station. From the results, this limit is found to be 1500 collisions per second. So when the number of collisions per second exceeds this limit, the partition is placed, and when it is below this value, partition is removed. By keeping these collisions per second as the factor for the transition between partitioned network and nonpartitioned network and vice versa, the simulation model

is run and the results are as shown in Figure 6.5. The results compare closely with the results which are shown in Figure 6.3.

### 6.3 CONCLUSION

The effect of network partitioning on the CSMA/CD LAN was shown by simulating the two segment Ethernet network. The 1-persistent CSMA/CD protocol was developed and was validated with the standard results. The event scheduling was used to model the protocol. The model was then modified to show the effect of network partitioning. The simulation results with and without network partitioning gave the conclusion that the performance of CSMA/CD LAN improves under heavy load and improvement factor depends on the pass through traffic through the partition station.

...

OFFERED LOAD	SCHOCH'S UTILISATION	SIMULATION RESULTS
70	70	70
80	80	80
90	90	90
100	94	95.6
120	96	97
150	96	97.2

TABLE 6.1 Comparison of simulation results with the Schoch's results for a packet size of 512 bytes



Transmission Speed = 1 Mbps  
Stations = 20  
Packet size = 1000 bits (constant )

---

Offered Load	Normalized Delay	
	Simulation	Analytical
0.05	1.032	1.03
0.1	1.07	1.072
0.15	1.09	1.088
0.2	1.145	1.142
0.25	1.219	1.218
0.30	1.272	1.272
0.35	1.364	1.365
0.40	1.464	1.464
0.45	1.602	1.602
0.5	1.706	1.708
0.55	1.856	1.859
0.60	2.18	2.231
0.65	2.4	2.534
0.7	3.08	3.281
0.75	3.88	4.01
0.80	4.64	4.96
0.85	8.44	8.97
0.90	20.95	22.76

---

TABLE 6.2

---

Number of packets	Mean Delay	Upper Confidence Limit	Lower Confidence Limit
500	2.2	3.2	1.2
1000	2.13	3.03	1.23
2000	1.99	2.69	1.29
3000	1.92	2.53	1.31
4000	1.9	2.44	1.36
5000	1.88	2.27	1.49
6000	1.86	2.11	1.61
8000	1.85	2.07	1.63

---

TABLE 6.3 Confidence intervals

Stations = 20

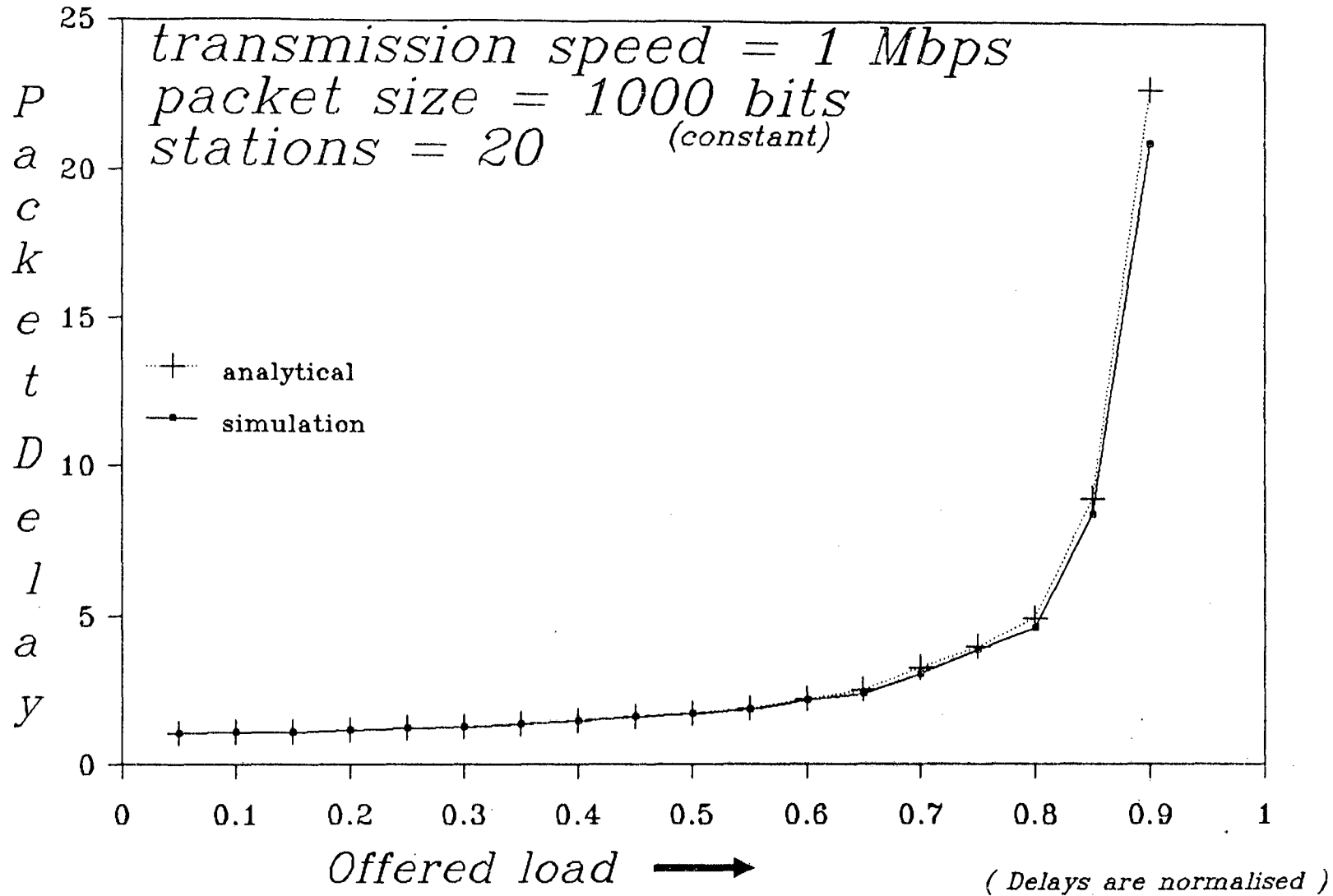
Transmission speed = 1 Mbps

Packet size = 1000 bits

<u>Offered load</u> ↓ %of discarded packets →	Packet Delay ( in micro sec.)			
	0%	1%	5%	10%
0.1	107.23	107.275	107.34	107.39
0.5	165.46	175.32	177.89	170.60

TABLE 6.4 Effects of transients on CSMA/CD

Fig 6.1 Offered load vs. Average packet delay



(Delays are normalized)

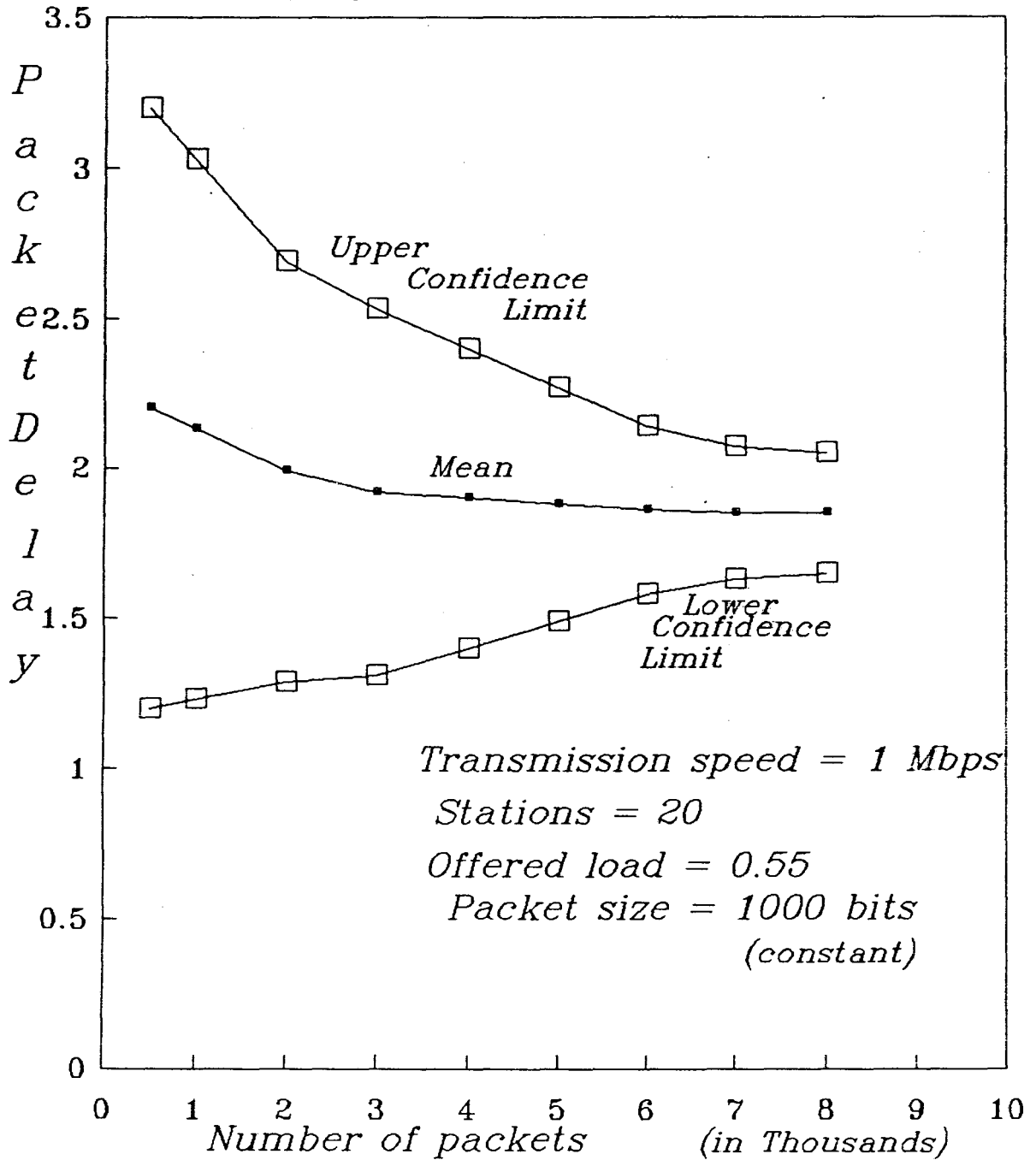
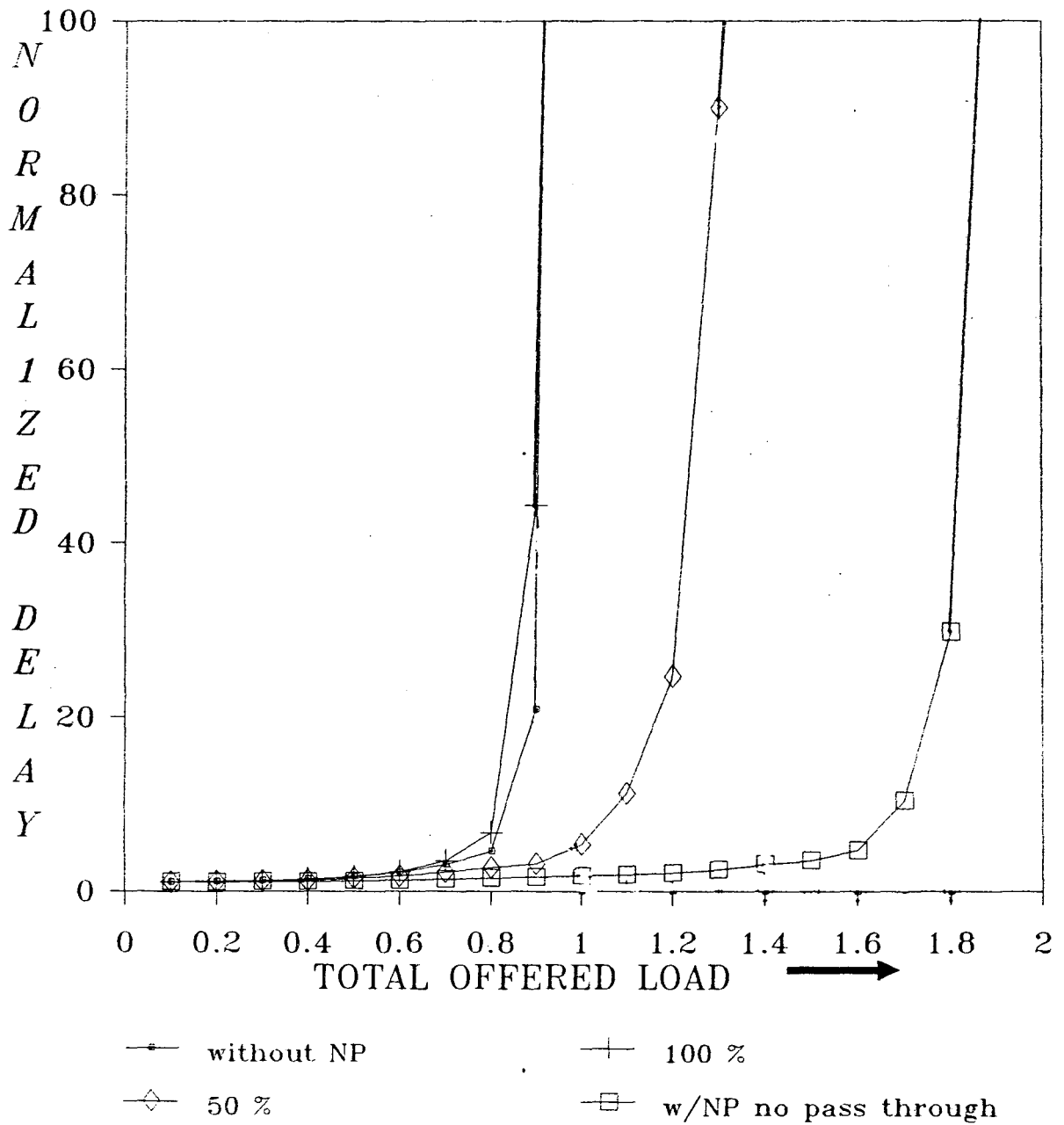
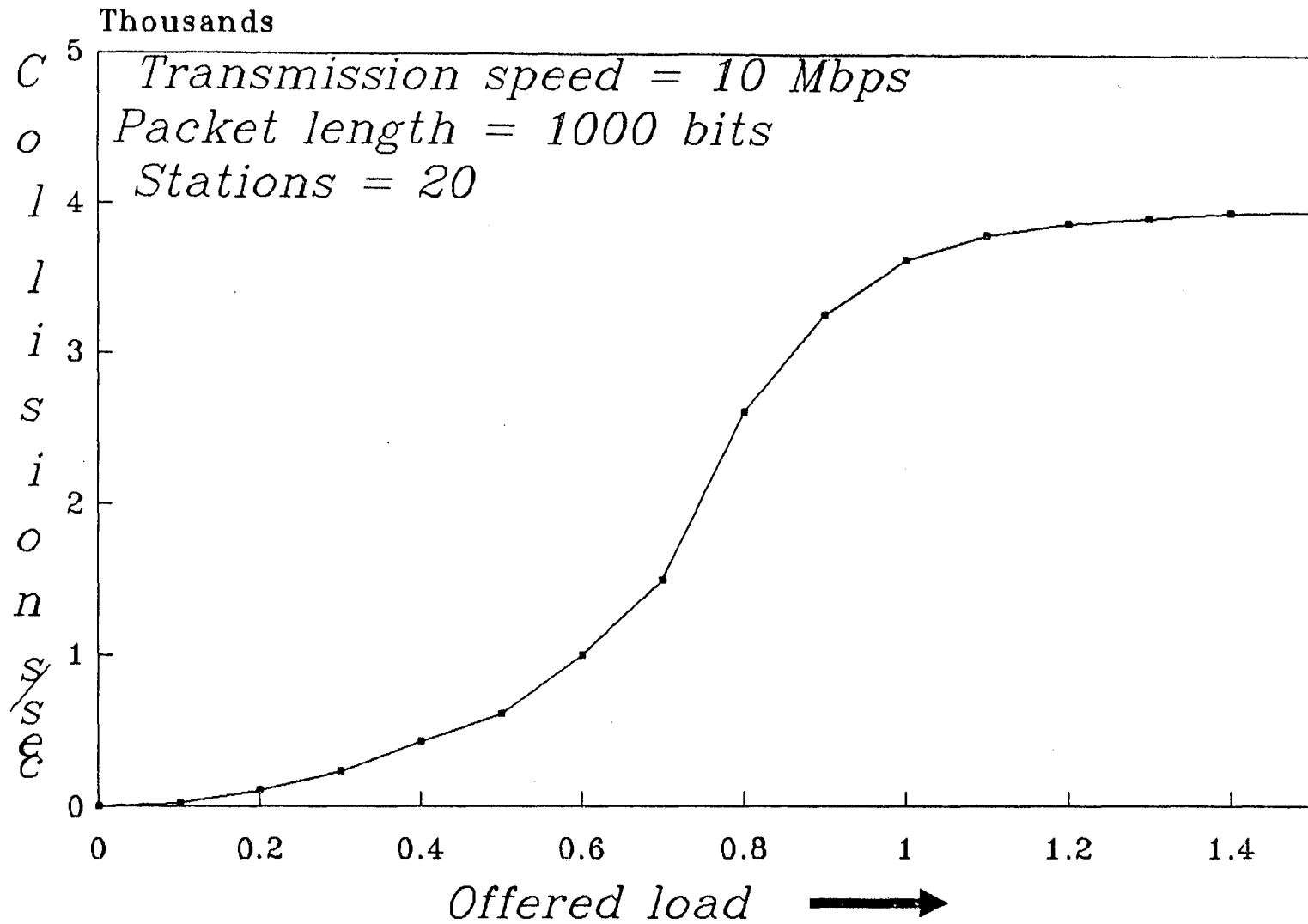


FIG. 6.2 CONFIDENCE INTERVAL



*FIG 6.3 COMPARISION OF SIMULATION RESULTS WITH AND WITHOUT NETWORK PARTITIONING*

Fig 6.4 Offered load vs. Number of collisions/sec



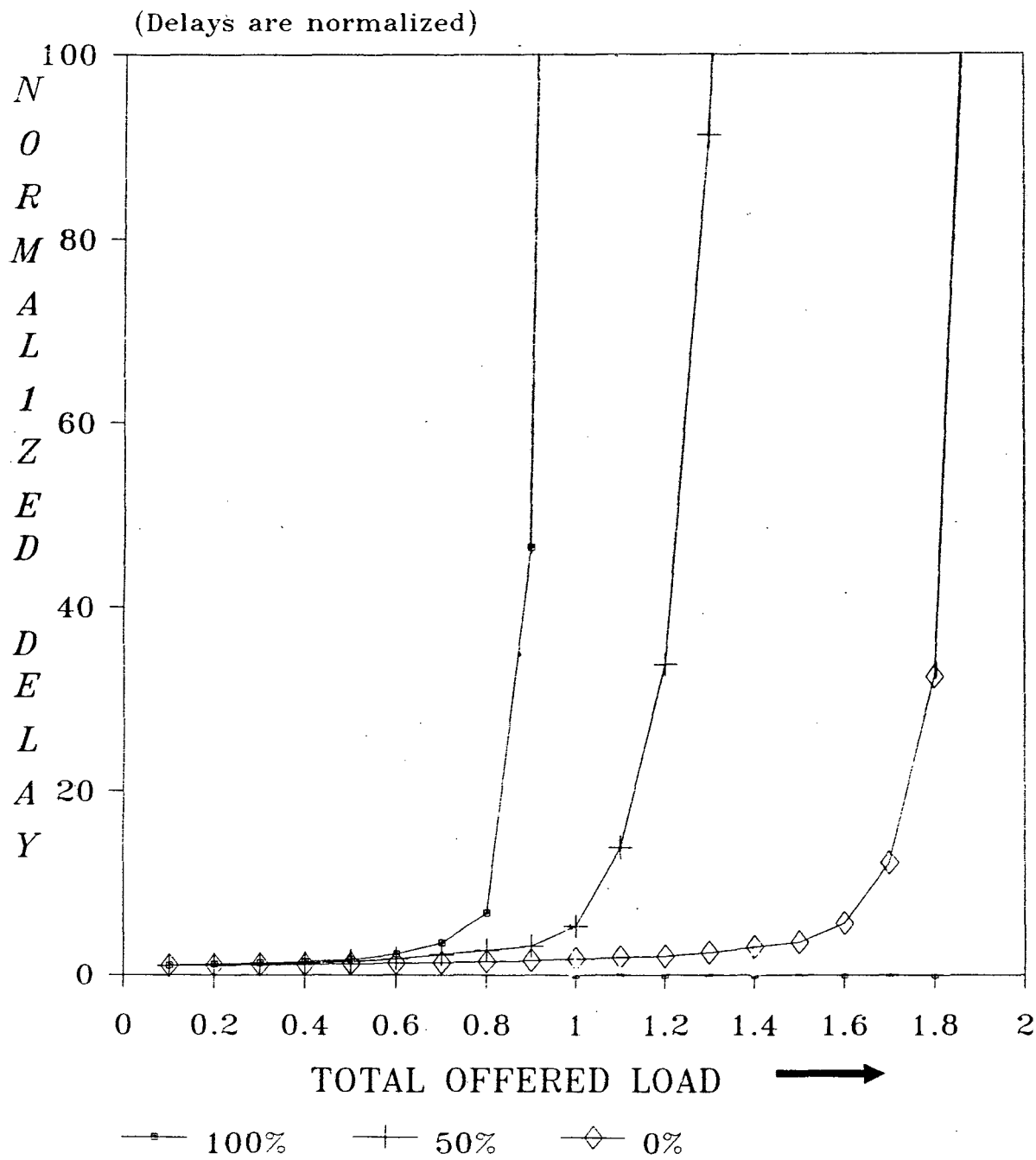


FIG. 6.5 SIMULATION RESULTS WITH NETWORK PARTITIONING - TRANSIENT MODEL



## PROGRAM LISTING

```
//=====
//      PROJECT : TO DEMONSTRATE THE CSMA/CD WITH NETWORK PARTITIONING
//=====
#include<iostream.h>
#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
#define nos1 50                // number of stations

FILE *fp;
FILE *fopen(const char *,const char *);

#define BUFFSIZE 50
#define maxbkoff 10
#define maxattempts 16

int    par_flag;              // partition flag
int    nos;
double lambda;
double lambda1;              //arrivalrate /sec /station
double trans_speed,prodel,slottime,inter_framegap,jamtime;
double dist[nos1],prop[nos1];
int    flag_backlog[nos1];   // =0 not backlogged
int    nofattempts[nos1];   // number of attempts
int    nofbackoffs[nos1];   // number of backoffs
int    retx_flag[nos1];     // retransmission flag
int    txnn;                 // node currently transmitting
int    cf;                   // collison flag
double simperiod;           // simulation period
int    discard;              // number of packets discarded
double packetsize;         // size of packet
double pkttxttime;         // packet transmission time
int    type;                 // type of the event
int    ntxstn;               // number of transmitting stations
unsigned int no_suctxpkt=0;  // number of successfully transmitted pkts
int    ncollison=0;         // total number of collisions
int    nofpkts[nos1];       // number of packets in a station
double cowindow;
int    csups;                // current upstream node
int    csdns;                // current downstream node
int    farupstntx;          // farthest upstream node transmitting
int    fardnstntx;          // farthest downstream node transmitting
double txsbyfdn;           // transmission stoptime by farthest downstream node
double txsbyfup;           // transmission stoptime by farthest upstream node
double clk;                 // clock
```

```

double cpdelay;
double pktstart;
int count,overfpkts=0;
double atbuff[nos1][BUFFSIZE]; // arrival time buffer
int last[nos1];
int oldpkt[nos1];
double topevtime;
unsigned int npkts=0;
int packets; // total number of packets generated
// avarage waiting time

double sum=0.0;
#define siz 24
double ar_avgwtime[siz];
double offeredload[siz];
double throughput[siz];
float scale2;
int end;
int k0,m0,n0,d;
double smy0,smy1,smy2,y0,h0;
//=====
#define idle 0
#define busy 1
class bus
{
public:
    int flag[nos1];
    bus() // constructor to intialise the flag
    { // to idle
        int j;
        for(j=0;j<nos1;j++)
            flag[j]=idle;
    }
    void makeidle(int i)// to make the channel status as seen by
    { // station i to idle
        flag[i]=idle;
    }
    void makebusy(int i)
    {
        flag[i]=busy;
    }
    int getstatus(int i)
    {
        return flag[i];
    }
};
bus channel;

```

```

//=====
// CLASS FLOAT-COUNTER DEFINED TO HOLD THE VALUE OF CURRENT SIMULATION
// CLOCK AND MEMBER FUNCTIONS TO GET AND TO SET THE VALUE OF IT
//=====
class float_counter
{
private:    double value;
public :
    float_counter()
    {
        value=0.0;
    }
    void increment(double x)
    {
        value+=x;
    }
    double get(void)
    {
        return value;
    }
    void set(double x)
    {
        value=x;
    }
};
float_counter sim_clk;

//=====
struct link
{
    double time;
    int type;
    link *next;
};

class linklist
{
private :    link *first;
public :
    linklist()
    {
        first = NULL;
    }
    double gettime(int t);
    double gettime()
    {
        return first->time;
    }
    int gettype()
    {

```

```

        return first->type;
    }
    void ascendadd(double d,int t);
    void display();
    void remove(int t);
    void remove()
    {
        link *templ=first;
        first=first->next;
        delete(templ);
    }
    int present (int t);
};

double linklist::gettime(int t)
{
    link *current=first;
    while((current!=NULL)&&(current->type !=t))
        current=current->next;
    if(current!=NULL)
        return (current->time);
    else return 0.0;
}
int linklist::present(int t)
{
    link *current=first;
    while((current!=NULL)&&(current->type !=t))
        current=current->next;
    if(current!=NULL)
        return 1;
    else return 0;
}

void linklist::ascendadd(double d,int t)
{
    link *newlink= new link;
    newlink->time=d;
    newlink->type=t;
    link *current=first;
    link *prev=new link;
    link *pr1=prev;
    while ((current!=NULL)&&(current->time<d))
    {
        prev=current;
        current=current->next;
    }
    newlink->next=current;
    if(prev!=pr1)
        prev->next=newlink;
}

```

```

        if(current==first)
            first=newlink;
        delete(pr1);
    }

void linklist::display()
{
    link *current=first;
    while(current!=NULL)
    {
        cout<<endl<<current->time<<"\t"<<current->type;
        current=current->next;
    }
}

void linklist::remove(int t)
{
    link *current=first;
    link *prev=first;
    while((current!=NULL) && (current->type!=t))
    {
        prev=current;
        current=current->next;
    }
    if(current!=NULL)
    {
        prev->next=current->next;
        if (first==current)
            first=current->next;
        delete(current);
    }
}

linklist evlist;
//=====
void set_standard1 ( void );
void set_standard2 ( void );
void channelacquire ( int );
void packet_completion ( void );
void collisondetect ( int );
void initbackoff(int);
void channelidle(void);
void insert (int );
void updatestat(int);
void drawgraph(void);
int obs(double);
void init_bm(int ,int);

```

```

//=====
//          THIS FUNCTION GENERATES A RANDOM NUMBER BETWEEN [0,1)
//=====
double frand(void)
{
    return ((double) rand()/((double)RAND_MAX+1.0));
}
//=====
//          THIS FUNCTION GENERATES ARRIVAL TIME
//=====
void arrival_time (int k)
{
    long double inter,f,lambda2;
    f=frand();
    if(k==0)
        lambda2=lambda1;
    else
        lambda2=lambda;
    inter = ((-1)*log(1-f)*1.0e6/lambda2)+1.0 ;
    if(inter<0.0) {
        printf("error: negative inter arrival time generation\n");
        exit(1);
    }
    evlist.ascendadd(sim_clk.get()+inter,k+nos+2);
    npkts++;
}
//=====
//          THIS FUNCTION RETURNS RANDOM INTEGER BETWEEN 0 TO 2**TRANSATTEMPT-1
//=====
int bin_exp_backoff (int trans_attempt)
{
    return ((unsigned int) (frand()*pow(2,trans_attempt)));
}
//=====
//          THIS FUNCTION IS USED TO DISCARD THE PACKET AFTER 16 ATTEMPTS
//=====
void discards(int i)
{
    oldpkt[i]=(oldpkt[i]+1)%BUFSIZE;
    flag_backlog[i]=0;
    nofbackoffs[i]=0;
    nofattempts[i]=0;
    retx_flag[i]=0;
    nofpkts[i]--;
    discard++;
}

```

```

//=====
//          THIS FUNCTION IS USED TO SET THE INPUT PARAMETERS
//=====
void set_parameters(void)
{
    int sp;
    int i;

    printf("Specify the distance array \n");
    printf("To give Ethernet parameters give 1 \n");
    printf("To give Schoch's parameters give 2 \n");
    printf("To give user specified parameters give 3 \n");
    scanf("%d",&sp);
    if(sp==1) set_standard1();
    else
    if(sp==2) set_standard2();
    else
    {
        for (i=0;i<nos-1;i++)
        {
            printf("Give the distance between station %d and %d \n",i,i+1);
            scanf("%f",dist[i]);
        }
        printf("Give transmission speed in Mbps \n");
        scanf("%lf",&trans_speed);
        printf("slottime in micro secs. \n");
        scanf("%lf",&slottime);
        printf("inter frame gap in micro secs\n");
        scanf("%lf",&inter_framegap);
        printf("Give jam time \n");
        scanf("%lf",&jamtime);
    }
    printf("the mean packetsize\n");
    scanf("%lf",&packetsize);
    fprintf(fp,"the mean packetsize == %f\n",packetsize);
    pkttxtime=packetsize/trans_speed;
    for(i=0;i<(nos-1);i++)
        prop[i]=dist[i]*prodel;
}
//=====
//          THIS FUNCTION IS USED FOR INTIALISATION
//=====
void start_simulate(void)
{
    int i;
    long double at,iat;
    long double lambda2;
    for(i=0;i<nos;i++)
    {
        nofpkts[i]=0;
    }
}

```

```

oldpkt[i]=0;
last[i]=0;
at=frand();
if ((i==0)&&(par_flag==1))
    lambda2=lambda1;
    else
        lambda2=lambda;
iat=((-1)*log(1-at)*1.0e6/lambda2)+1;
evlist.ascendadd(iat,2+nos+i);
nofattempts[i]=0;
nofbackoffs[i]=0;
retx_flag[i]=0;
flag_backlog[i]=0;
}
}
//=====
//      THIS FUNCTION SETS THE STANDARD ETHERNET PARAMETERS
//=====
void set_standard1(void)
{
int i;
fprintf(fp, " \nETHERNET PARAMETERS \n ");
for (i=0;i<nos-1;i++)
    dist[i]=111.1111111111;           //55.555555;
trans_speed=1;
slottime=51.2 ;
inter_framegap=9.6;
prodel=0.0512;
jamtime=3.2;
}
//=====
//      THIS FUNCTION SETS THE SCHOCH'S EXPERIEMENTAL ETHERNET PARAMETERS
//=====
void set_standard2(void)
{
int i;

for (i=0;i<nos-1;i++)
    dist[i]=61.11111111;
trans_speed=2.94;
slottime=5.5;
inter_framegap=9.6;
prodel=0.005;
jamtime=3.2;
}

```



```

//=====
// THIS IS THE MAIN FUNCTION USED. DEPENDING ON THE TYPE OF THE EVENT
// OCCURED, IT MAKES DIFFERENT FUNCTIONS.
//=====
void csmacd(int k)
{
int i;
switch(k)
{
case 0:
i=csup;
channel.makeidle(i); // senses no carrier on channel
if((flag_backlog[i]==0)&&(nofpkts[i]>=1))// if not backlogged
channelacquire(i); // and pkts are waiting then acquire
else
{
if((flag_backlog[i]==1)&&(evlist.gettime(i+2)<=sim_clk.get()))
channelacquire(i);
//if backlogged & backlog completion event completes
else // then acquire else set chidle for adjacent stn
{
if(i!=0)
{
csup=i-1;
evlist.ascendadd(sim_clk.get()+prop[csup],0);
}
}
}
break;

case 1:
i=csdns;
channel.makeidle(i);
if((flag_backlog[i]==0)&&(nofpkts[i]>=1))
channelacquire(i);
else
{
if((flag_backlog[i]==1)&&(evlist.gettime(i+2)<=sim_clk.get()))
channelacquire(i);
else
{
if(i!=(nos-1))
{
csdns=i+1;
evlist.ascendadd(sim_clk.get()+prop[csdns-1],1);
}
}
}
break;
}
}

```

default:

```
    if ((k>=2) && (k<=(nos+1)))
    {
        if (channel.getstatus(k-2)==0) channelacquire(k-2);
    }
    else
    {
        if (k==(2*nos+2))
        {
            i=txnn;
            no_suctxpkt++;
            nofattempts[i]=0;
            nofbackoffs[i]=0;
            flag_backlog[i]=0;
            updatestat(txnn);
            oldpkt[i]=(oldpkt[i]+1)%BUFSIZE;
            nofpkts[i]--;
            csups=txnn;
            evlist.ascendadd(sim_clk.get()+inter_framegap,0);
            if (txnn!=(nos-1))
            {
                csdns=txnn+1;
                evlist.ascendadd(sim_clk.get()+prop[txnn]+inter_framegap,1);
            }
        }
        else
        {
            i=k-nos-2;
            if (nofpkts[i]<BUFSIZE)
            {
                atbuff[i][last[i]]=sim_clk.get();
                last[i]=(last[i]+1)%BUFSIZE;
                nofpkts[i]++;
            }
            else
            {
                overfpkts++;
                arrival_time(i);
                if ((channel.getstatus(i)==0) && (flag_backlog[i]==0))
                    channelacquire(i);
            }
        }
    }
    break;
```

```
}
```

```

//=====
// IF A STATION IS READY TO TRANSMIT IT CALLS THIS FUNCTION.THIS CHECKS
// IT CAN SUCCESSFULLY TRANSMIT OR NOT.INCASE OF COLLISION IT CALLS FUNCTION
// 'CHANNEL IDLE'.
//=====
void channelacquire(int i)
{
    int temp1;
    double temp2,temp3;
    ntxstn=1;           // no.of transmitting stns
    int j;
    int m,n;
    evlist.remove(0);
    evlist.remove(1); // remove channel idle events from eventlist
    txsbyfdn=txsbyfup=sim_clk.get()+slottime+jamtime;//initialise to worstcase
    channel.makebusy(i);
    farupstntx=fardnstntx=i; // extreme node participating in collison
    if(i!=(nos-1))
    {
        clk=sim_clk.get();
        cowindow=sim_clk.get()+prop[i];
        cpdelay=prop[i];
        temp1=i;
        for(j=i+1;j<=(nos-1);j++)
        {
            collisondetect(j);
            if(cf==1) // collison occurs
            {
                fardnstntx=j;
                temp2=cowindow+jamtime;
                if(txsbyfdn>temp2)
                    txsbyfdn=temp2;
                if(temp1==i)
                {
                    temp3=pktstart+cpdelay+jamtime;
                    if(txsbyfup>temp3)
                        txsbyfup=temp3;
                }
                temp1=j;
                if(j!=(nos-1))
                {
                    cowindow=pktstart+prop[j];
                    cpdelay=prop[j];
                }
                clk=pktstart;
                ntxstn++;
            }
        }
        if(cf==0)
        {
            if (j!=(nos-1))

```

```

        {
            cowindow+=prop[j];
            cpdelay+=prop[j];
        }
    }
}
if(i!=0)
{
    clk=sim_clk.get();
    cowindow=sim_clk.get()+prop[i-1];
    cpdelay=prop[i-1];
    temp1=i;
    for(j=i-1;j>=0;j--)
    {
        collisondetect(j);
        if(cf==1)
        {
            farupstntx=j;
            temp2=cowindow+jamtime;
            if(txsbyfup>temp2)
                txsbyfup=temp2;
            if((temp1==i)&&(fardnstntx==i))
            {
                temp3=pktstart+cpdelay+jamtime;
                if(txsbyfdn>temp3)
                    txsbyfdn=temp3;
            }
            temp1=j;
            if(j!=0)
            {
                cowindow=pktstart+prop[j-1];
                cpdelay=prop[j-1];
            }
            clk=pktstart;
            ntxstn++;
        }
        if(cf==0)
        {
            if(j!=0)
            {
                cowindow+=prop[j-1];
                cpdelay+=prop[j-1];
            }
        }
    }
}
if(ntxstn>=2)
{
    flag_backlog[i]=1;
}

```

```

    retx_flag[i]=1;
    ncollison++;
    if(maxbkoff>nofbackoffs[i])
    {
        nofbackoffs[i]++;
        nofattempts[i]++;
    }
    else
        if(maxattempts>nofattempts[i])
            nofattempts[i]++;
        else
            discards(i);
    channelidle();
}
if(ntxstn==1)
{
    txnn=i;
    evlist.ascendadd(sim_clk.get()+pkttxttime,2*nos+2);
}
}
//=====
// FUNCTION WHICH CHECKS WHETHER A COLLISION OCCURS OR NOT.
//=====
void collisondetect(int j)
{
    cf=0;
    if(flag_backlog[j]==1) //if station j is backlogged check whether
    {
        if((evlist.present(j+2)==1)&&(evlist.gettime(j+2)<=cowindow))
        {
            evlist.remove(j+2);
            cf=1;
            if(evlist.gettime(j+2)<=sim_clk.get())
            {
                if(channel.getstatus(j)==1)
                    pktstart=cowindow;
                else pktstart=sim_clk.get();
            }
            else
            {
                pktstart=evlist.gettime(j+2);
            }
        }
    }
    else //if station is not backlogged
    {
        if(nofpkts[j]>=1)
        {
            cf=1;
            pktstart=(channel.getstatus(j)==1)?cowindow:sim_clk.get();
        }
    }
}

```

```

    retx_flag[i]=1;
    ncollison++;
    if(maxbkoff>nofbackoffs[i])
    {
        nofbackoffs[i]++;
        nofattempts[i]++;
    }
    else
        if(maxattempts>nofattempts[i])
            nofattempts[i]++;
        else
            discards(i);
    channelidle();
}
if(ntxstn==1)
{
    txnn=i;
    evlist.ascendadd(sim_clk.get()+pkttxtime,2*nos+2);
}
}
//=====
// FUNCTION WHICH CHECKS WHETHER A COLLISION OCCURS OR NOT.
//=====
void collisondetect(int j)
{
    cf=0;
    if(flag_backlog[j]==1) //if station j is backlogged check whether
    {
        if((evlist.present(j+2)==1)&&(evlist.gettime(j+2)<=cowindow))
        {
            evlist.remove(j+2);
            cf=1;
            if(evlist.gettime(j+2)<=sim_clk.get())
            {
                if(channel.getstatus(j)==1)
                    pktstart=cowindow;
                else pktstart=sim_clk.get();
            }
            else
            {
                pktstart=evlist.gettime(j+2);
            }
        }
    }
}
else //if station is not backlogged
{
    if(nofpkts[j]>=1)
    {
        cf=1;
        pktstart=(channel.getstatus(j)==1)?cowindow:sim_clk.get();
    }
}

```

```

    }
    else
    if (evlist.gettime(nos+j+2) <= cowindow)
    {
        cf=1;
        pktstart=(channel.getstatus(j)==1)?cowindow:evlist.gettime(nos+j+2);
    }
}
channel.makebusy(j);
if(cf==1)
{
    if(pktstart < sim_clk.get())
        printf("error: pkt start is less \n");
    flag_backlog[j]=1;
    retx_flag[j]=1;
    if(maxbkoff > nofbackoffs[j])
    {
        nofbackoffs[j]++;
        nofattempts[j]++;
    }
    else
    if(maxattempts > nofattempts[j])
        nofattempts[j]++;
    else discards(j);
}
}
//=====
// THIS SETS THE RETRANSMISSION BACKOFF TIME FOR THE STATIONS WHICH ARE
// BACKLOGGED
//=====
void channelidle(void) // set the backoff completions for collided packets
{ // and set channel idle events

float delay=0.0;
double temp1,temp2,temp4;
int temp3,j;
double time,timedn,timeup;

for(j=farupstntx;j<fardnstntx;j++) // distance between extreme nodes
    delay+=prop[j]; // participating in collison
if(txsbyfup < sim_clk.get())
    printf(" txsby f up node is less than sim_clk\n");
if(txsbyfdn < sim_clk.get())
    printf(" txsby fdown node is less than sim_clk\n");

if(abs(txsbyfup-txsbyfdn) > delay)
{
    if(txsbyfup > txsbyfdn)
        txsbyfup=txsbyfdn+delay;
    else

```

```

        txsbyfdn=txsbyfup+delay;
    }
    timedn=txsbyfup;
    timeup=txsbyfdn+delay;
    for(j=farupstntx;j<=fardnstntx;j++) // this gives transmission stoptime
    { // at a node
        if((timeup+0.001-timedn)>=0.0)
        {
            csups=j;
            csdns=j+1;
            time=timeup;
            temp1=timeup;
            temp2=timedn+prop[j];
        }
        else
            time=timedn;
        if(j!=fardnstntx)
        {
            timeup-=prop[j];
            timedn+=prop[j];
        }
        if(retx_flag[j]==1) // set the backoff completion event
        {
            retx_flag[j]=0; // for nodes which are backlogged
            temp3=bin_exp_backoff(nofbackoffs[j]);
            if(temp3<0)
            {
                printf("\nerror: in backlog time generation \n");
                exit(1);
            }
            evlist.ascendadd(time+temp3*slottime+1,j+2);
            if(evlist.gettime(j+2)<sim_clk.get())
            {
                printf("\n error: in setting backlog \n");
                exit(1);
            }
        }
        ncollison++;
    }
}
evlist.ascendadd(temp1,0);
if(evlist.gettime(0)<sim_clk.get())
{
    printf(" error: ch idle 0 is set to less \n");
    exit(1);
} // set channel idle events since col;
if(csdns<nos)
{
    evlist.ascendadd(temp2,1);
    if(evlist.gettime(1)<sim_clk.get())

```



```

    {
        printf(" error: ch idle 1 is set to less \n");
        exit(1);
    }
}
}
//=====
//          UPDATING THE STATISTICS ( AVERAGE PACKET WAITING TIME )
//=====
void updatestat(int i)
{
    double pvertime;    // packet waiting time
    pvertime=sim_clk.get()-atbuff[i][olopkt[i]];
    end=obs(pvertime);
    sum=sum+pvertime;
    packets++;
}
//=====
//          INITIALIZATION OF THE BATCH PARAMETERS
//=====
void init_bm(int ma,int mb)
{
    d=ma;
    m0=mb;
    smy0=smy1=smy2=0.0;
    k0=n0=0;
}
//=====
//          BATCH MEANS ANALYSIS
//=====
int obs(double y)
{
    int r=0;
    double var;
    if(d)
        { d--;
          return r;
        }
    smy0+=y;
    n0++;
    if(n0==m0)
        {
            smy0/=n0;
            smy1+=smy0;
            smy2+=smy0*smy0;
            k0++;
            smy0=0.0;
            n0=0;
        }
}

```

```

    if (k0 >= 10)
    {
        y0 = smy1 / k0;
        var = (smy2 - k0 * y0 * y0) / k0 - 1;
        h0 = 1.65 * sqrt(var / k0);
        r = 1;
    }
}
return r;
}
//=====
//                               FOR DRAWING GRAPHS
//=====
void rgraph(double *x1, double *y1)
{
    int x[siz];
    int y[siz];
    int scale1 = 200 / siz;
    setbkcolor(GREEN);
    int i; int k;
    for (i = 0; i < siz; i++)
    {
        x[i] = ((int) (*x1 + i) * scale1 * siz);
        y[i] = ((int) (*y1 + i) * scale2);
    }
    line(20, 460, 20 + 400 * 1.2, 460);
    line(20, 60, 20, 460);
    for (i = 1; i <= siz; i++)
    {
        k = scale1 * i * 1.2;
        putpixel(20 + k, 460, 4);
        if (i <= 10)
            putpixel(20, 60 + k, 4);
    }
    line(20, 460, 20 + x[0], 460 - y[0]);
    for (i = 0; i < (siz - 1); i++)
    {
        circle(20 + x[i], 460 - y[i], 2);
        putpixel(20 + x[i], 460 - y[i], 4);
        line(20 + x[i], 460 - y[i], 20 + x[i + 1], 460 - y[i + 1]);
    }
    circle(20 + x[i], 460 - y[i], 2);
}

```

```

//=====
// THIS IS THE MAIN FUNCTION WHICH INITIALIZES ,FINDS THE EVENT WITH EARLIEST
// SCHEDULE TIME AND CALLS 'CSMACD' FUNCTION
//=====
int main(void)
{
    int stop;
    int i;
    double offload=0.0;
    double ncollisonsps[siz];
    double offeredload0[siz];
    double throughput0[siz];
    double ar_avgwtime0[siz];
    double ncollisonsps0[siz];
    double offeredload1[siz];
    double throughput1[siz];
    double ar_avgwtime1[siz];
    double ncollisonsps1[siz];
    double offeredload2[siz];
    double throughput2[siz];
    double ar_avgwtime2[siz];
    double ncollisonsps2[siz];
    double pass_thr_traffic;

    fp = fopen("c:\\rama\\res2.cpp","a");

    int gdriver = DETECT;
    int gmode=EGALO;
    initgraph(&gdriver,&gmode,"c:\\borlandc\\bgi");

    int k;
    randomize();
    for(i=0;i<3000;i++) // leave first 3000 random nos.
        k=rand();
    int j;

    do
    {
        printf("Indicate with network partitioning(give 1) or without(give 0) \n");
        scanf("%d",&par_flag);
        printf("%d\n",par_flag);
        if (par_flag==1)
        {
            printf(" Give the pass through traffic \n");
            scanf("%lf",&pass_thr_traffic);
            printf("%lf\n",pass_thr_traffic);
            nos=nos1/2;
        }
    }
    else nos=nos1;
}

```

```

fprintf(fp, "\n===== \n"
fprintf(fp, "NUMBER OF STATIONS == %d \n ", nos);
set_parameters();
printf("\n===== \n"
printf("offeredload    throughput    avg.waiting time    no.collisionspersec."
printf("\n===== \n"
double offlimit;
if (par_flag==1)
    offlimit=2.01;
else offlimit=1.01;
j=0;
offload=0.0;
while(offload<offlimit)
{
    end=0;
    npkts=0;
    init_bm(2000,2000);
    offload+=0.05;
    if (par_flag==1)
        lambda1=(offload/pkttxttime/nos*1.0e6)*(pass_thr_traffic*nos+1);
    else
        lambda1=offload/pkttxttime/nos*1.0e6;
        lambda= offload/pkttxttime/nos*1.0e6;
    start_simulate();
    no_suctxpkt=0;
    ncollison=0;
    packets=0;
    for(i=0;i<nos;i++)
    {
        channel.makeidle(i);
    }
    sum=0.0;
    sim_clk.set(0.0);
    simperiod=1.0e6;
    k0=0;
    smy0=smy1=smy2=y0=h0=0.0;
    while(!end)
    {
        topevtime=evlist.gettime();
        type=evlist.gettype();
        evlist.remove();
        sim_clk.set(topevtime);
        csmacd(type);
    }
}
fprintf(fp, "BUFFER SIZE === %d \n", BUFFSIZE);
fprintf(fp, "SIMULATION PERIOD === %lf \n", sim_clk.get());
int s,n;

```

```

if (par_flag==1)
{
    s=2*nos;
    n=2;
}
else
{
    s=nos;
    n=1;
}
printf(" %lf \t",npkts*pkttxttime/sim_clk.get());
printf("%lf \t",((double)no_suctxpkt)*n*pkttxttime/sim_clk.get());
fprintf(fp,"OFFERED LOAD == %lf ",npkts*pkttxttime/sim_clk.get());
fprintf(fp,"throughput=%lf \n",((double)no_suctxpkt)*n*pkttxttime/sim_clk.get());
fprintf(fp,"no.of packets=%u \n",npkts);
fprintf(fp,"successfully trans. pkts=%u \n",no_suctxpkt);
fprintf(fp,"overflow packets %d \n",overfpkts);
fprintf(fp,"discarded packets=%d \n",discard);
fprintf(fp,"collisions per second ==%d \n", (int)(ncollison*1e6/sim_clk.get()));
printf("%lf +/- %lf \t",sum/packets/pkttxttime,h0/pkttxttime);
printf("%d \n", (int)(ncollison*1e6/sim_clk.get()));
if((par_flag==1)&&(pass_thr_traffic==0.0))
{
    ar_avgwtime0[j]=(sum/packets)*1e-4;
    offeredload0[j]=lambda*s*pkttxttime/sim_clk.get();
    throughput0[j]=no_suctxpkt*n*pkttxttime/sim_clk.get();
    ncollisionsps0[j]=(double)ncollison*1e5/sim_clk.get();
}
if((par_flag==1)&&(pass_thr_traffic==0.5))
{
    ar_avgwtime1[j]=(sum/packets);
    offeredload1[j]=lambda*s*pkttxttime/sim_clk.get();
    throughput1[j]=no_suctxpkt*n*pkttxttime/sim_clk.get();
    ncollisionsps1[j]=(double)ncollison*1e5/sim_clk.get();
}
if((par_flag==1)&&(pass_thr_traffic==1.0))
{
    ar_avgwtime2[j]=(sum/packets);
    offeredload2[j]=lambda*s*pkttxttime/sim_clk.get();
    throughput2[j]=no_suctxpkt*n*pkttxttime/sim_clk.get();
    ncollisionsps2[j]=(double)ncollison*1e5/sim_clk.get();
}
if(par_flag==0)
{
    ar_avgwtime[j]=(sum/packets);
    offeredload[j]=lambda*s*pkttxttime/sim_clk.get();
    throughput[j]=no_suctxpkt*n*pkttxttime/sim_clk.get();
    ncollisionsps[j]=(double)ncollison*1e5/sim_clk.get();
}

```

```

    j++;
} // end of for loop
printf(" If you want to stop press 0 else any key \n");
scanf("%d",&stop);
printf("%d\n",stop);
}
while (stop!=0);
int graphd;
while(1)
{
printf(" IF YOU WANT THE GRAPH BETWEEN OFFERED LOAD VS. THROUGHPUT give 1\n");
printf(" OFFERED LOAD VS. AVG.WAIT.TIME give 2 \n");
printf(" OFFERED LOAD VS. NO.COLLISIONS PER SEC. give 3 \n");
scanf("%d",&graphd);
cleardevice();
if (graphd==1)
{
scale2=1.0;
outtextxy(80,20,"offeredload vs. throughput");
rgraph(offeredload,throughput);
rgraph(offeredload0,throughput0);
rgraph(offeredload1,throughput1);
rgraph(offeredload2,throughput2);
}
else if (graphd==2)
{
scale2=1.0;
outtextxy(80,10,"offeredload vs. average waitingtime");
rgraph(offeredload,ar_avgwtime);
rgraph(offeredload0,ar_avgwtime0);
rgraph(offeredload1,ar_avgwtime1);
rgraph(offeredload2,ar_avgwtime2);
}
else if (graphd==3)
{
scale2=1.0;
rgraph(offeredload,ncollisionsps);
}
else exit(0);
}
return 0;
}

```

## BIBLIOGRAPHY

- [1] W. Stallings, " Local networks performance", IEEE communications Magazine, 22(1984) 27-35.
- [2] W. Stallings, " Local Networks", Computing Surveys, 16: 3-41, March 1984.
- [3] W. Bux, "Performance issues in local\_area networks", IBM system Journal, 23(1984) 351-374.
- [4] R.M. Metcalfe and D.R. Boggs, " Ethernet: distributed packet switching for local computer networks", Communications of ACM 19(1976) 395-404.
- [5] F.A. Tobagi and V.B. Hunt, " Performance analysis of carrier, sense multiple access with collision detection", Computer networks 4(1980) 245-259.
- [6] J.F. Shoch and J. Hupp, "Measured performance of an Ethernet local network", Communications of ACM 23(1980) 711-721.
- [7] Robert A.Pitts, Ralph Martinez and Larry C. Schooley, "CSMA/CD with network partitioning", Computer networks and ISDN systems, 26(1993) 423-432.
- [8] David L.Eldredge , John D.McGregor and Marguerite K.Summers, " Applying the Object Oriented paradigm to discrete event simulations using the C++ language ", Simulation, Feb 1990, 83-91.

- [9] Andrew S. Tanenbaum, "Computer Networks", Prentice Hall of India, 1988.
- [10] H. Kobayashi, "Modelling and Analysis", An Introduction of system Performance Evaluation Methodology", Addison-Wesley publishing company, 1978.
- [11] Francis Neelamkavil, "Computer Simulating and Modelling", John Wiley and Sons, 1987.
- [12] Mc Dougall, "Simulating Computer Systems: Techniques and tools", MIT Press.
- [13] D. Bertasakas and R. Gallager, "Data Networks", Prentice Hall, 1989.
- [14] Kernighan and Ritchie, "The C programming language", Prentice Hall of India, 1990.
- [15] Stroustrup, Bjarne, "The C++ Programming language", Addison-Wesley Publishing Co., 1988.
- [16] Tanenbaum, Langsam and Augestein, "Data Structures using C", Prentice Hall of India 1994.
- [17] Robert Lafore, "Object Oriented Programming in Turbo C++", GALGOTIA Publications, 1993.

...