

780

DESIGN AND SIMULATION OF MULTIMEDIA PROTOCOL

*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**

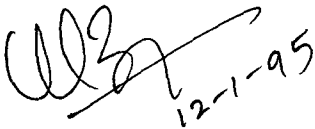
SANGITA GUPTA

**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110 067
JANUARY 1995**

CERTIFICATE

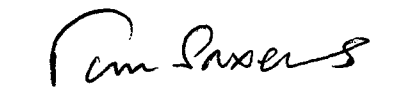
This is to certify that the dissertation titled "DESIGN AND SIMULATION OF MULTIMEDIA PROTOCOL" being submitted by SANGITA GUPTA to JawaharLal Nehru University in partial fulfilment of the requirements for the award of the degree of Master of Technology, is a record of the original work done by her under the supervision of Prof. P.C. Saxena, Professor, School of Computer and System Sciences, Jawaharlal Nehru University, New Delhi during the Monsoon Semester, 1994.

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


12-1-95

Prof. K.K. Bharadwaj
Dean,
School of Computer
and System Sciences,
Jawaharlal Nehru
University,
New Delhi.




Prof. P.C. Saxena
Professor,
School of Computer
and System Sciences,
Jawaharlal Nehru
University,
New Delhi.

ACKNOWLEDGEMENTS

I wish to express my sincere and heartfelt gratitude to Prof. P.C. Saxena, School of Computer and System Sciences, Jawaharlal Nehru University, for the unfailing support he has provided throughout. In all respects, I am very grateful to the patience he has exhibited and for the time he has spent with me discussing the problem. It would have been impossible for me to come out successfully without his constant guidance.

I extend my thanks to Prof. K.K. Bharadwaj, Dean, School of Computer and System Sciences, JNU for providing me the opportunity to undertake this project. I would also like to thank the authorities of our school for providing me the necessary facilities to complete my project.

I acknowledge and thank each and everyone of those who, directly or indirectly, helped me in this work.


(SANGITA GUPTA)

CONTENTS

Chapter #1	INTRODUCTION
Chapter #2	NETWORK BACKBONE ARCHITECTURE
2.1	CHOICE OF BACKBONE ARCHITECTURE
2.1.1	FIBERNET
2.1.2	EXPRESSNET
2.1.3	C-NET
2.1.4	D-NET
2.2	D-NET ARCHITECTURE
Chapter #3	RANDOM ACCESS STRATEGIES
3.1	OVERVIEW
3.2	ALOHA
3.3	LCSMA
3.4	LCSMA-CD
3.5	PROTOCOL DESIGN
3.6	FAIRNESS SCHEMES
3.6.1	EQUAL TIMER SETTING
3.6.2	CHANNEL ACCESS PROBABILITY ASSIGNMENT
3.7	MULTIMEDIA TRAFFIC
3.7.1	VIDEO DATA TRAFFIC
3.7.2	VOICE DATA TRAFFIC
3.7.3	ORDINARY DATA TRAFFIC
3.8	PRIORITY TRANSMISSION
3.9	DELAYS ENCOUNTERED BY PACKETS
3.9.1	QUEUING DELAY

	3.9.2	ACCESS DELAY
	3.9.3	TRANSFER DELAY
	3.10	PERFORMANCE INDICES
Chapter #4		SIMULATION
	4.1	INTRODUCTION
	4.2	CONTINUOUS SYSTEM SIMULATION
	4.3	DISCRETE SYSTEM SIMULATION
	4.3.1	FIXED TIME STEP MODEL
	4.3.2	EVENT TO EVENT MODEL
	4.4	STOCHASTIC SIMULATION
	4.5	RANDOM NUMBER GENERATION
	4.6	LENGTH OF SIMULATION RUN
	4.7	SIMULATION MODEL
	4.8	SIMULATION OF MULTIMEDIA TRAFFIC
Chapter #5		PERFORMANCE ANALYSIS
	5.1	DELAYS
	5.1.1	QUEUING DELAY
	5.1.2	ACCESS DELAY
	5.1.3	TRANSFER DELAY
	5.2	THROUGHPUT Vs. LOAD
	5.3	SUCCESSFUL TRAFFIC TRANSMISSION
	5.4	SIMULATION RUN LENGTH
Chapter #6		CONCLUSION AND FUTURE DIRECTIONS
		APPENDIX A
		APPENDIX B
		BIBLIOGRAPHY

CHAPTER ONE

INTRODUCTION

CHAPTER ONE

INTRODUCTION

Communication network have rapidly become an important and almost indispensable part of our lives today: from the telephones in our homes to E-Mail in our offices, the utility of these networks is undeniable. Traditionally, wide area networks have evolved separately around the end applications that they support. For example, the telephone network, which almost exclusively carries voice, has developed independently of information networks that carry data and as a result, the technologies involved are significantly different. These distinct networks served their purpose very effectively for the applications for which they were designed. But current trends are increasingly pushing diverse networks towards integrating into a common high-speed network. The high bandwidth nature of new applications necessitates an underlying network that can provide the raw bandwidth needed.

Multimedia computers process various kinds of data like video, voice, text, image and graphics. Distributed computing involving multimedia data on a local area network needs integration of transmission of all the above specified data traffics on the same network, to give one single integrated information distribution system. The underlying

network should be able to cater for the special requirements of video and voice traffics, i.e. high bandwidth and tightly bounded network delay.

Progress in optical fiber communications has generated tremendous interest in its application to local area network. The propagation loss of glass fiber at the optical wavelength of 1550 nm is as low as 0.2 dB/km and therefore fiber transmission system can be operated at 20-100 Gbits/sec data rates over distances greater than 100 km. Moreover optical fibers are immuned to EMI, they are chemically inert, light in weight and small in size. All these properties make optical fiber the best transmission medium available today.

For implementation of optical fiber LAN's there are various architectures available like star shaped bus, bidirectional bus structure like Ethernet and unidirectional bus structure. For this project I have chosen a unidirectional bus structure because reliability of a bus is more than that of a ring and low loss bidirectional taps for transmission and reception are difficult to implement.

The objective of this project is to design a network architecture to which multimedia computers can be connected. After an exhaustive study of various unidirectional bus architecture based optical fiber networks like C-Net, D-Net,

Fastnet, Express-Net and comparing their relative merits and demerits D-Net architecture has been chosen. D-Net possesses the advantages of high efficiency, low delay and simplicity in protocol. The multimedia computers are connected to the underlying network by active optical taps. Active optical taps override any existing signal on the bus, thus no slot is wasted unless nobody on the network wants to send data in that slot, but this also results in an inherent unfairness to upstream sources. Therefore the protocol design should be such that it removes the unfairness caused by active taps, so that all computers can access the transmission channel with equal fairness irrespective of their position on the network. The delay requirements of the video and voice traffic should be met. Efficient bandwidth utilization is another essential feature of the protocol design.

After the design of the multimedia protocol its performance analysis is carried out using a stochastic computer simulation.

CHAPTER TWO

NETWORK BACKBONE ARCHITECTURE

CHAPTER TWO

NETWORK BACKBONE ARCHITECTURE

2.1 CHOICE OF NETWORK ARCHITECTURE

The advantage of the low loss and wide bandwidth of optical fiber has opened new dimensions for such high bandwidth LAN applications. The very high data rate of optical fiber leads the network designer to consider a topology which uses as few links of fiber as possible so that the capacity of the fiber is maximally utilised. To implement fiber optic network we can use various architectures like star shaped bus, unidirectional bus, bidirectional bus, ring architecture etc. Reliability and survivability of bus is better than ring so bus is better suited than ring for implementation of this network. Low loss bidirectional T-taps are difficult to implement hence unidirectional bus architecture is chosen. An exhaustive study of the following unidirectional bus architectures is made.

2.1.1 FIBERNET

Fibernet is a fiber optics version of Ethernet. It uses a star shaped bus to connect the stations. The access protocol is based on the principle of carrier sense multiple

access with collision detection-CSMACD. Fibernet is very efficient when the end-to-end signal propagation delay is much less than the packet transmission time. But as the bus data increases, the efficiency drops to the range far below the acceptable levels. Another shortcoming of fibernet is that the delay of a packet is not deterministic. In worst case a packet can encounter an unlimited number of collisions this makes fibernet unsuitable for real time (voice, video) communications.

2.1.2 EXPRESSNET

It uses a unidirectional transmission medium. Each station is connected to the fiber at three taps, one to sense carrier (S), second to transmit (T) and third to receive data (R). S of a station can only sense carrier transmitted from "upstream" stations. While R of any station can receive any signal transmitted by any station. Each station of expressnet operates as follows.

(1) To transmit a packet a station waits for the event of next end of carrier (EOC) to be detected at S. When EOC is detected (t_d seconds after the real EOC occurrence), the station starts to transmit a packet, which consists of a preamble portion P_p , and an information portion P_i . Within t_d which is shorter than the transmission time of P_p , the station can detect at S whether there is collision.

2) If a collision is detected, the station aborts the packet transmission immediately, letting the Pi part of the packet from an upstream station go through , and it goes back to step 1. Otherwise it finishes its own packet transmission.

3) After successfully transmitting a packet, it waits for end-of-train (EOT) to be detected at its R, the station is allowed to search for (EOC) again.

Expressnet has efficiency of nearly one and delay is also low so that for a limited number of stations connected, voice communication quality is guaranteed. The main disadvantage of Express-Net is that the algorithm is complicated , which implies that the implementation of it is expensive.

All these complication arise from the fact that Express-Net uses EOC and EOT as synchronising events to make the protocol completely distributed, still the access protocol of the first station remains slightly different from the rest. It detects no EOC event. Hence because of its high complexity Express-Net is also not suitable for my application.

2.1.3 C-NET

C-Net uses a unidirectional bus network. A station is connected to the bus through taps R, S and T which are to

receive packets, sense carrier and transmit packets respectively. After detecting EOT at its R, a station is allowed to sense the outbound channel using its S. If the channel is busy, then it waits until the channel becomes idle. If the channel is idle it transmits a packet if it has to transmit. If this leads to a collision due to the simultaneous transmission of packets by the upstream stations, then station stops transmitting immediately and waits until the bus becomes idle again: otherwise, it finishes its packet transmission. After successfully transmitting a packet, a station has to wait until its R detects the EOT of the train which contains its own packet. Then it is allowed to sense the channel again.

On analysing the channel access strategy of C-Net it is found that the maximal number of voice channels of C-Net is only half that the expresnet. The poorer performance of C-Net is compensated by the gain in protocol simplicity.

2.1.4 D-NET

It is a unidirectional bus architecture. It has a locomotive generator at one end which is connected to the fiber at two points (Transmitter {T} and receiver {R}). It transmits at its T, a locomotive (pulse) every time it detects an EOT at its R. Each station uses EOC as the synchronisation event to send the packet. Whenever a

collision is detected, a station stops transmitting until the next EOC is detected at its S. This protocol is simpler than that of C-Net. Tap R of each station except that of the locomotive generator need not detect EOT as needed in Express-Net and C-Net. Tap T of each station except that of the locomotive generator does not need to generate a locomotive as needed in every station of Express-Net. D-Net is a much simpler and less expensive system than E-Net & C-Net.

On the analysis of all the above discussed architectures it was found that, D-Net possesses the advantages of high efficiency, low delay and simplicity in protocol. Hence, for this application I have chosen D-Net as the basic underlying network. A detailed discussion of D-Net architecture is given in the next section.

2.2 D-NET ARCHITECTURE

D-Net uses a unidirectional bus architecture (Fig 1). It consists of a locomotive generator (L.G.) which is located farthest upstream. Its function is to implement the slotted system by generation of slot timing pulse. A station can transmit data only when it detects these pulses at its carrier sense end. The interval between consecutive slot pulses is determined by the packet length chosen for the network and the bandwidth of the optical fiber. Each station

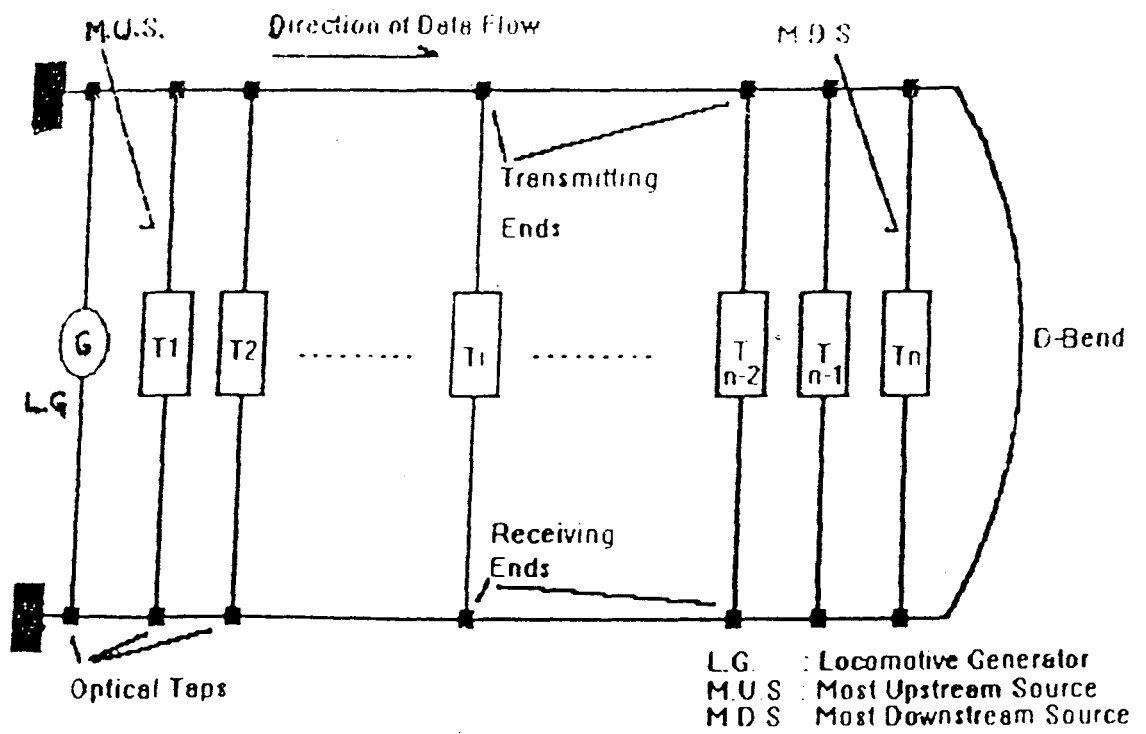


FIG. 1
D-NET ARCHITECTURE

(A multimedia computer) is connected to the fiber at two ends, one is called the transmitting end and the other is the receiving end. The optical fiber bus takes the shape of alphabet "D" that is why this network is called D-Net. The stations are connected to the fiber by optical taps. Optical taps can be of two types active or passive. A passive optical tap transmits the "OR" of the locally generated signal and the signal on the network. An active optical tap can switch between the locally generated signal and the signal on the network. A source transmits at its transmitting end and receives signal, from all of the sources, at the receiving end. The signal passes the transmitter end at each source, then it passes the D-bend and reaches the receiver end of the sources.

The sources whose transmitting ends are nearer to the locomotive generator are called upstream sources and those sources which are farther away from the L.G. are called downstream sources. A slot pulse starts at the locomotive generator, traverses the fiber starting from the transmitting end of the most upstream source to most downstream source, then it reaches the D-bend. After crossing the D-bend it traverses the receiving ends of all sources starting from most downstream to the most upstream, and finally it reaches the L.G. When L.G. receives a slot at

its receiver it generates a slot and transmits it on the fiber and this cycle continues.

Fig (2) describes the propagation of signals in the D-Net. The signal at the transmitting end Z_i of the i th source is the composite signal from the sources that are upstream from source terminal " T_i " and is referred to as "local information". This is called local because it provides partial information about the state of the network at a particular instant of time since only signals transmitted by relatively upstream sources are included and signals by the downstream sources are not present at that point. The signal at receiving end " R_i " of a sources that arrive at the bend in the 'D' network at a particular instant of time and is referred to as "global informations". The global signal provides complete information on the state of the network at a particular instant of time. When a slot accessed by a source reaches the receiving end of that source, it means completion of one round trip. This global signal is used by a source to determine when a retransmission is required. When a source receives the signal it has transmitted, it knows that the signal has been successfully transmitted and no collision has occurred with this packet. If it does not receive the transmitted packet, the station understands that the packet has met a collision and retransmission is required. This way it is a self acknowledging system.

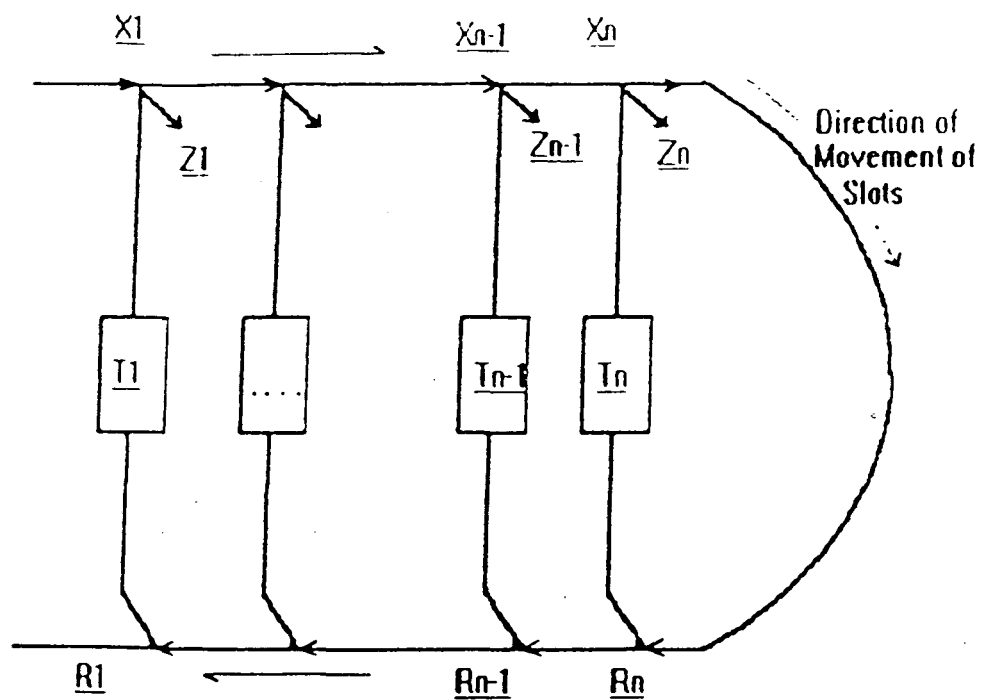


FIG. 2

DIRECTION OF DATA FLOW
IN D-NET

This network is unfair to the upstream sources. This unfairness is discussed with the help of fig.3. A source T1 transmits a packet in a slot. When this slot reaches source T4, it puts its packet in that slot overriding the already placed packet i.e. slot occupied by T1 is snatched by T4. This is because we are using active taps which override the already present data on the channel and place their data. This way downstream sources have a much better chance of getting their message transmitted successfully than the upstream sources. The collision rate of the packets of upstream sources is much higher than that of downstream sources.

Another unfairness is due to the self acknowledgement technique being used. a downstream source packet is acknowledged faster than an upstream source packet, because the packet of an upstream source has to travel greater distance to reach the receiving end. Therefore upstream sources have to wait longer for their next transmissions.

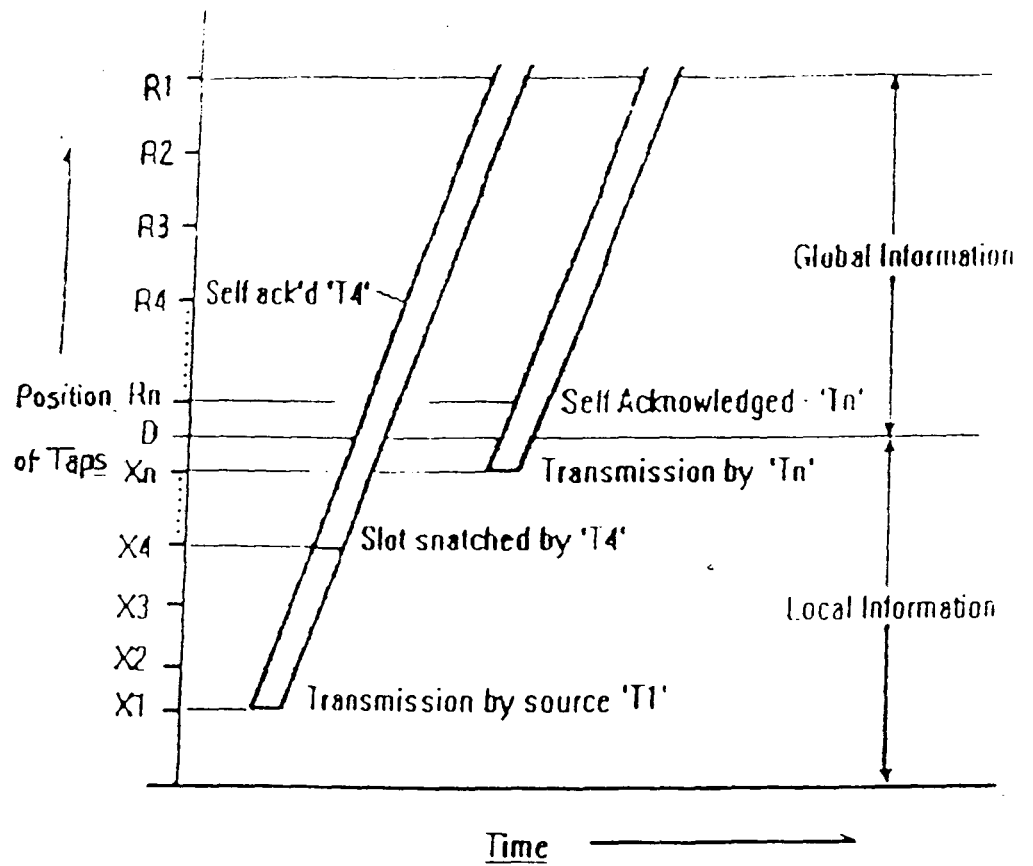


FIG. 3

INHERENT UNFAIRNESS IN D-NET

CHAPTER THREE

RANDOM ACCESS STRATEGIES

CHAPTER 3

RANDOM ACCESS STRATEGIES

3.1 OVERVIEW

An access strategy has to be designed to access the D-Net network. The strategy should use the information at the transmitter end in the unidirectional network for channel access. It should not constrain the distance or transmission rate of a network and should be able to use the capabilities of fiber optic LAN. Twelve random access strategies have been studied and analysed. The twelve access strategies consist of three protocols, each of which can use two timing arrangements and two network access devices. The three protocols are ALOHA, LCSMA, LCSMA-CD. The prefix 'L' is used to distinguish strategies that use local information at the transmitter from strategies that use complete information at the receiver. Each of these strategies can be implemented with passive taps, that transmit the 'OR' of the locally generated signal and the signal on the network, or active taps that switch between the locally generated signal and the signal on the network. The timing arrangement can be slotted or unslotted. In unslotted system a station can transmit whenever they find the channel free (in case of CSMA/CSMA-CD) or whenever they are ready to transmit (in

case of ALOHA). But in slotted system, the source furthest from the bend in the D-network periodically generated timing pulses that signify the start slot. A source is allowed to transmit only when it detects the start of slot pulse at its transmitting end. The implementation and merits and demerits of the three protocols with their combination of taps & timing arrangement are discussed in the next section. In this discussion the strategies are referred to as protocol/timing/access where

- The protocol is ALOHA, LCSMA or LCSMA-CD
- The timing is unslotted (U) or slotted (S)
- The network access is passive (P) or active (A)

An X indicates that all of the values of a parameter are considered.

3.2 ALOHA

ALOHA/U/P is the conventional ALOHA protocol. A source transmits as soon as it has a packet, and it retransmits as soon as it comes to know about the collision. Acknowledgements are not required in ALOHA/U/P to determine when a signal was received without collision because the same global signal is received by all of the receivers, and a source can determine whether or not it has collided by examining this signal.

ALOHA/U/A uses the same strategy as ALOHA/U/P except that when a collision occurs, the signal from the downstream source successfully acquired the network, whereas a ALOHA/U/P, a collision means no valid data is carried in that slot. Therefore ALOHA/U/A gives preference to downstream sources.

ALOHA/S/P is the conventional slotted ALOHA protocol. Packets that arrive during a packet transmission interval are transmitted at the beginning of the next interval. In ALOHA/U/P there is no concept of slot intervals, as soon as a packet is ready, it is transmitted.

3.3 LCSMA

In LCSMA/X/X, a source listens to the signal from the upstream sources before transmitting. If an upstream source is transmitting, the local source delays its transmission. This strategy gives preference to upstream sources. In LCSMA/U/P, a source

- (1) is delayed by packets from upstream sources that are propagating past the transmitter when a packet arrives.
- (2) collides with packets from upstream sources that arrive at the transmitter during the packet transmission.
- (3) collides with downstream sources that are already transmitting when the packet arrives at their transmitter.

This strategy results in fewer collisions than that in ALOHA/U/P. In this system, sources that detect a busy channel can implement a retry strategy immediately instead of waiting for a round trip propagation delay as in ALOHA/X/X. LCSMA/U/A combines taps that give priority to downstream sources with a transmission rule that gives priority to upstream sources and results in a fair access strategy. Though LCSMA has some advantages over ALOHA, but the protocol is complex and difficult to implement.

3.4 LCSMA-CD

In LCSMA-CD/X/X, a source listens to the signal both before and during its transmission. A source does not transmit if an upstream source is active, and stops transmitting if an upstream source becomes active. Upstream sources always have preference over downstream sources. In LCSMA-CD/U/P, a source does not transmit if a packet from an upstream source is passing the local transmitter when packet arrives, and is preempted if an upstream source transmits a packet that arrives at the local transmitter while a packet is being transmitted.

In LCSMA-CD/S/P, one source wins in every slot in which one or more sources transmit, as in ALOHA/S/A. However, in LCSMA-CD/S/P, upstream, rather than downstream, sources are given priority. In LCSMA-CD/U/P, a source that uses a

persistent retry strategy acquires the channel as soon as there are no upstream sources waiting to transmit. the LCSMA-CD/U/P protocol can be made more efficient by adopting a preemptive resume strategy, in which a source that stops transmitting, transmits the interrupted portion of the packet, rather than the entire packet, when it resumes. This strategy increases the throughput, by not retransmitting data that have gotten through successfully, but significantly increases the complexity of the system.

3.5 PROTOCOL DESIGN

On carrying an exhaustive analysis of the above mentioned access strategies, it was found that these strategies provide mechanism for trading three types of complexity for throughput.

1. TIME SYNCHRONISATION
2. SIGNAL PROCESSING
3. TAP STRUCTURE

Time synchronised or slotted systems, adjust the transmission times of the various sources so that all fixed size packets arrive at a common point in the network at the same instant. Signal processing is used to avoid collisions with upstream sources by examining the channel before or during transmission. Taps that switch a transmitter into the

medium are used to allow sources to win in a contention situation. Slotted systems are always more efficient than unslotted systems and when synchronisation is combined with either signal processing or switched taps, these systems can approach a utilization of one.

On analysing the merits and demerits of the random access strategies, I have chosen a strategy which uses ALOHA protocol with active optical taps on a slotted time system.

Multimedia systems are connected to the optical fiber channel through active optical taps. Multimedia systems offering traffic to network are referred to as sources/stations. The systems which are switched on at a particular time are referred to as 'active'.

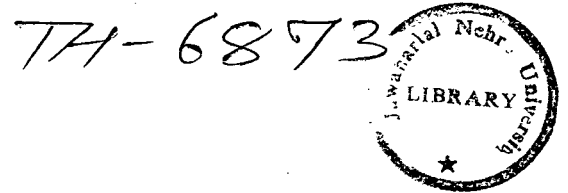
In a particular slot, which travels across all the active sources, every active sources can try to send a packet. When a source puts its data on the slot, any preexisting data is removed and new one is placed. This is because optical taps are reciprocal. In order to insert a fraction of a signal on to the fiber, the same fraction of the signal on the fiber must be removed. The active optical tap places a fraction of regenerated signal on the bus, replacing the incoming signal.

A source is ready to transmit when it has a new packet to transmit or when the waiting time, after the last

transmission attempt, is over. A source transmits the same packet till it succeeds. After this packet has reached its destination and is self acknowledged by the transmitter itself, then only next packet if any, is considered. After a successful channel access attempt, the source waits till at least that slot comes to its receiver end. If this slot has the same data which was transmitted by this source, it becomes self acknowledged and the transmission attempt is successful. This waiting time for a source, before retransmission attempt, is equal to the distance between the two ends of the source in terms of slots on the fiber.

Since all the data transmitted passes through receiving ends of all the sources, a self acknowledgement mechanism works and no separate acknowledgement traffic is generated. When the transmission of packet at a station is successful, its buffers can be released. The network specifications are given in Appendix B.

3.6 FAIRNESS SCHEMES



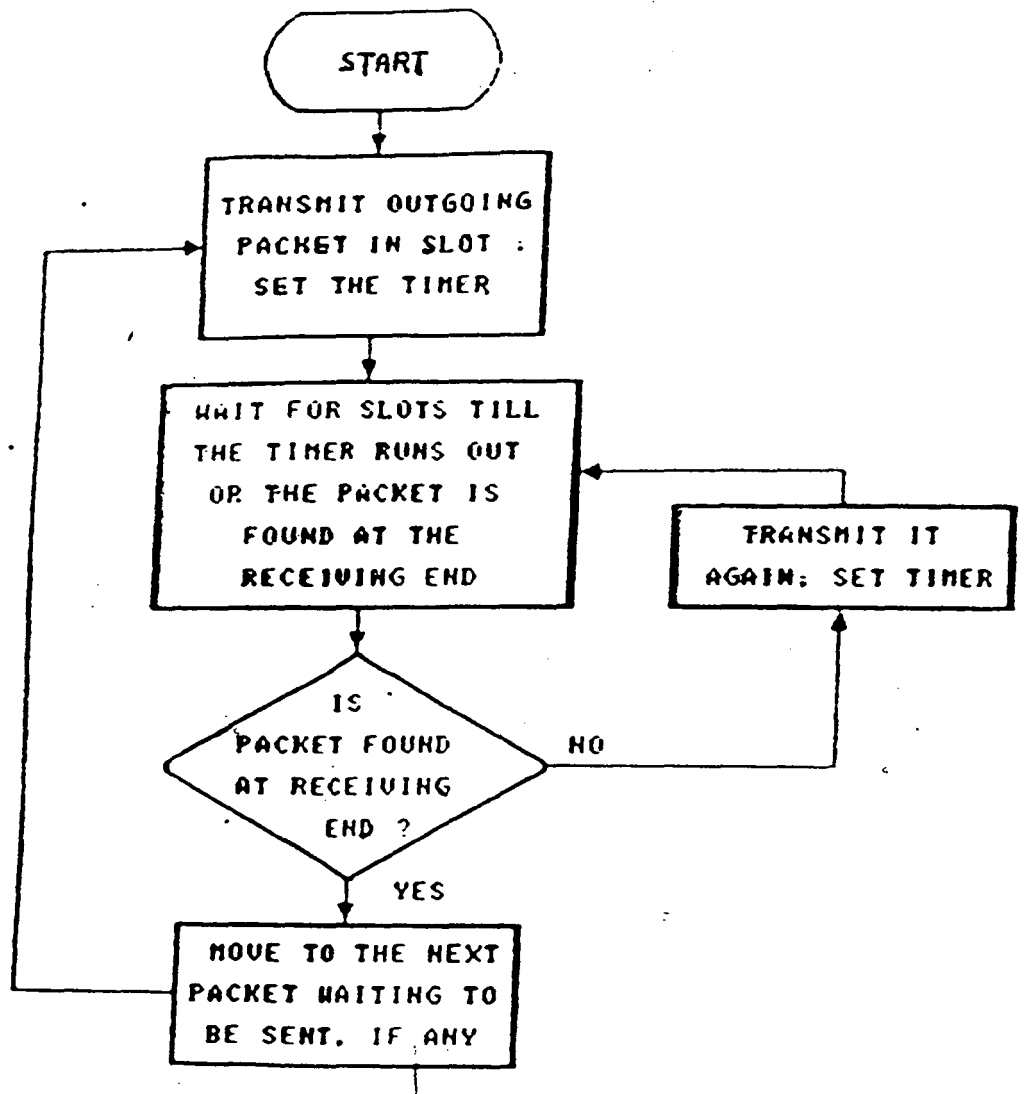
3.6.1 EQUAL TIMER SETTING

Every time a source sends a packet, it sets a timer within which the packet should reach the transmitting end. This timer delay is set according to the position of the source on the fiber. It is less for downstream sources than

upstream sources. Therefore, the downstream sources receive quicker self acknowledgement and quickly release the output buffer assigned to these acknowledged packets and they can attempt transfer of new packets. This system is unfair to the upstream sources. To provide fairness in this scenario, a technique called E.T.S. is employed. In this every source, irrespective of its position, waits for equal time before making a retransmission attempt after a successful channel access, in case its packet is not self acknowledged within timer intervals and hence is deemed to have suffered a collision. This time delay is kept equal to the time gap between the two ends of the most upstream active source, independent of the actual time gap between the two ends of the source. This balances the favour enjoyed by the downstream sources. The channel access procedure using the E.T.S. is described in flow chart #1

3.6.2 CHANNEL ACCESS PROBABILITY ASSIGNMENT

After passing across transmitting ends of all sources, the slot carries data sent by the most downstream one (i.e. the last one) of those sources which had put their data on this slot. So the data transmitted by an upstream in a slot can only get through when no downstream source accesses that slot. Thus going downstream on the fiber, it becomes progressively easier for the sources to get their data through. This is clearly favourable to downstream sources.



FLOW CHART #1
CHANNEL ACCESS IN E.T.S. SCHEME

To provide equal fairness to all sources independent of their position, the downstream sources are made less greedy in accessing the channel. The channel access protocol is modified to make it a p-persistent ALOHA rather than 1-persistent ALOHA. The probability (P) of a ready source trying to access a slot is made dependent on the position of the source on the fiber. Upstream sources have higher probability and downstream sources have lower probability of accessing the channel as shown in fig - 4. Before accessing a slot, a source generates a random number (0,1) and if this number is less than the probability value assigned to this source, it accesses this slot, otherwise leaves it. This process is shown in flowchart #2. Thus the downstream sources leave some slots free before accessing a slot. The data transmitted by the upstream sources gets through in these free slots. The probability assignment is such that if all the sources are active, then all the sources have equal effective value of slot access probability independent of their position.

In an "N" active sources arrangement, the probability assigned to the ith source from start of the bus is

$$P(i) = 1/(i)$$

If all of them, turn by turn, try to access the same slot, the effective probability of every source is equal to

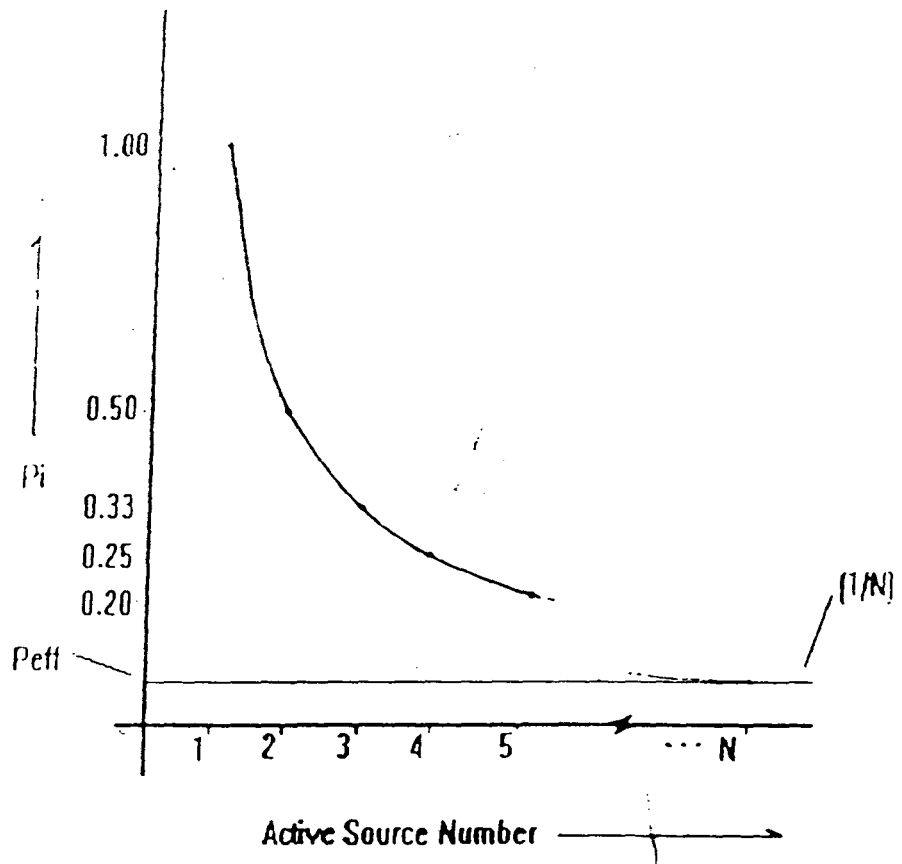
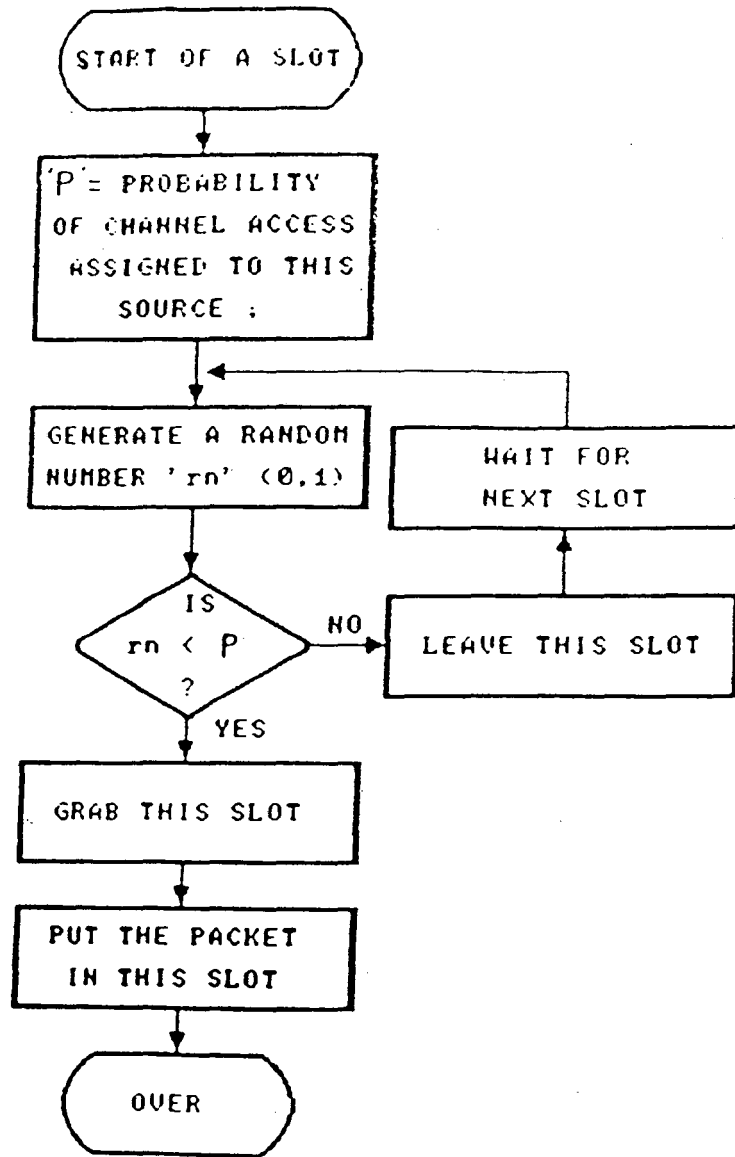


FIG. 4
CHANNEL ACCESS PROBABILITY ASSIGNMENT



FLOW CHART # 2
CHANNEL ACCESS IN C.A.P. ASSIGNMENT

$$P_{eff} = 1/(N)$$

because the downstream source can overwrite the signal from the upstream sources in that slot.

Consider such a slot in which every source tries to put its data with above probability assignment. The mathematical analysis given below derives the effective probability of channel access.

MATHEMATICAL ANALYSIS FOR EFFECTIVE CHANNEL ACCESS

Total number of active sources = N

Probability assigned to the i th source from start of bus is given by P_i where

$$P_i = (1/i)$$

Therefore probability assigned to most upstream source ($i=1$)

$$P_1 = 1/1 = 1$$

Probability assigned to N th source (most downstream source)

$$P_N = (1/N)$$

Effective probability of i th source

= P_i * (probability that none of the sources further downstream access this slot)

$$\begin{aligned}
&= P_i * (1-P_j) \\
&= P_i * (1-(1/i+1)) * (1-(1/i+2)) * \dots * (1-(1/N)) \\
&= (1/i) * (i/i+1) * (i+1/i+2) * \dots * (1-N/N) \\
&= 1/N
\end{aligned}$$

So for any value of i , effective channel access probability for source " i " = $(1/N)$ where N = number of active sources.

This analysis shows that by this probability assignment, all the sources get equal effective access to the channel. Hence resulting in a fair access strategy.

The strategy works like this : every time a source sends a packet, it sets a timer within which the packet should reach the transmitting end. This timer delay can be set according to the position of the source on the fiber or it can be set to be equal to the distance between the transmitter and receiver ends of the most upstream source. If the packet is not acknowledged before the timer runs out, a retransmission attempt is made at the next slot with a probability assigned to this source. This attempt is made at every consecutive slot with this probability, until success occurs. In case of failed attempt, the same timer is set repetitively till the packet transmission is self acknowledged within the timer interval. The timing diagram for channel access is given in Fig - 5.

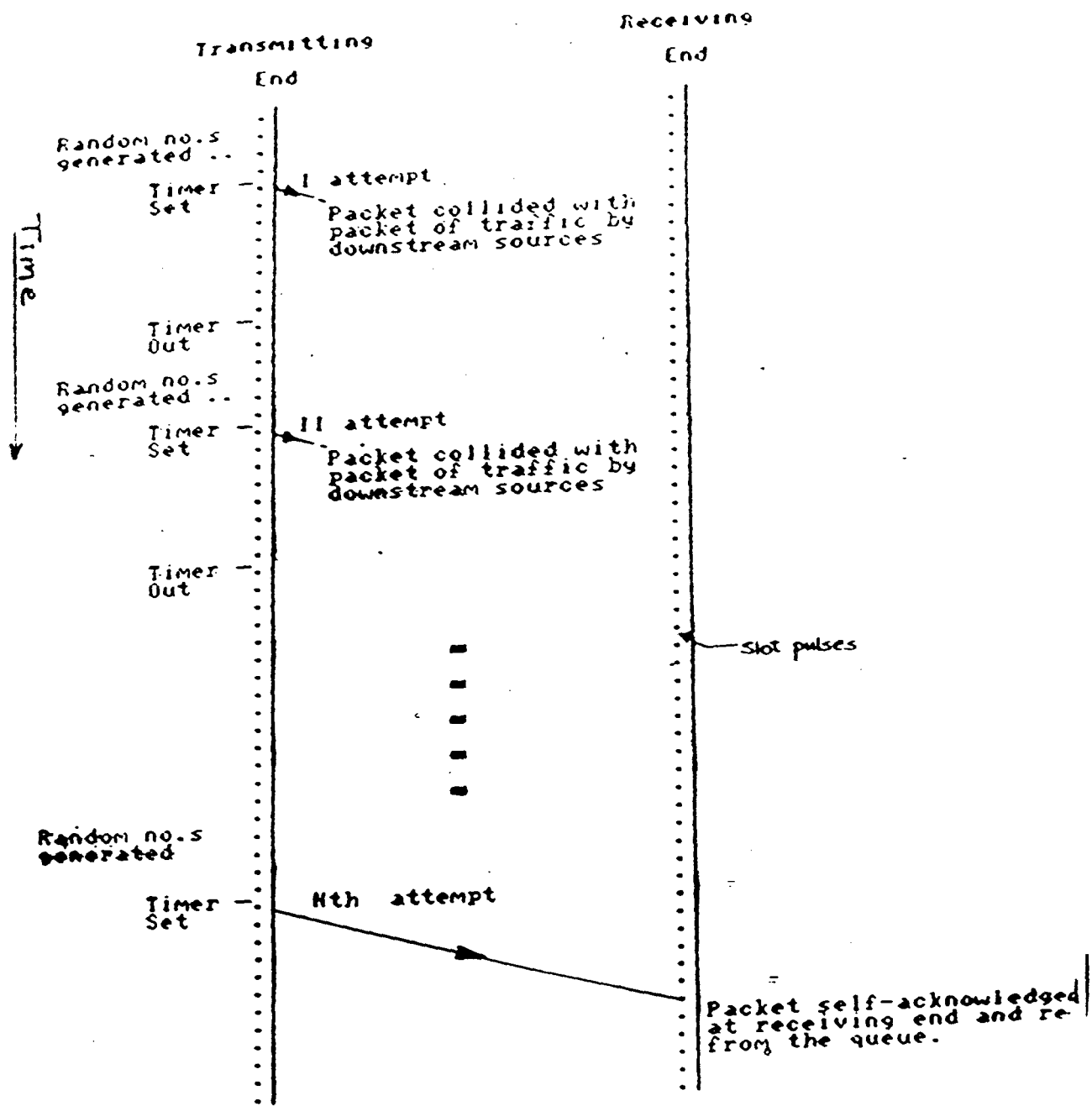


FIG. 5
 CHANNEL ACCESS USING E.T.S.
 AND C.A.P. SCHEMES

3.7 MULTIMEDIA TRAFFIC

Multimedia computers offer three kinds of data to the network. These are video data, voice data, ordinary data (text, graphics). These are categorise on the basis of bandwidth requirements, nature of bitstream, nature of traffic and delay constraints (Appendix B).

3.7.1 VIDEO TRAFFIC

Video communication on a multimedia computer network offers a steady, compressed bitstream with tightly bounded delay requirements. The refreshing rate of a video screen is 30 frames per second, so the time gap between two consecutive packets reaching the destination should not be more than 33ms. Uncompressed video needs bandwidth requirement of 90 Mbps but Intel's digital video interactive technique enables full motion video to be transmitted at a rate of 1.5 Mbps.

Motion Picture Experts Group (MPEG) of International Standarads Organistaion (ISO) has suggested the following standards for compression of motion video and associated audio.

Compressed bit rate :	1.5 Mbps
Frame rate :	30 frames/sec without interlacing
Resolution of video :	352*240 pixels

Motion Picture Compression (MPC) is different from still picture compression. MPC makes use of the extensive frame to frame redundancy present in all video sequences.

Joint Photographic Experts Group (JPEG) of ISO has suggested standard for compression of still picture. Still pictures can be compressed at different bit rates starting from 0.25 bits/pixel to 2 bits/pixel. More the number of bits/pixel, better is the picture quality. This way by compressing a still picture, it can be converted to digital image computer data.

3.7.2 VOICE DATA TRAFFIC

Voice traffic is bursty in nature. Speech consists of alternate talkspurts and silence intervals. The average talkspurt length is 1.67ms and average silence interval is 1.33ms. The frequency range of speech is 20Hz - 4000Hz. So a sampling rate of 8Kbps (according to Nyquist's Theorem) is needed. Using 8 bits per sample gives a bandwidth requirement of 64Kbps. The upper bound on the time taken to deliver a speech sample to the listener after the instant at which it was generated is typically 170-200ms. A speech packet delayed more than this is worthless. Voice data can tolerate some loss of packets with unnoticeable or little degradation of the intelligibility of the received speech signal. The acceptable packet loss percentage is maximum one percent.

3.7.3 ORDINARY DATA TRAFFIC

Ordinary computer data consist of text or garaphic data. These data can tolerate long delays but they cannot tolerate any loss, as loss of a packet means corruption of the whole data. So data transmission requires a channel with low error rate with a bandwidth of 64Kbps.

3.8 PRIORITY TRANSMISSION

Video traffic has a tightly bounded delay constraint of 33ms. Voice traffic can tolerate a delay upto 170ms and ordinary data traffic has no upper bound on delay. Therefore these data are prioritised accordingly. The video traffic is given the highest priority, voice traffic comes next and ordinary data traffic has the lowest priority. At the transmitter end of the station, an output traffic queue is maintained. Any newly generated packet which has to be placed on the network is placed in the queue. This queue is a priority queue i.e. a new packet generated takes position ahead of any lower priority packets and behind all the packets of the same priority. So queue at any moment contains all the video packets at the top, followed by voice packets and then the ordinary data packets are placed. The

queue length is assumed to be infinite to prevent any data loss under heavy load conditions.

3.9 DELAYS ENCOUNTERED BY PACKETS

3.9.1 QUEUING DELAY

When a packet is generated inside the multimedia source, it is placed in the output queue at transmitter end at a position according to the priority of the type of data it contains. Starting from the time of generation of a packet to the time when it reaches the top of the queue (considered for transmission) for the first time, the delay is counted as queuing delay for this packet.

3.9.2 ACCESS DELAY

After a packet reaches the top of the queue, the source tries to send it on channel. It makes several attempts as shown in fig - 5, till it succeeds. Meanwhile, If a higher priority packet is generated it is placed ahead of this packet. Now transmission attempts for sending this packet are done only when the higher priority packet is successfully transmitted. Starting from the instant a packet reaches the top of the queue for the first time to the

instant it is successfully acknowledged at the receiving end of the source, this period is called access delay. This delay includes the data transmission time and delay for propagation across the network. The transmission delay component is equal to one slot length. The propagation delay is fixed for a source according to the length of fiber between the transmitter and receiver ends of the source. The remaining component is variable depending on the number of attempts in which the packet is transmitted, and the number of times it is pushed back by a high priority packet, once it reaches the top of the queue.

3.9.3 TRANSFER DELAY

It is the total delay suffered by a packet starting from the time at which it is generated and put up in the queue, to the time at which it is self acknowledged at the receiving end of the source. This is equal to the sum of the queuing delay and access delay.

3.10 PERFORMANCE INDICES

For testing the system's performance and estimating maximum capability, different workloads are applied. As single multimedia computer offers more or less constant

load, different workloads are applied by varying the number of active multimedia systems.

Performance indices for testing and analysing the system's performance are throughput, delays faced by packets in transfer across the network, and load supported by the network. Since due to bus architecture, the delays faced by packets are also dependent on the position of the source relative to other sources, the degree of fairness provided by the protocol to the upstream sources is an important performance index.

CHAPTER FOUR

SIMULATION

CHAPTER 4

SIMULATION

4.1 INTRODUCTION

Simulation is a powerful technique for solving a wide variety of problems. To simulate is to copy the behaviour of a system or phenomenon under study. Computer simulation allows us to mimic the behaviour of the real life system, however complex, and get a measure of its performance. Simulation is becoming increasingly popular in the class of dynamic systems like communication networks with random traffic inputs. Simulation provides the means to visualize a system that is not yet built, to analyse a system to determine critical elements and to act as design accessory in order to evaluate proposals. Simulations use simulating models and based on these, perform experiments which enable the analyst to determine the behaviour of a system.

From the view point of simulation there are two fundamentally different types of systems :

(1) Systems in which the state changes smoothly or continuously with time are called **continuous systems**.

(2) Systems in which the state changes abruptly at discrete points in time are called **discrete systems**.

4.2 CONTINUOUS SYSTEM SIMULATION

Continuous systems are those systems in which the state or the variables vary continuously with time. These systems are generally described by means of differential equations. If the set of differential equations describing a system are ordinary, linear and time invariant, an analytic solution is usually easy to obtain. Simulating the system often gives added insight into the problem besides giving the required numerical solution.

4.3 DISCRETE SYSTEM SIMULATION

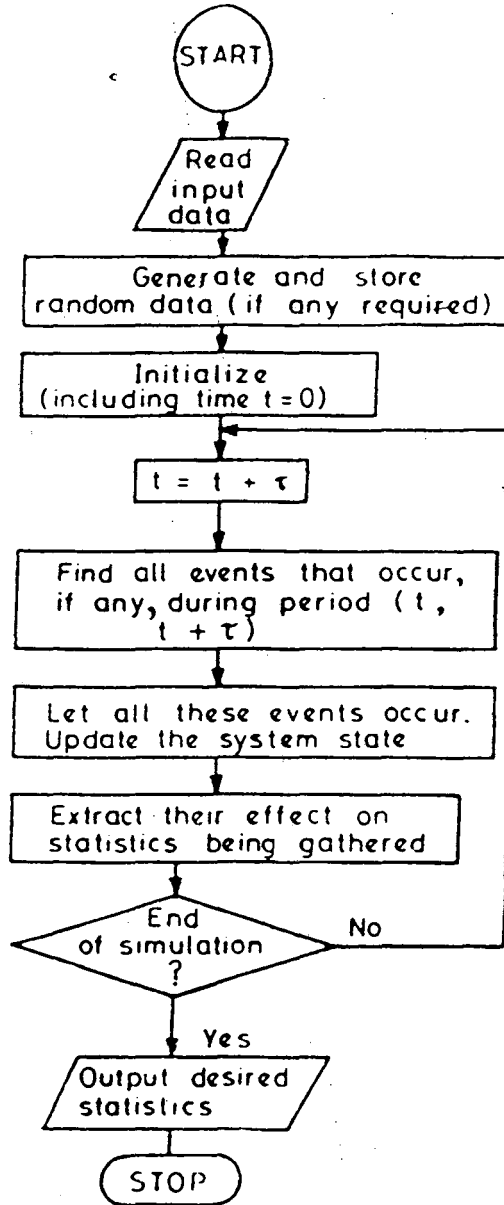
Discrete systems are those systems in which changes in the objects are discontinuous. Each change in the state of the system is called an event. Therefore the simulation of a discrete system is often referred to as **discrete - event simulation**. In simulating any dynamic system, continuous or discrete, there must be a mechanism for the flow of time. For we must advance time, keep track of the total elapsed time, determine the state of the system at the new point in time, and terminate the simulation when the total elapsed time equals or exceeds the simulation period. For continuous systems time is advanced in small increments of t for as long as needed. In simulation of discrete systems, there are two fundamentally different models for moving a system through time.

4.3.1 FIXED TIME STEP MODEL

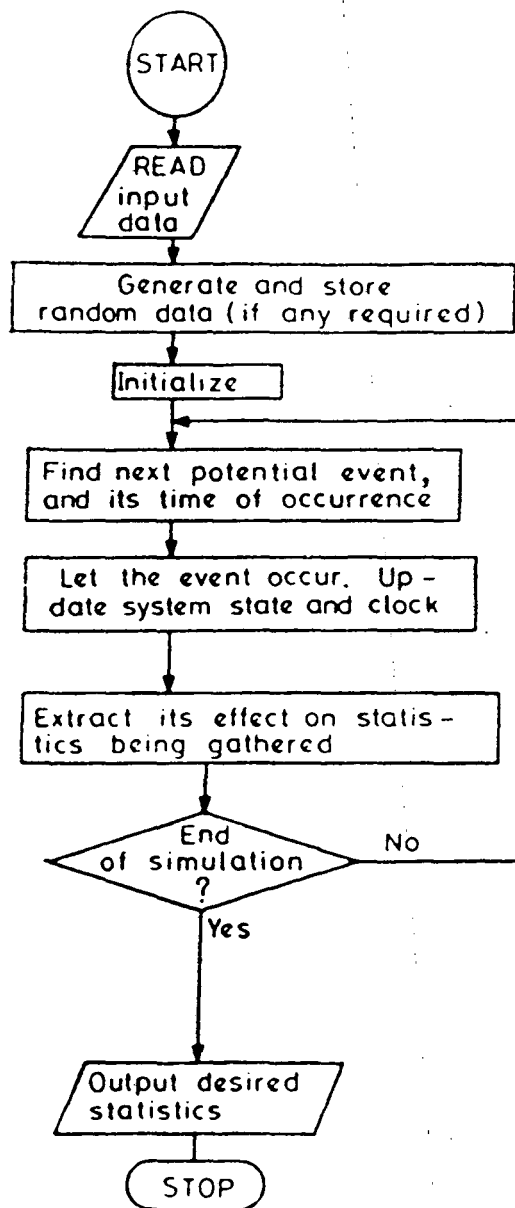
In time step model a timer or clock is simulated by the computer. This clock is updated by a fixed time interval t and the system is examined to see if any event has taken place during this time interval. All events that take place during this period are treated as if they occurred simultaneously at the tail end of this interval. The fixed time step simulation works as shown in flow chart # 3.

4.3.2 EVENT TO EVENT MODEL (NEXT EVENT MODEL)

In this simulation model the computer advances time according to the occurrence of the next event. It shifts from event to event. The system state does not change in between. Only those points in time are kept track of when something of interest happens to the system. Event to event model is preferred to fixed time step model because in this model no computer time is wasted in scanning those points in time when nothing takes place. This waste is bound to occur if a very small value of t is picked. On the other hand if t is so large that one or more events must take place during each interval then the model becomes unrealistic and may not yield meaningful results. The implementation of event to event model is more complicated than the fixed time step model. The event to event model is described in flow chart # 4.



FLOW CHART # 3
FIXED TIME STEP SIMULATION



FLOW CHART # 4
NEXT EVENT SIMULATION

4.4 STOCHASTIC SIMULATION

Discrete dynamic systems can be classified as deterministic or stochastic. The deterministic systems are less demanding computationally than the stochastic systems and are frequently solved analytically. Stochastic systems are systems in which atleast one of the variables are given by a probability function. There is inherent randomness or unpredictability in the system's behaviour. To simulate such random variables, we require a source of randomness. In simulation experiments, this is achieved through a source of uniformly distributed random numbers. These numbers are samples from a uniformly distributed random variable between some specified interval, and they have equal probability of occurrence. Stochastic simulation is of two types : static stochastic simulation and dynamic stochastic simulation.

STATIC STOCHASTIC SIMULATION

When the distribution of random numbers is stationary and the random samples are not co-related, the simulation is called static stochastic simulation.

DYNAMIC STOCHASTIC SIMULATION

In dynamic stochastic simulation initially the distribution of random numbers is not stationary and the random samples are co-related. This stage known as

transient, has to be crossed and only then the observation of results is started.

4.5 RANDOM NUMBER GENERATION

Random events can be simulated by generating random numbers on a computer. The random numbers follow a certain distribution.

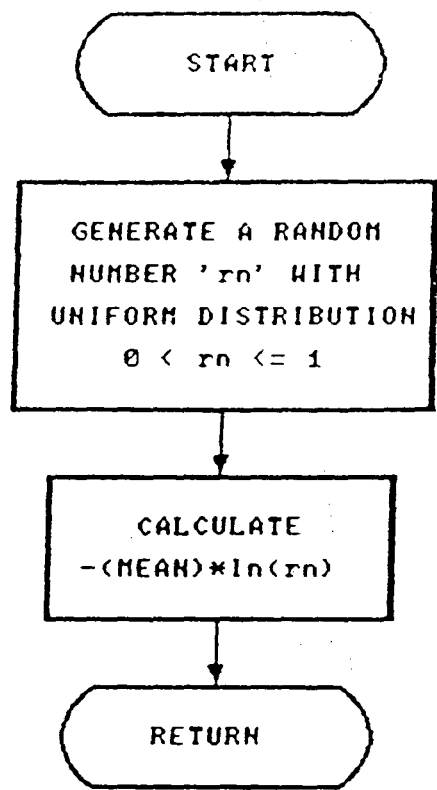
(1) **Exponential Distribution** : To generate an exponential distribution, first of all a random number 'r' is generated with uniform distribution i.e. the number lies between 0 and 1. Then calculate the value of E_i using the following equation.

$$E_i = -(\text{Mean}) * \ln(r)$$

E_i gives the instance of the desired exponentially distributed random variate. This process of generation of exponentially distributed random variate is described in flow chart # 5.

(2) **Poisson Distribution** : Poisson distribution is a discrete distribution in which the probability of an event occurring exactly "k" times during a time interval t is given by the probability mass function

$$G_k(T) = ((tL)^k * (1/(k!))) * (e^{(-t*L)})$$



FLOW CHART # 5
EXPONENTIALLY DISTRIBUTED
RANDOM VARIATE

Where L is the average number of times the event occurs in a unit period. The procedure required to get a poisson distributed random variate is to form the product of successive uniformly distributed random numbers, until the following equation is satisfied.

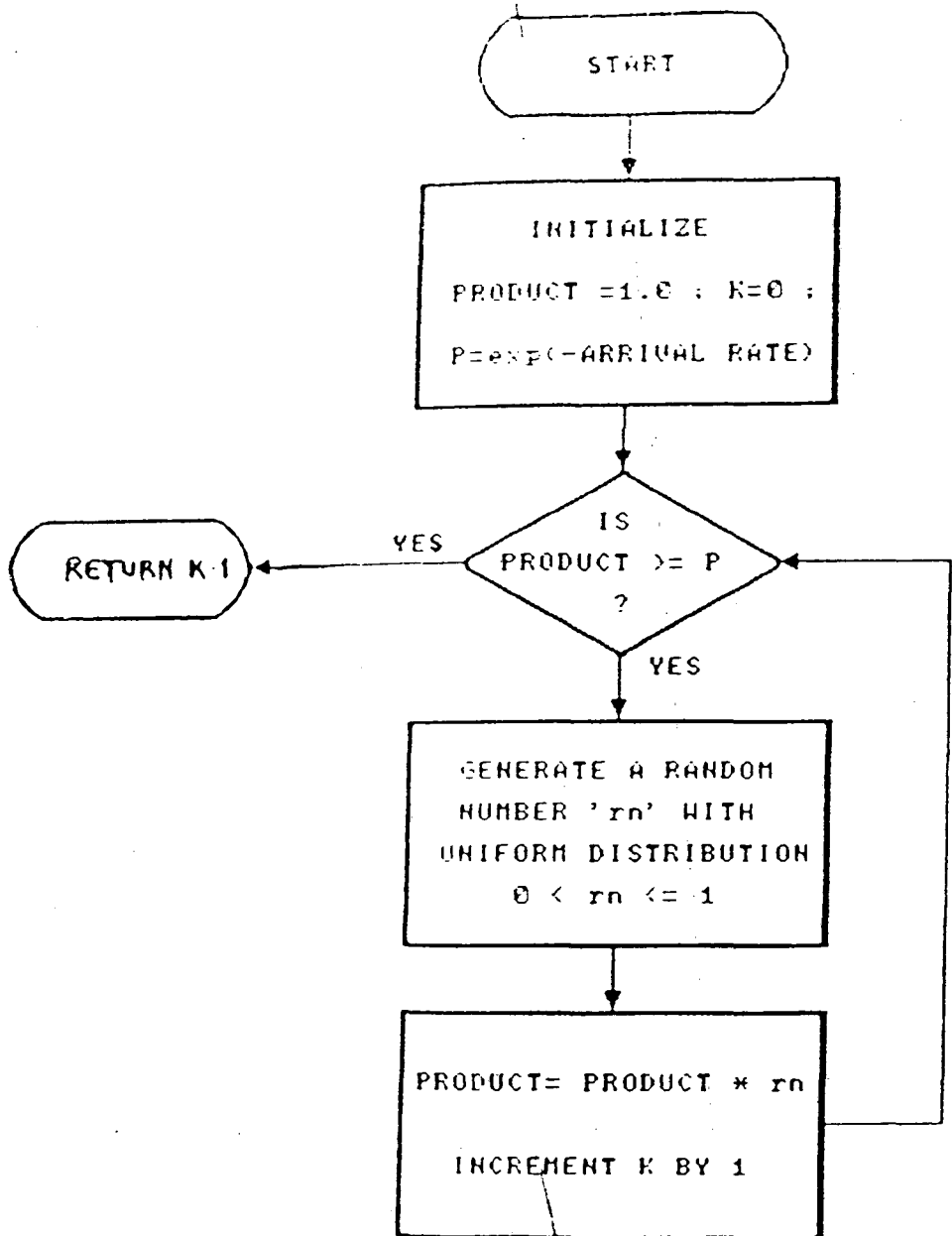
$$U_i < \exp(-L)$$

Where U_i is a uniformly distributed (0,1) random number, L is the mean of the distribution. The desired random variate instance N_i will be one less than the required number of uniformly distributed random numbers as shown in flowchart # 6.

4.6 LENGTH OF SIMULATION RUN

A simulation run is an uninterrupted recording of a system's behaviour under a specified combination of controllable variables. How long to run a simulation experiment to achieve a reasonable degree of confidence in the numerical results of the the experiment, is vital for validification of simulations involving randomness.

A problem specific variable entity is chosen as control variable. This control variable should have important bearing on the results of the simulation. The simulation run continues till the value of the chosen control variable stabilises. The check on the random number generator can



FLOW CHART # 6
 POISSON DISTRIBUTED
 RANDOM VARIATE

serve as a secondary method for the simulation length control. In experiments using random numbers, the sample mean (\bar{X}) should be as close to population mean (L) as possible. These become exactly equal when number of samples taken are infinite. However, this needs simulation to continue for infinitely long time which is not practical. In practice, simulation is allowed to run till a confidence level in the results is reached.

Confidence limit (t) : It is the permitted variation in the observed results from the theoretical results.

Confidence level : It is the probability that observed results are within the confidence limits.

Confidence level calculation : Consider a random variable X with mean L and standard deviation D . If $X_1, X_2, X_3, \dots, X_n$ are the samples taken, their average is

$$\bar{X}_{avg} = \frac{\sum X_i}{n}$$

and if $n \rightarrow \infty$, $(\bar{X}_{avg} - L) \rightarrow 0$

According to **Central Limit Theorem**, the sample mean \bar{X}_{avg} is itself a random variable with mean L and standard deviation $D/(\sqrt{n})$. The number of samples needed (n) is given by

$$n = \frac{Y^2 * D^2}{t^2}$$

For $(1 - \alpha) = 90\%$, $Y = 1.65$

Where t = confidence limit, Y is a standardised normal static for probability $(1 - \alpha)$ and D^2 is the variance of the samples. D^2 is not known in advance, so it is estimated as :

$$D^{2est} = \frac{\sum (X_i - \bar{X})^2}{(n - 1)}$$

So for static stochastic simulation follow the following steps :

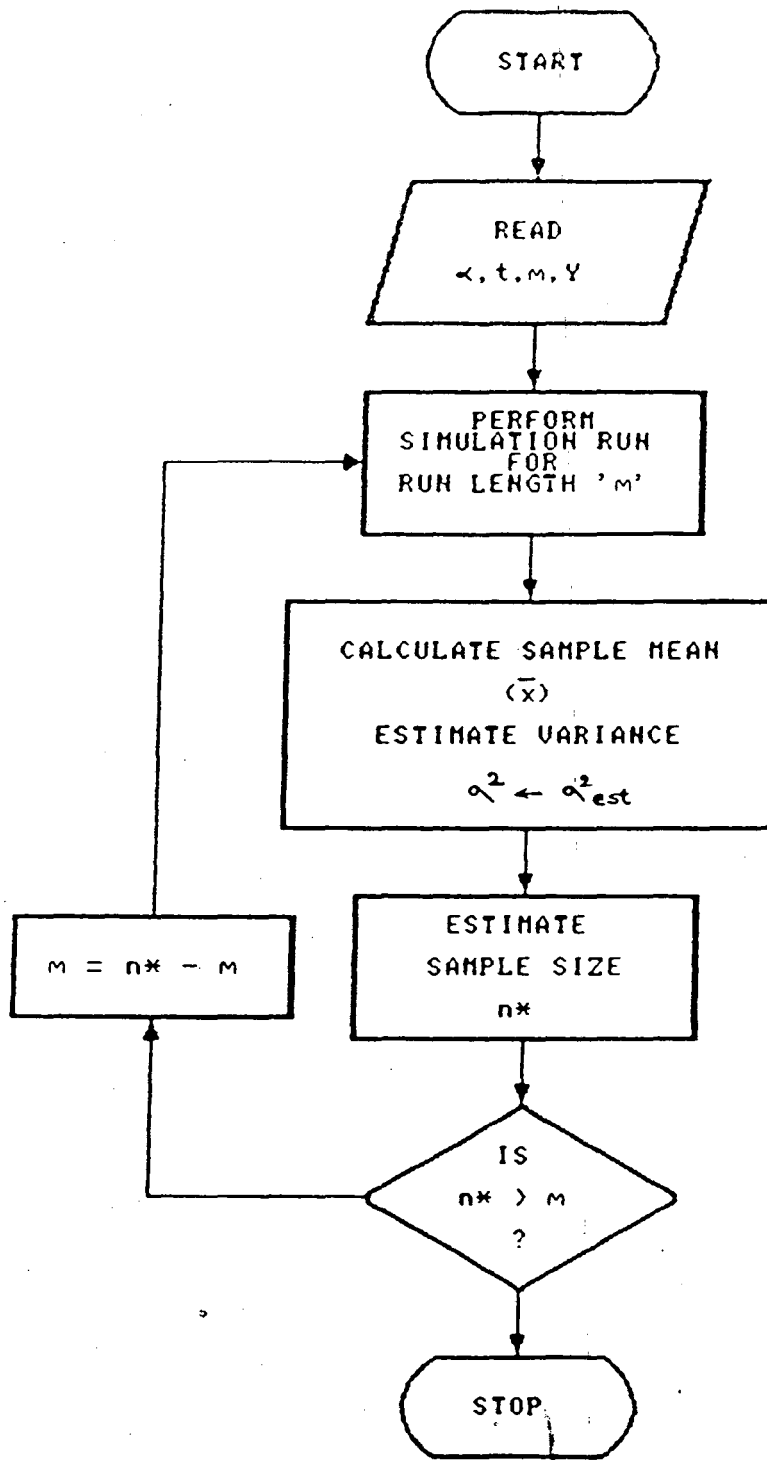
- (1) Simulate for $n_i = 2000$
- (2) Calculate D^{2est} and recalculate n (i.e. n_i)
- (3) If $n_{i+1} > n_i$, continue upto n_{i+1}
- (4) Repeat above steps till for some "j", $n_{j+1} \leq n_j$

This process is explained in flow chart # 7.

4.7 SIMULATION MODEL

Data slots moving on a unidirectional bus is a continuous system with continuous flow of information. However the state of the network changes only when a slot start pulse reaches the transmitting or receiving end of a source (Multimedia system). At these time instants, a source

- (1) Takes decision for accessing the channel.



FLOW CHART # 7
 ATTAINING CONFIDENCE LEVEL

(2) Checks the existing packet at receiving end, whether it is its own traffic, for self acknowledgement. If it is this packet is removed from the queue.

(3) Increments its internal clock used for generation of traffic packets.

This discrete event system based on a continual transmission of data by sources is simulated by fixed time step model. One time step is movement of slots train on the network by one small unit distance. Every new allignment of slot train on the network leads to some state changes due to activities of the active sources. These events are allowed to occur and system state is updated.

To simulate the actual behaviour of such a system as closely as possible, the whole simulation software is driven by a simulated slot pulse train movement and their allignment with active sources on the network. This approach has been used since their are many different events and their sequence of occurence affects the system behaviour. A simulated slot pulse movement gives an effective solution for the sequencing of potential events.

In this simulation, initially, the random number generator is warmed up and its first thousand random outputs which are generally co-related are ignored. This fulfills the two preconditions for static stochastic simulation.

The network in the simulation is also warmed up to its steady state. This is done by ignoring the packet delay results obtained during first one thousand slots that pass across the bus in a simulation run.

The control variable chosen is the average transfer delay for the video data. The video data is the traffic on the network. Hence choice of this entity for controlling the length of the simulation is justified. After every thousand video packets transferred across the network, new value of the average transfer delay is recorded. Confidence level estimation is done on these values obtained. The confidence level estimation on the random number generator proceeds simultaneously. Whenever any of these two checks indicate that the simulation has achieved maturity stage, the simulation run is stopped and the results are recorded. The confidence level chosen for the random number generator check is 90% with confidence limit 0.05.

4.8 SIMULATION OF MULTIMEDIA TRAFFIC

VIDEO TRAFFIC

Simulation of video packet is done by periodic generation of a video data packet. This period conforms to the bandwidth requirement of video traffic and the bandwidth supported by the underlying optical fiber network. Maximum

communication delay allowed for a video data packet is 30ms. More than one percent of total video should not face communication delay more than this. Otherwise the service offered by the network is unsatisfactory.

VOICE TRAFFIC

Bursty voice data is modelled by poisson arrival of data packets. The average bandwidth requirement is 64Kbps. One talkspurt of this traffic is accomodated in one packet of the network. Because of fixed length of packets, some capacity in packet is unutilized by voice traffic while the packet length (2000 bits) exactly fits the requirements of one packet of video traffic.

ORDINARY DATA TRAFFIC

Ordinary data traffic is also simulated by poisson arrival. The average bandwidth requirement of the data is taken to be 64Kbps, which is equal to approximate requirement of data retrieval on demand.

CHAPTER FIVE

PERFORMANCE ANALYSIS

CHAPTER 5

PERFORMANCE ANALYSIS

In this chapter the performance analysis of the protocol designed is carried out. The system is simulated on the computer and the simulation results are analyzed for delays (Queuing delay, Access delay and Transfer delay), throughput of the system and load supported by the network. For each of the above mentioned performance indexes, effects of fairness schemes (C.A.P. and E.T.S.) are also analyzed. Graphs are plotted for analyzing the simulation results. These graphs include the queuing delay, access delay and transfer delay faced by packets of different data types for different number of active stations. Number of active stations supported by the network on applying different fairness techniques is also described. The effect of fairness techniques on individual delays of sources is analyzed. The channel access probability (CAP) assigned to multimedia sources can be equal for all sources, this case is indicated by <CAP : OFF> in the graphs. If the channel access probability is assigned according to the relative positions of the sources, then in the graphs it is indicated by <CAP : ON>. Equal Timer Setting (ETS) is implemented on top of <CAP : ON> assignment. Whenever it is used it is indicated by <ETS : ON>. The delays are expressed in terms

of number of slots where one slot has duration of 13.33 micro seconds.

5.1 DELAYS

In the previous chapters we had discussed the various kinds of delays a packet faces. The time it takes from the time of generation of a packet till it reaches the top of the queue for the first time is called Queuing Delay. Access Delay is the time gap between when the packet reaches the top of the queue for the first time till it is self acknowledged. Transfer Delay is the sum of queuing delay and access delay. Queuing delay varies with the nature of the packet. Video packets are given highest priority in the prioritized output queue, next comes the voice packet and ordinary data has the least priority. These priorities have been assigned according to the maximum delay constraints of various data traffic.

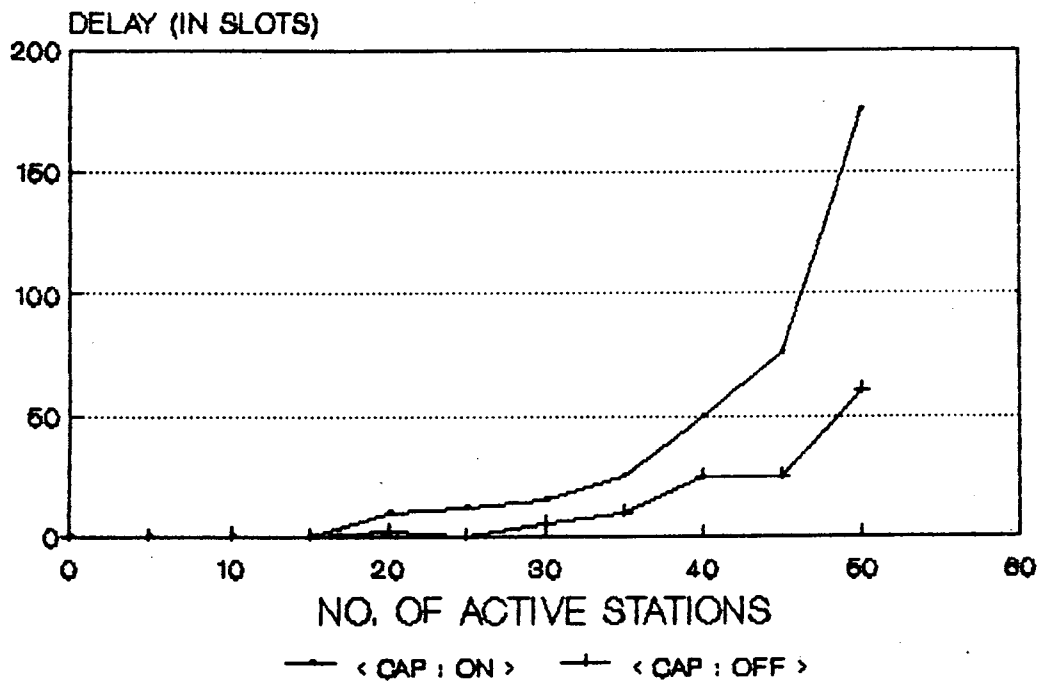
5.1.1 QUEUING DELAY

Queuing delay depends on the nature of the packet. Queuing delay for different data traffics is calculated and results are analyzed from the graphs.

Video Packet

Video packets face minimum queue delay as they have the highest priority in the queue. Graph #1 and #2 show the per

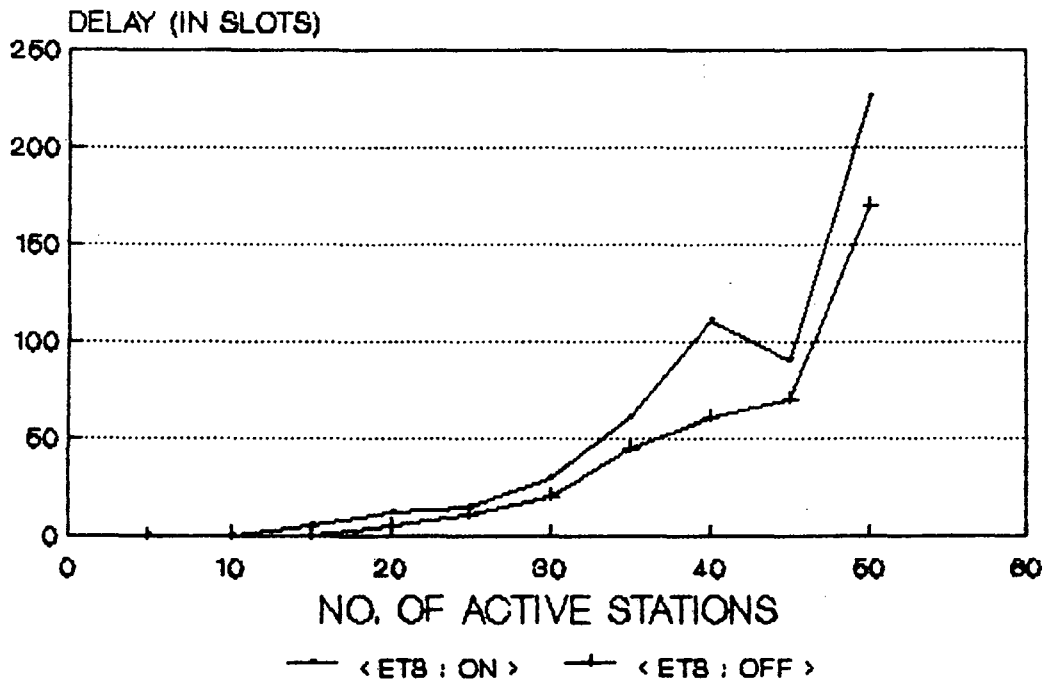
QUEUING DELAY VIDEO DATA



Graph #1

QUEUING DELAY

VIDEO DATA < CAP : ON >



Graph #2

packet queuing delay faced by video packets plotted for different number of active stations, Graph #1 shows the effect of CAP assignment on the queuing delay and graph #2 shows the effect of ETS implemented when CAP is ON.

Case 1 : <CAP : OFF> <ETS : OFF>

The queuing delay is negligible for low loads i.e. till number of active stations is less than 25. As the load increases, queues are formed and result in steep rise in queuing delay.

Case 2 : <CAP : ON> <ETS : OFF>

In this case the queuing delay is negligible only till twelve stations, after that the delay rises very fast. This is because when number of active sources increases, the probability assignment changes. The probability decreases sharply (exponentially) so the number of slots left free by the sources increases, resulting in increase in queue length and hence queuing delay.

Case 3 : <CAP : ON> <ETS : ON>

The queuing delay in case of < ETS : ON > is higher than in case of <ETS : OFF> because on applying ETS, the wait before making a retransmission attempt increases for all the active sources except the most upstream source. So in general queue length increases.

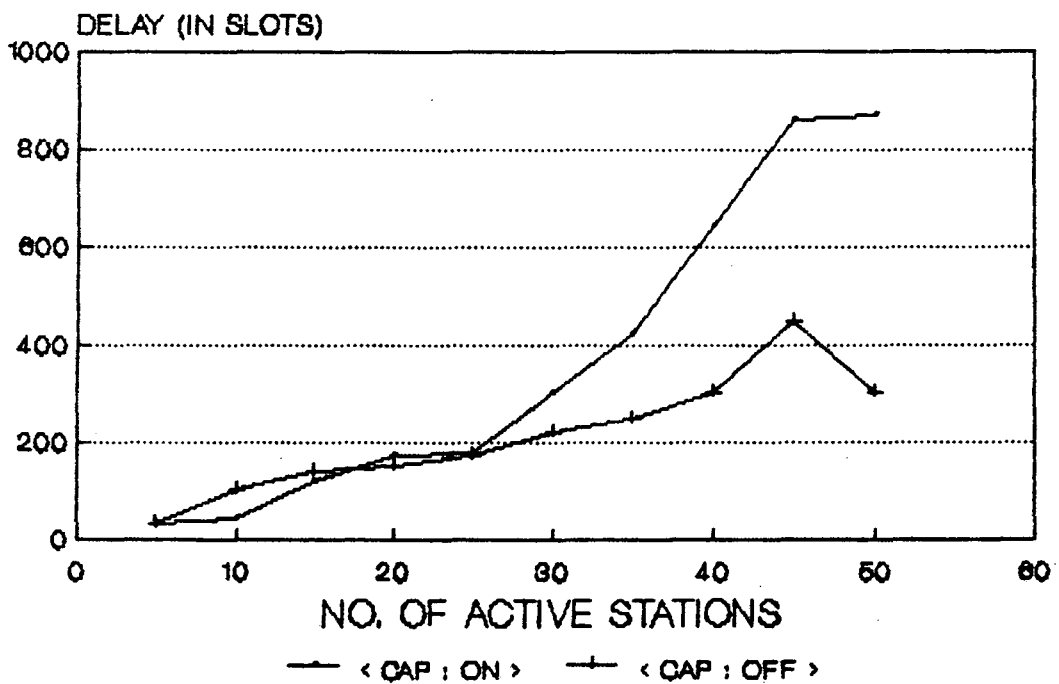
Voice Packet

Voice packets have lower priority than the video packets and higher priority than the ordinary data packets. Graphs #3 and #4 describe the queuing delay faced by these packets. They show the queuing delay for voice packets plotted for different number of active stations.

Graph #3 : The effect of CAP assignment is shown in this graph. The queuing delay for the <CAP : OFF> is smaller than that for <CAP : ON>. At loads below 20 active stations the delay is smaller in <CAP : ON> than in <CAP :OFF>.

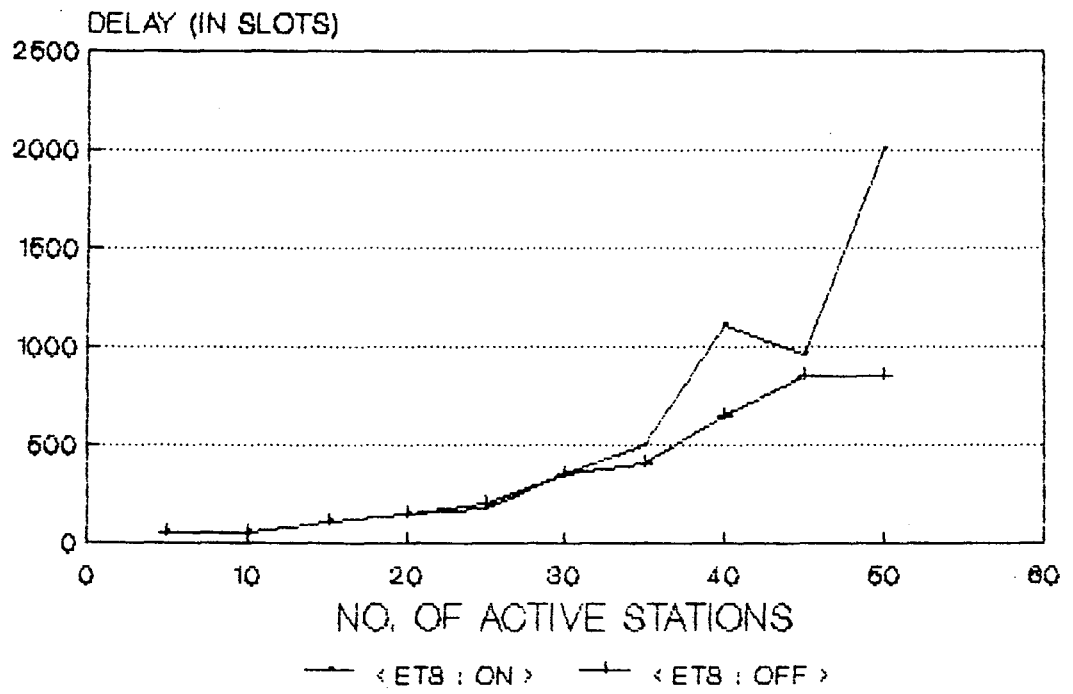
Graph #4 : This graph shows the queuing delay of voice packet plotted for different number of active stations. It shows the effect of ETS technique on queuing delay. The CAP assignment is graded. The queuing delay in <ETS : ON> is much higher than in <ETS :OFF> technique. At lower loads (i.e. till number of active stations is less than 30) the queuing delay is same in both <ETS : ON> and <ETS : OFF> but as number of active stations increases further, The queuing delay in <ETS : ON> scheme encounters a steep rise. This is because under heavy load conditions, the number of retransmission attempts increases, as collisions increase. In <ETS : ON> each source has to wait for a long time for retransmission.

QUEUING DELAY VOICE DATA



Graph #3

QUEUING DELAY VOICE DATA < CAP : ON >



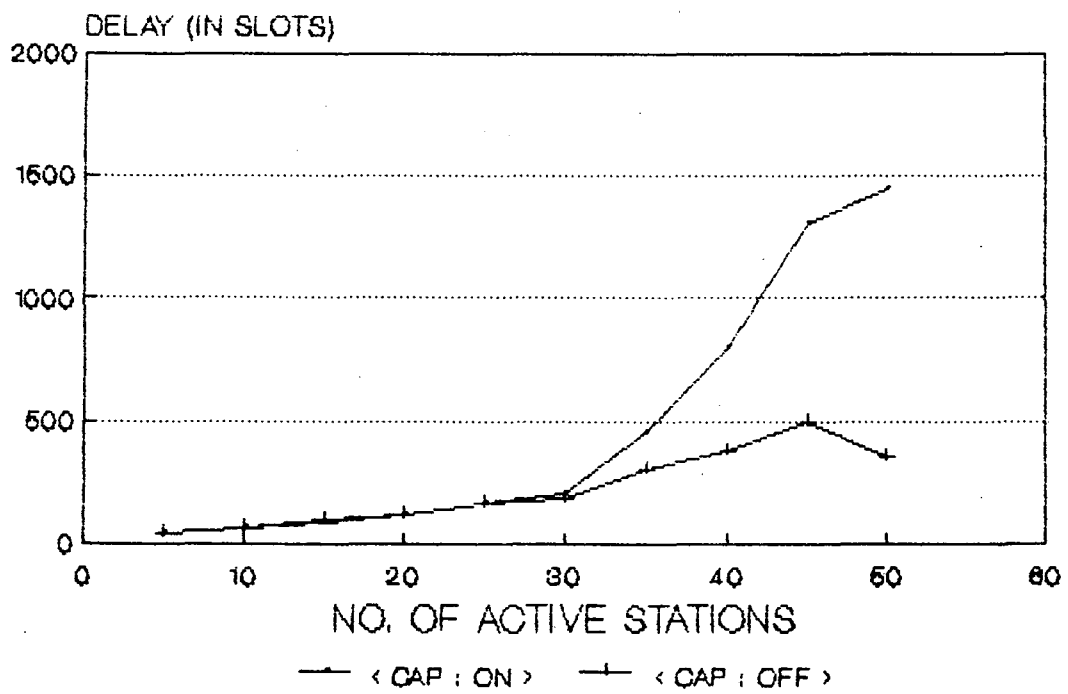
Graph #4

Ordinary Data Packet

Ordinary data is placed next to the existing video and voice data in the queue of a source station. So an ordinary data packet is transferred only when there is no video or voice packet present in the queue. Any new video and voice packets are put ahead of the already existing ordinary data packets in the output queue. Graphs #5 and #6 describe the queuing delays faced by ordinary data packets as the number of stations increases.

Graph #5 : This graph shows the effect of CAP assignment on queuing delay for ordinary data plotted for different number of active stations. The nature of the graph is similar to that for voice data (graph #3). The same explanation is valid for this graph also. But the absolute values of the delays are more for ordinary data than for voice data or video data. This is because ordinary data has least priority in the output queue. Queuing delay in <CAP : ON> and <CAP : OFF> is same under low load conditions (i.e. upto 25 stations) after that a steep rise in queuing delay in <CAP : ON> is encountered. This is because as number of active stations increase, the channel access probability assigned to them decreases exponentially and because of this size of queues increases, increasing the queuing delay.

QUEUING DELAY ORDINARY DATA



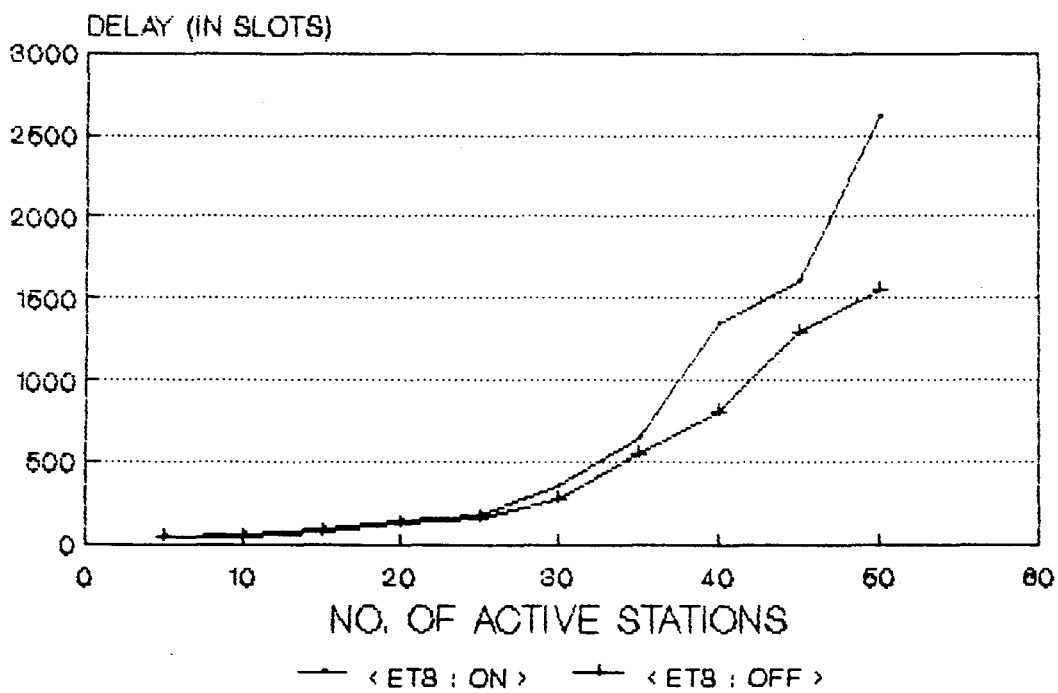
Graph #5

Graph #6 : This graph shows the effect of ETS technique on queuing delay for ordinary data plotted for different number of active stations. The CAP assignment is graded. Upto 25 stations, the queuing delay in case of <ETS : ON> is almost the same as that in case of <ETS : OFF>. At loads higher than this, the delay in case of <ETS : ON> is much higher than in case of <ETS : OFF>. This is because queues in the case of <ETS : OFF> clear off quickly. The wait time between successive retransmission attempts is smaller in case of <ETS : OFF>.

5.1.2.ACCESS DELAY

Starting from the instant a packet reaches the top of the queue for the first time to the instant it is successfully self acknowledged at the receiving end of the source, this period is called the access delay. This delay includes the data transmission time and delay for propagation across the network. The transmission delay component is equal to one slot length. The propagation delay is fixed for a source according to the length of fiber between the two ends of the source. The remaining component is variable depending on the number of attempts in which the packet is transmitted.

QUEUING DELAY ORDINARY DATA < CAP : ON >



Graph #6

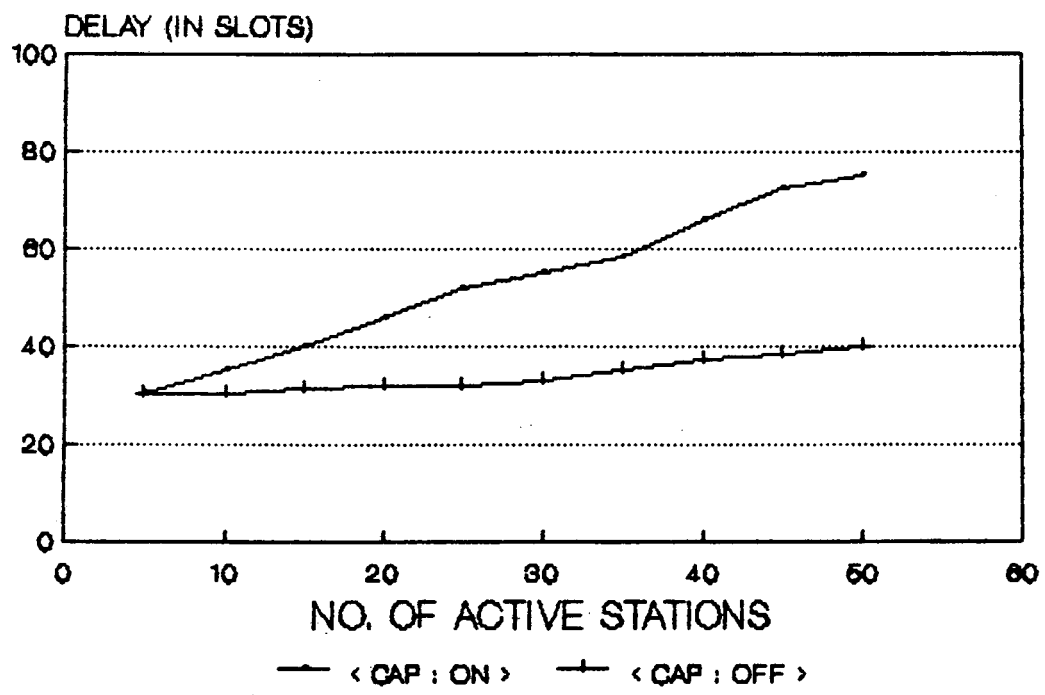
Video Packet

When a video packet comes at the head of the queue, it gains the first preference. It becomes the first packet to be transmitted by the source station. As video packets have highest priority in the queue, the access delay depends only on the no. of transfer attempts needed before the packet is successfully transmitted. Graphs #7 and #8 show the access delay of video packet plotted for different number of active stations.

Graph #7: This graph shows the effect of CAP assignment on Access delay for video data plotted for different number of active stations. The access delay under <CAP: OFF> is almost same for increasing number of stations. In case of <CAP: ON>, the rise in access delay with increasing load is faster than in the case of <CAP: OFF>. Since, as the number of active stations increases, the probability values assigned to sources decreases, resulting in more and more number of slots left free in wait by active sources. This results in larger access delays and their faster increase with increasing load.

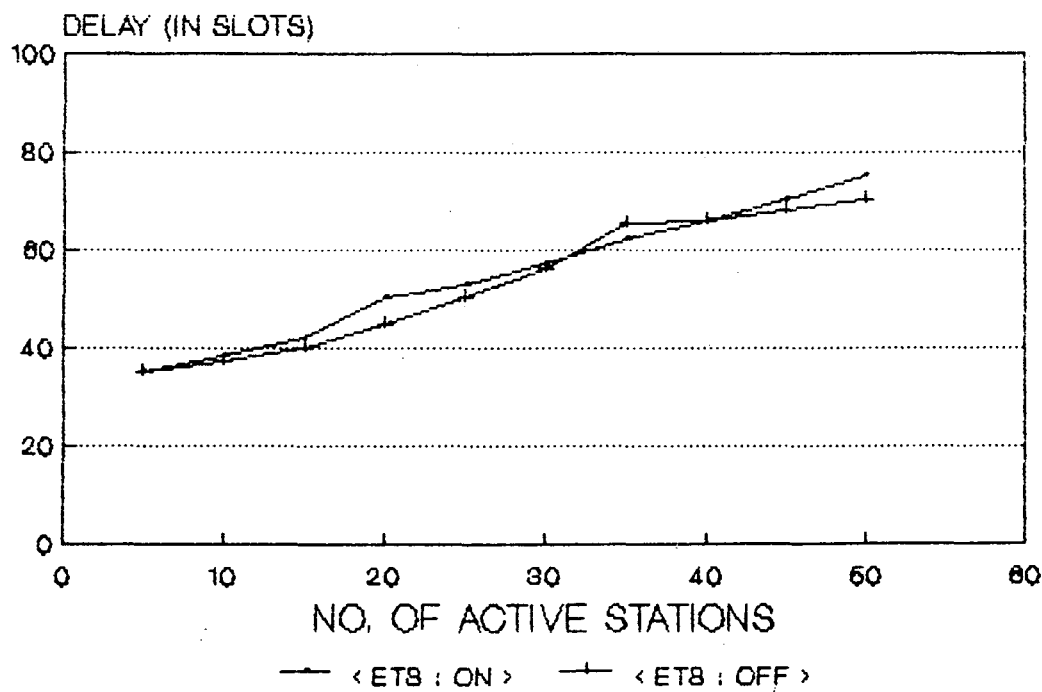
Graph #8: This graph shows the effect of ETS technique on the per packet access delay plotted for different number of active stations. The access delay in <ETS: ON> case is slightly larger than in <ETS: OFF> case. In <ETS: ON> case,

ACCESS DELAY VIDEO DATA



Graph #7

ACCESS DELAY VIDEO DATA < CAP : ON >



Graph #8

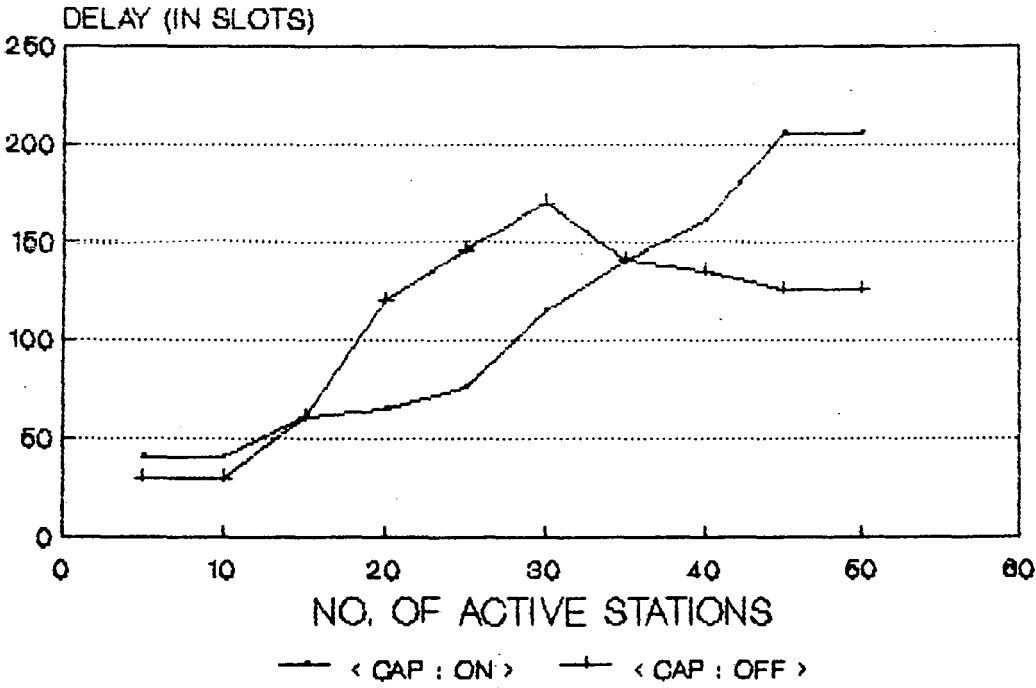
access delay curve shows a slow and steady increase with increasing load.

Voice Packet

Once the voice packet reaches the top of the queue, the access delay starts till the packet is self acknowledged at the receiver end. As the voice packet is second in the priority list of the output queue, even after it reaches the top the queue, a video packet can supersede it and occupy the top of the queue position. Therefore the access delay for voice packets depends upon the no. of retransmission attempts needed for successful transmission and the frequency with which video packets are generated, as they can supersede the transmission of voice packets. Graphs #9 and #10 show the access delay of voice packets plotted for different number of active stations.

Graph #9: Graph #9 shows the effect of CAP assignment on access delay for voice data plotted for different number of active stations. Initially upto 35 active stations access delay for <CAP: OFF> case is higher than the access delay for <CAP: ON> case. This is because of the equal mix of traffic from upstream and downstream sources in the transferred packets and since the upstream sources have high delays in <CAP: OFF> case. But at higher loads, again due to domination of channel by the downstream sources, the access delay for <CAP: OFF> case starts reducing. The access delay

ACCESS DELAY VOICE DATA



Graph #9

for <CAP: ON> case increases slowly upto 45 active stations and after that it is nearly constant upto 50 active stations. The domination stage in this case is postponed to higher load than the <CAP: OFF> case, where it is only 35 active stations.

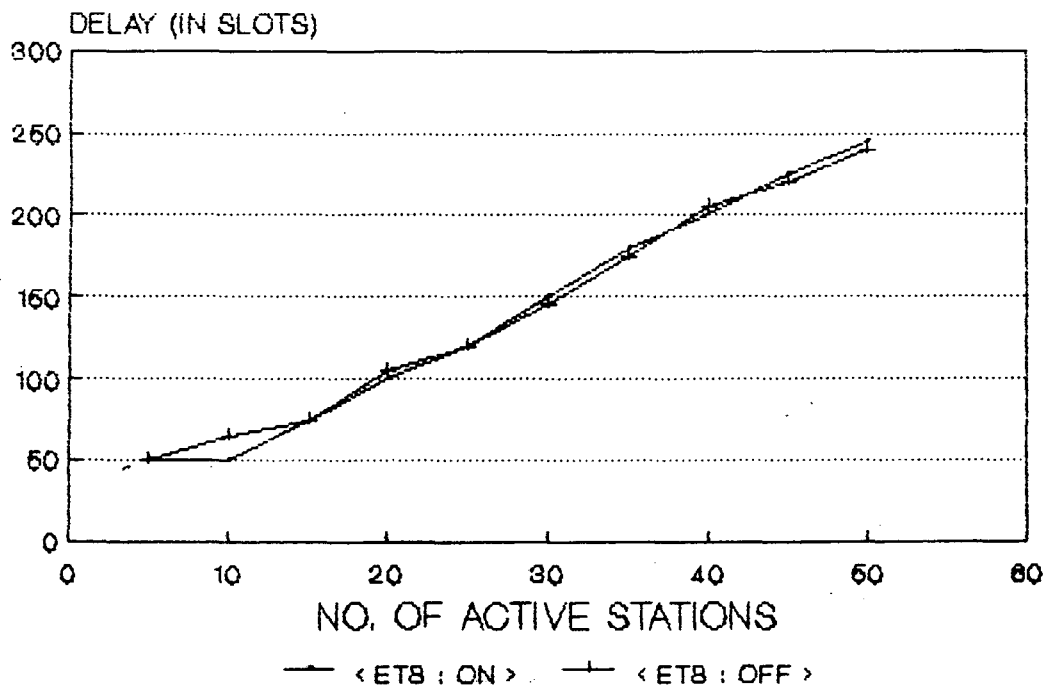
Graph #10: This graph shows the effect of ETS technique on the per packet access delay plotted for different number of active stations. The CAP assignment is graded. The access delay under <ETS: ON> and <ETS: OFF> is almost same. The access delay is almost constant till the number of active stations is less than 30 stations. After that there is a smooth rise in delay which stabilizes at loads higher than 50 active stations.

Ordinary Data Packet

The absolute value of access delay is more for ordinary data than for voice data. This is because when an ordinary data packet reaches the head of the queue, a newly generated video or voice packet, if any is placed ahead of it. Graphs #11 and #12 describe the access delays faced by ordinary data packets as the number of station increases.

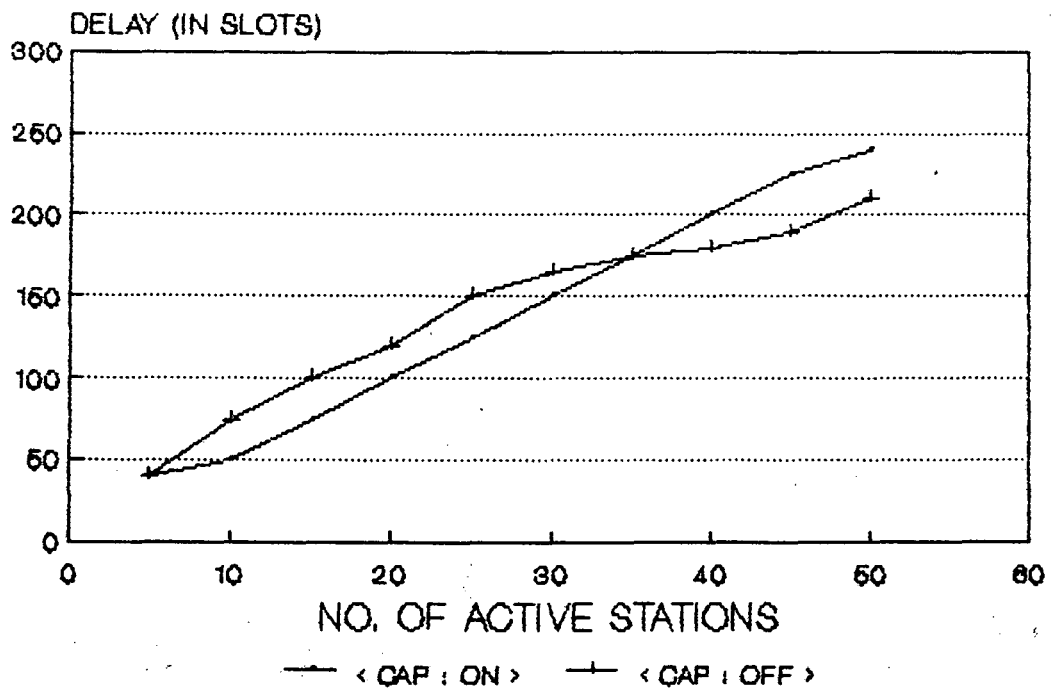
Graph #11 : This graph shows the effect of CAP assignment on access delay for ordinary data plotted for different number of active stations. In <CAP: OFF> and <CAP:ON>, till 32 active stations, the access delay is almost same for both

ACCESS DELAY VOICE DATA < CAP : ON >



Graph #10

ACCESS DELAY ORDINARY DATA



Graph #11

and constant. Then the access delay in <CAP: ON> increases whereas the access delay in <CAP: OFF> remains more or less constant under heavy load conditions also.

Graph #12: This graph shows the effect of ETS technique on the per packet access delay for ordinary data. The CAP assignment is graded. The access delay in the case of <ETS: ON> is same as that in case of <ETS: OFF>. So the ETS technique does not effect the access delay much. The access delay shows increase as the no. of active stations increase. This is because collision rate increases with increase in traffic, this calls in for increased number of retransmission and hence increase in access delay.

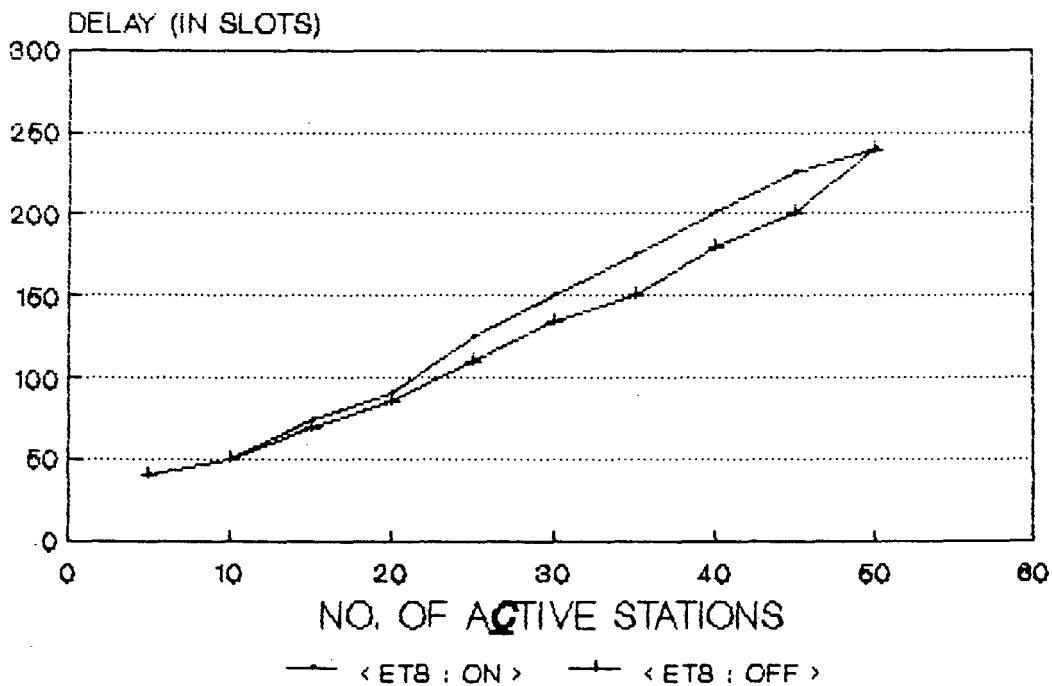
5.1.3 TRANSFER DELAY

Transfer Delay is the sum of access delay and queuing delay. The total time taken from the time a packet is generated, till it is successfully transmitted is called Transfer Delay.

Video Packet

Graphs #13 and #14 show the per packet transfer delay faced by video data packets plotted for different number of active stations. They show the effect of channel access probability assignment and equal timer setting technique on the transfer delay.

ACCESS DELAY ORDINARY DATA < CAP : ON >

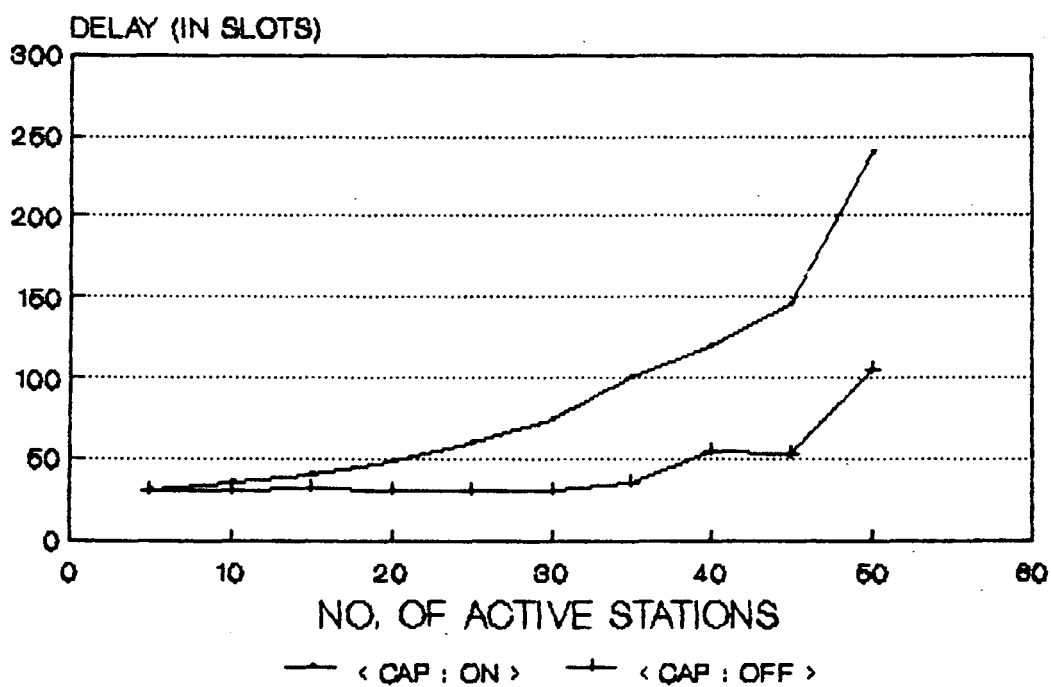


Graph #12

Graph #13: This graph shows the effect of channel access probability assignment on the transfer delay for different number of active stations. When CAP assignment is OFF, as the load increases, the delay increases initially very slowly and at higher loads, it increases sharply. When CAP is ON i.e. channel access probabilities are different for different stations, the delay rises sharply with increasing load and at high load the increase is steep. The delay in case of <CAP: ON> is higher than in case of <CAP : OFF>. This is because, when numbers of active multimedia systems increases, the probability assignment changes. The probabilities decrease sharply so the number of slots left free by the sources increases fast. This results in fast increase in the delays faced by the packets.

Graph #14: This graph shows the effect of Equal Timer setting technique on the per packet transfer delay of video data. The channel access probability is graded. On application of ETS, average transfer delay increases. The delay rises faster than in <ETS: OFF>. The increase at higher loads (around 50 active stations) is steep. This is because the wait before retransmission attempt is much higher than in case of <ETS OFF>. Every source, whatever be the gap between its two ends, has to wait for a fixed number of slots before making retransmission attempt in case a packet is not self acknowledged within timer interval. The

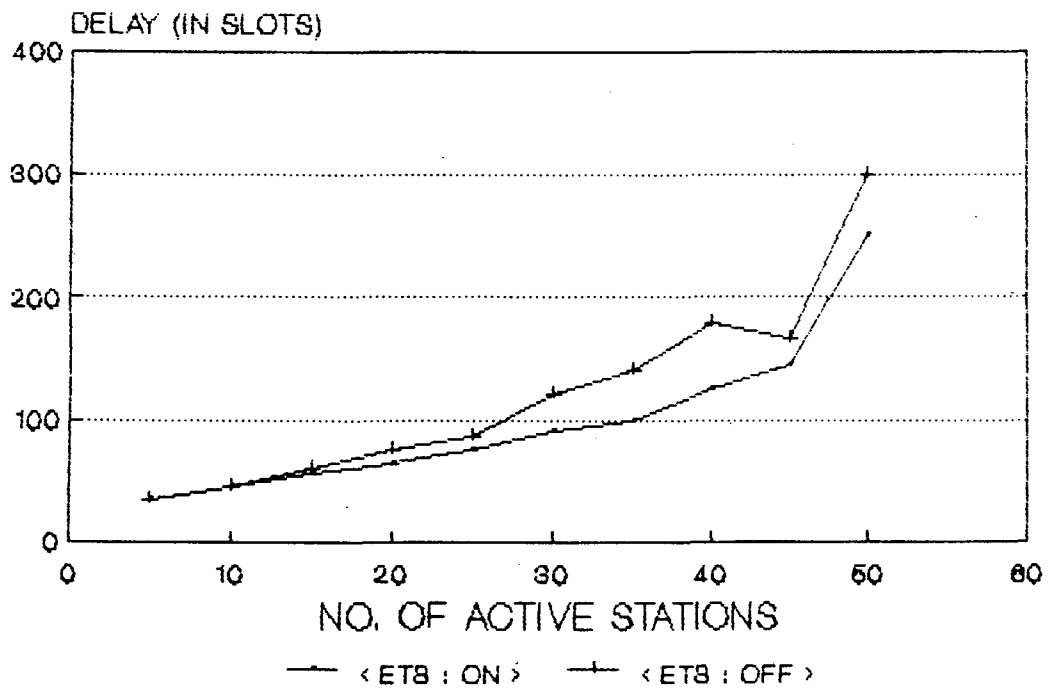
TRANSFER DELAY VIDEO DATA



Graph #13

TRANSFER DELAY

VIDEO DATA < CAP : ON >



Graph #14

timer value set is equal to gap between two ends of the most upstream source.

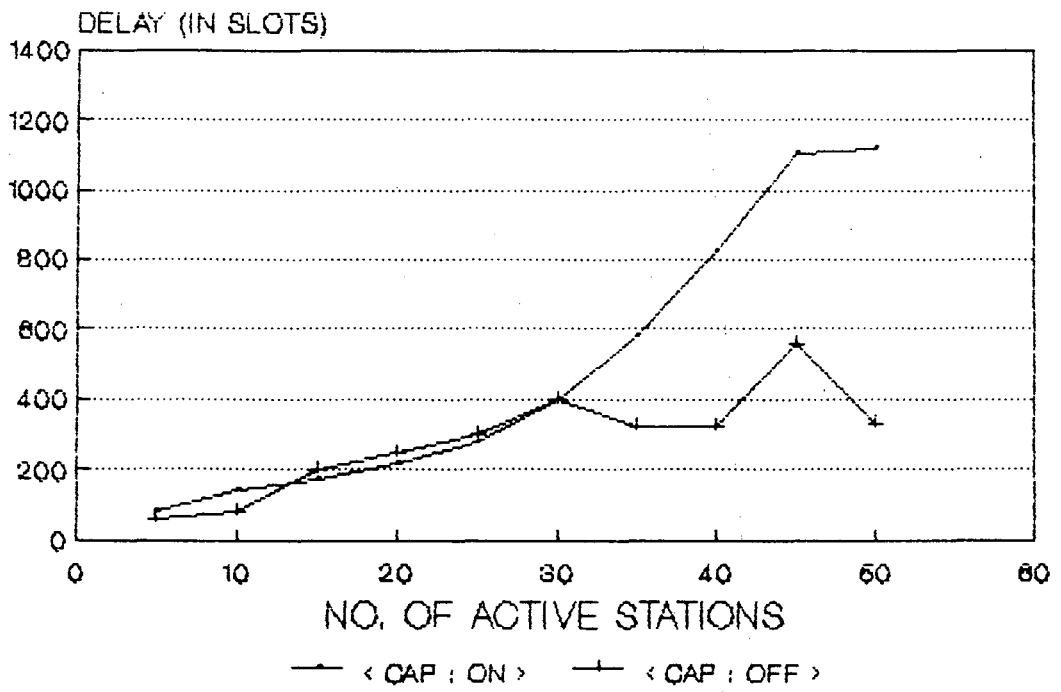
Voice Packet

Voice data is second in priority to video data in the output queue of multi-media sources. Graphs #15 and #16 show the per packet transfer delay for voice data plotted for different number of active stations. They show the effect of channel access probability assignment and Equal Timer setting techniques on the transfer delay.

Graph #15: This graph shows the effect of channel access probability assignment on the transfer delay for different number of active stations. When CAP is OFF, the delay increases slowly upto 25 active stations and then with some fluctuations stabilizes for higher load when CAP is ON, the average transfer delay increases with offered load. Initially upto 30 stations, it rises slowly but later upto 45 active stations, it rises steeply and then for further load, the delay starts stabilizing as now the downstream sources dominate the channel and have very low transfer delay, so average transfer delay stabilizes.

Graph #16: This graph shows the effect of ETS technique on the per packet transfer delay for voice data. The channel access probability assignment is graded. Upto 30 stations, the transfer delay is same in both the cases. At higher

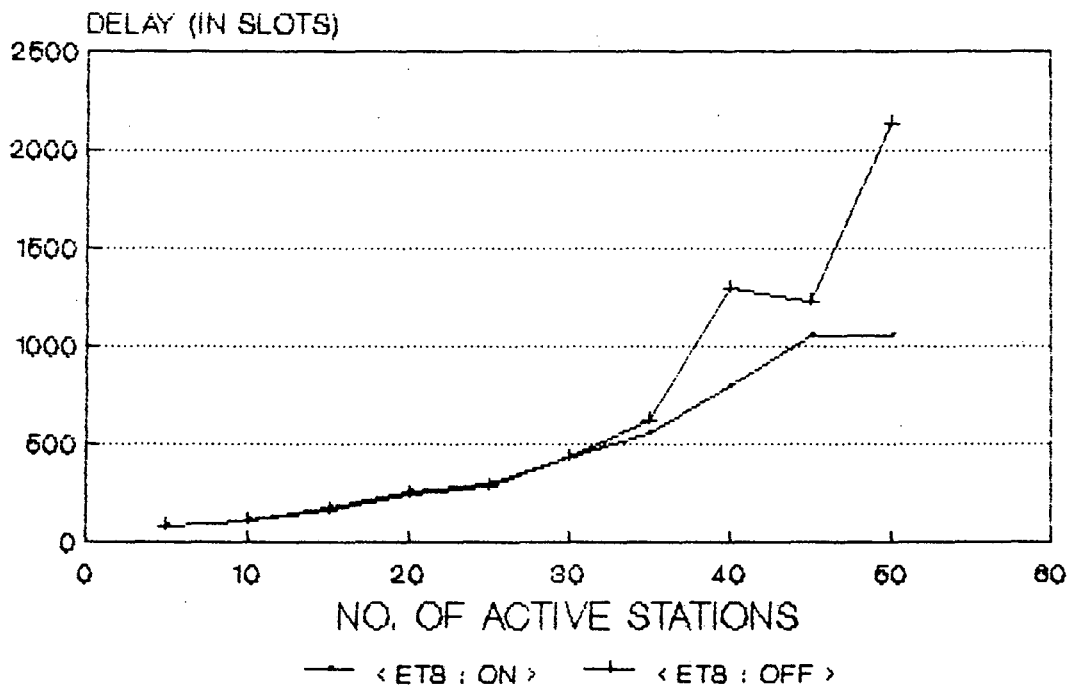
TRANSFER DELAY VOICE DATA



Graph #16

TRANSFER DELAY

VOICE DATA < CAP : ON >



Graph #16

loads, the <ETS:ON> case has higher transfer delay than in <ETS: OFF> case. The increase in transfer delay at higher loads is very steep.

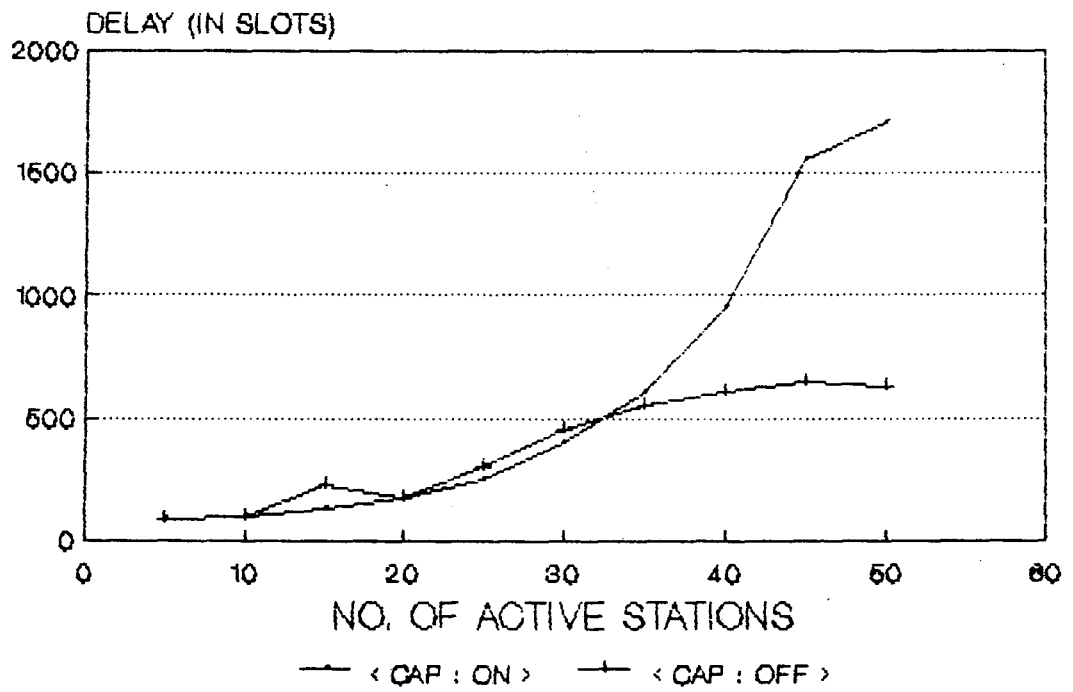
Ordinary Data

Ordinary data has the least priority in the output queue. So the ordinary packet is transmitted only when no video or voice packet is present in the queue. So, transfer delay of ordinary data packet is maximum as compared to video or voice data. Graphs #17 and #18 show the per packet transfer delay for ordinary data plotted for different number of active stations.

Graph #17: This graph shows the effect of channel access probability assignment on the transfer delay for different number of active stations. Upto 30 active stations, an ordinary data packet faces more transfer delay in <CAP: OFF> than in the case of <CAP: ON>. At higher loads (40 active stations) the delay value stabilizes under <CAP: OFF>. Under <CAP: ON> the delay rises steeply after 35 active stations.

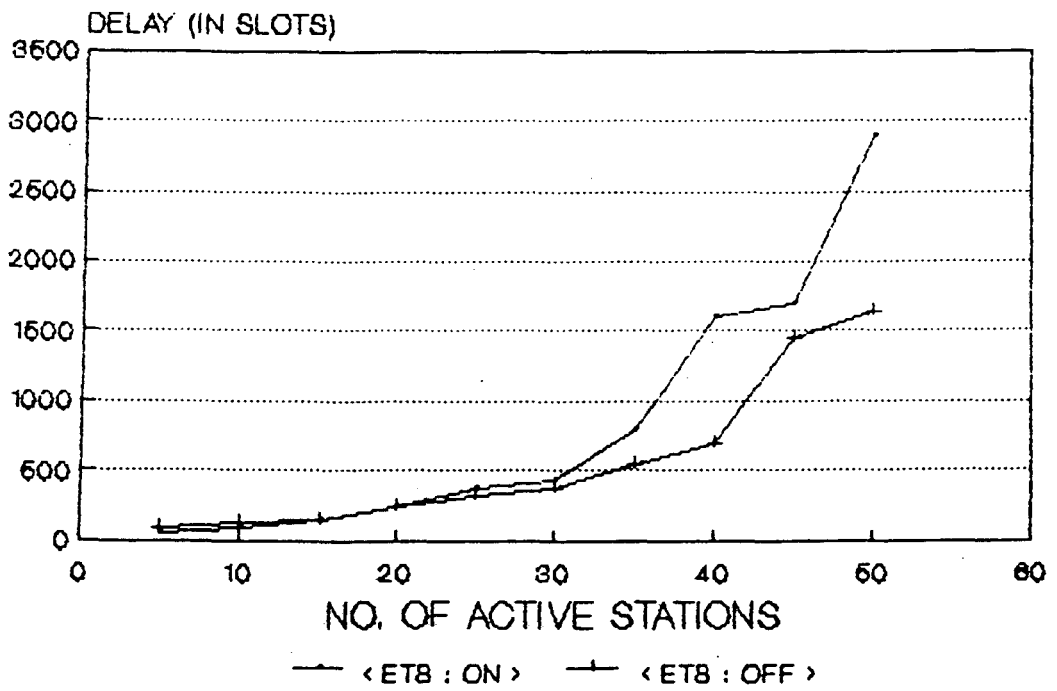
Graph #18: This graph shows the effect of ETS technique on the per packet transfer delay for ordinary data. The channel access probability assignment is graded. The transfer delay in case of <ETS: OFF> is higher than for the case of <ETS: OFF> upto 30 stations the delay for the two schemes is only slightly different. But on higher loads, the delay increases

TRANSFER DELAY ORDINARY DATA



Graph #17

TRANSFER DELAY ORDINARY DATA < CAP : ON >



Graph #18

steeply in <ETS: ON> In <ETS:OFF> stabilization occurs after 40 active stations whereas with <ETS: ON> stabilization occurs after 50 active stations.

In all cases, the absolute value for delay are much lower than the maximum delay limits imposed by various data traffics i.e.

<u>Data traffic</u>	<u>Max. delay limit</u>
Video	2500 slots
Voice	12750 slots
Ordinary data	--

5.2 THROUGHPUT Vs. LOAD

A multimedia system offers various kinds of data traffics. These data traffics use different bandwidths. Video uses a Bandwidth of 1.5 Mbps, voice data uses a bandwidth of 64Kbps and ordinary computer data uses a bandwidth of 64kbps of the channel. Therefore one multimedia system uses 1.085% of channel bandwidth. As the number of active stations increase, the load offered increases. The load can also be expressed in terms of fraction of channel bandwidth. When a source transmits data in a slot, the already existing signal on the bus is replaced by this locally generated signal. In the case of a collision, the data of the downstream source exists on the slot and the

slot is not wasted. Thus a slot is utilized if one or more sources put their data in it. Graphs #19, #20 and #21 show the channel utilization plotted for different amounts of load offered. These graphs show the effect of fairness schemes on the channel utilization.

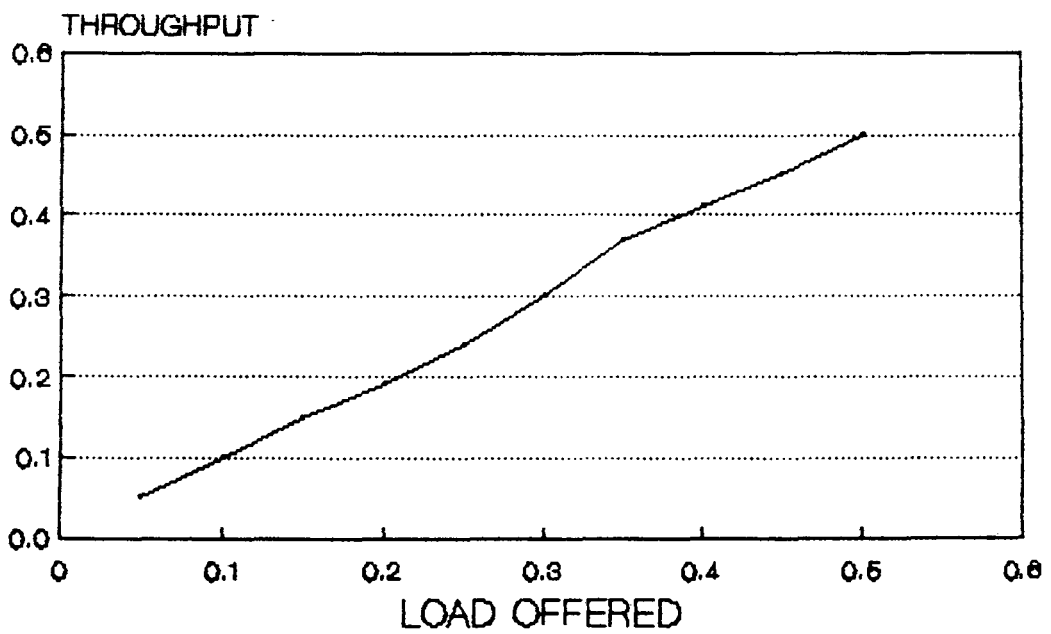
As the load increases, more slots are utilized for transmission of data and the throughput of the system increases. The graphs between throughput and the load offered are simple straight lines for these loads offered. These graphs are different in nature to the "Throughput vs load offered" graph for simple ALOHA contention system. In the case of ALOHA, a collision results in wastage of the slot since both the colliding packets are lost. Hence the throughput increases with increasing load offered only up to a certain load level (0.18 for unslotted ALOHA to 0.36 for slotted ALOHA). When the load is increased further the throughput starts decreasing. In contrast, as explained earlier, throughput in this system increases with increasing load. These graphs shall show saturation when the offered load becomes equal to the capacity of the network. That stage comes at nearly 90 active stations.

5.3 SUCCESSFUL TRAFFIC TRANSMISSION

A station is said to be successful if it is able to successfully transmit data traffic. Graphs #22, #23, #24

THROUGHPUT/LOAD OFFERED

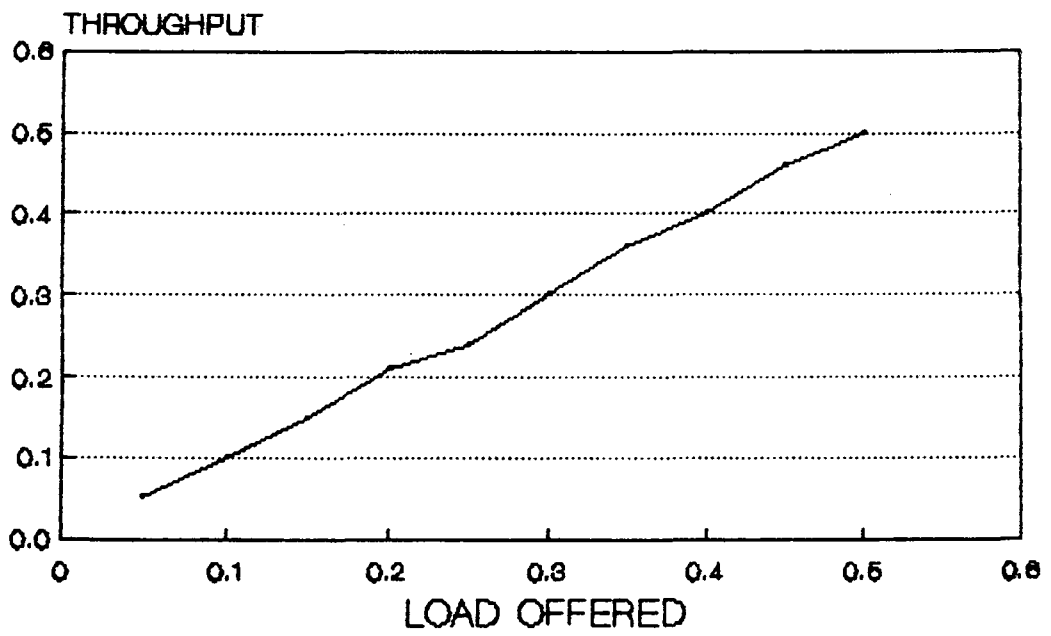
< CAP : OFF >



Graph #19

THROUGHPUT/LOAD OFFERED

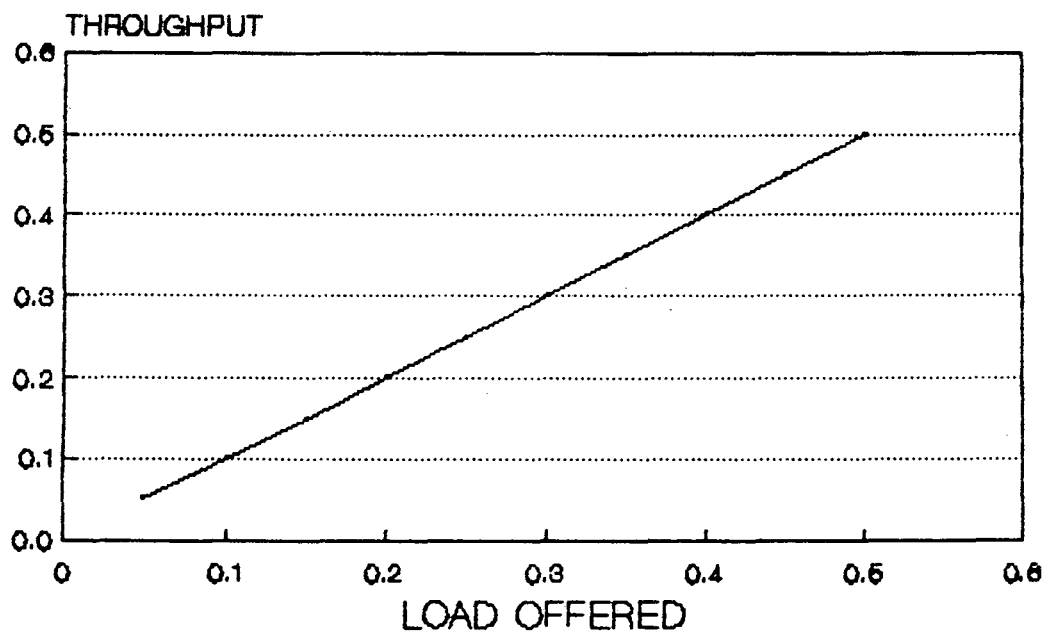
< CAP : ON >



Graph #20

THROUGHPUT/LOAD OFFERED

< CAP : ON > < ETS : ON >



Graph #21

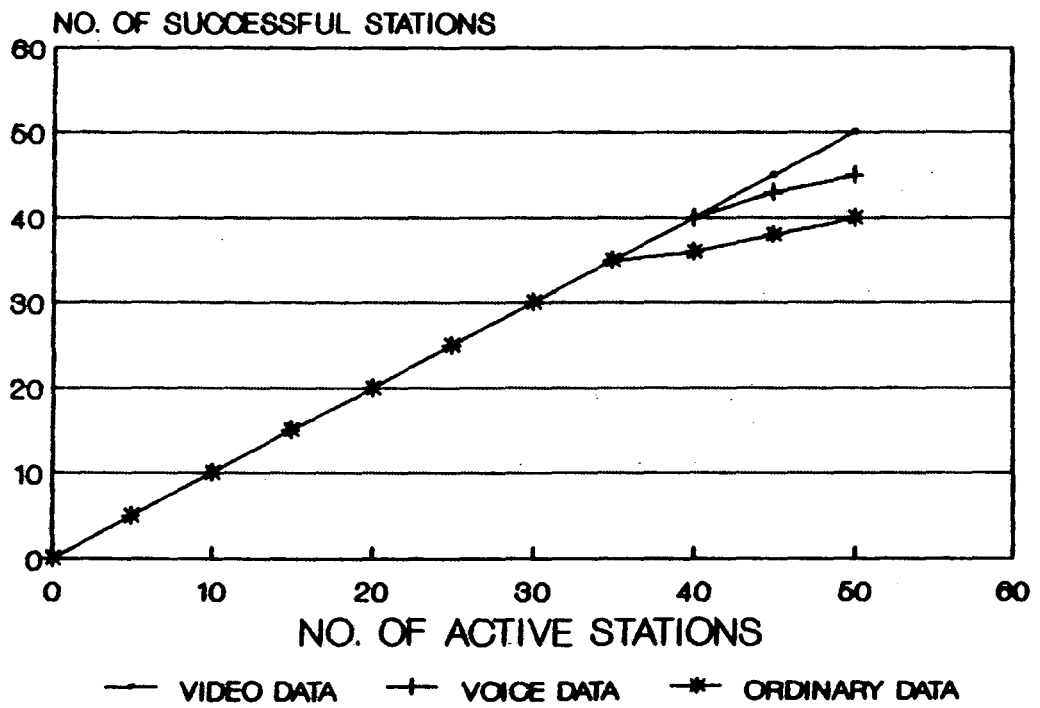
show the number of successful stations at different loads under different fair schemes.

Graph #22: This graph shows the no. of successful stations for different data types when Channel Access probability for all stations is same i.e. equal to one. When 40 stations are active , only 35 of them are able to transfer ordinary data. When 50 stations are active only 45 are able to transfer voice data and only 42 are able to transfer ordinary data.

Graph #23 : This graph shows the number of successful stations at different loads under <CAP:ON> assignment. Video data is transferred by all the stations upto 50 active stations. All stations are able to transfer all types of data traffics, till number of active stations is less than 50. On further increasing the number of active stations, some stations fail to transfer voice and ordinary data, though all of them are able to transfer video data.

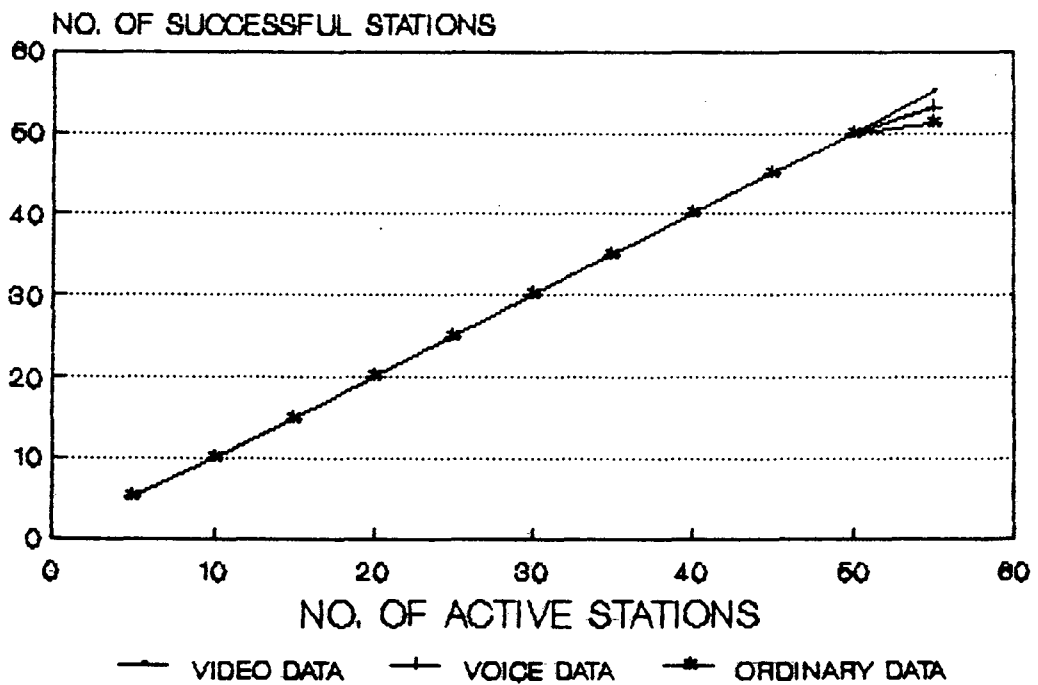
Graph #24: This graph shows the effect of <E.T.S.:ON> technique with <CAP:ON> on the number of successful stations at different loads. All stations are successful till the load is less than 35 active stations. On increasing the number of active stations further i.e. when 40 stations are active, one fails to transfer voice data and two fail to transfer ordinary data. This situation deteriorates as number of

SUCCESSFUL TRAFFIC TRANSMISSION < CAP : OFF >



Graph #22.

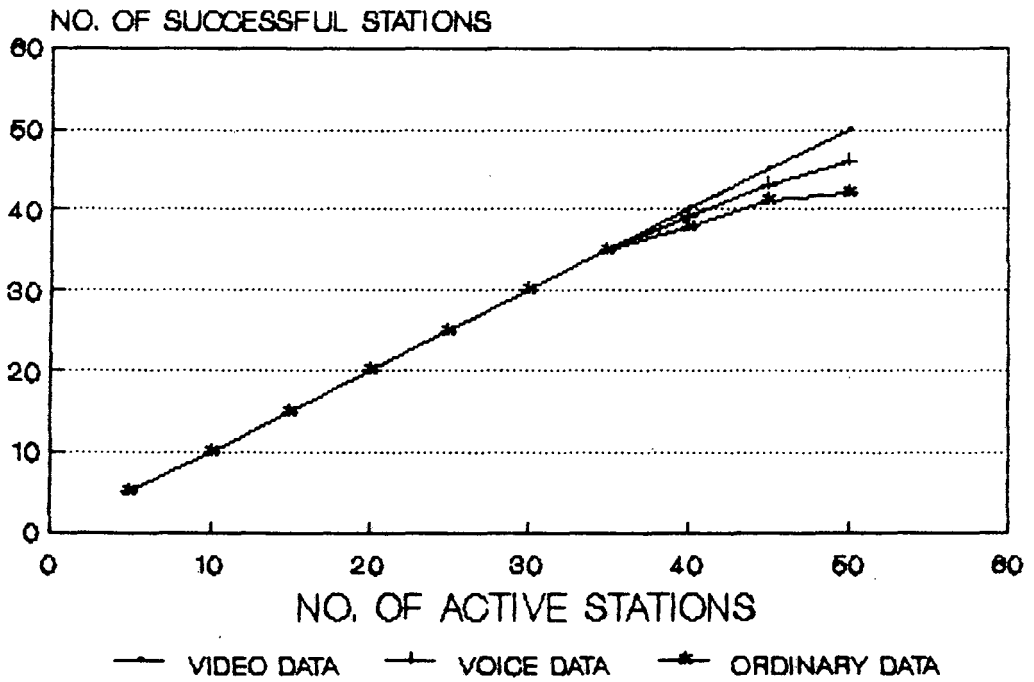
SUCCESSFUL TRAFFIC TRANSMISSION < CAP : ON >



Graph #23

SUCCESSFUL TRAFFIC TRANSMISSION

< CAP : ON > < ETS : ON >



Graph #24

stations are further increased. At 50 active station, 4 stations fail to transfer voice data and 8 stations fail to transfer ordinary data. Video data is successfully transferred by all stations for the load of 50 active stations.

On analyzing the above three graphs it was found that when channel Access Prob. of all stations is kept equal, the network is able to cater for only 35 active stations. Whereas when Channel Access Probability is graded for giving fair access to all sources, it was found that the system was able to cater for 50 active stations. On applying equal Timer setting technique and graded channel access probability, it was found that only 35 stations were able to transfer data successfully therefore I conclude that Channel Access probability assignment without equal timer setting technique is best suited for this application.

5.4 SIMULATION RUN LENGTH

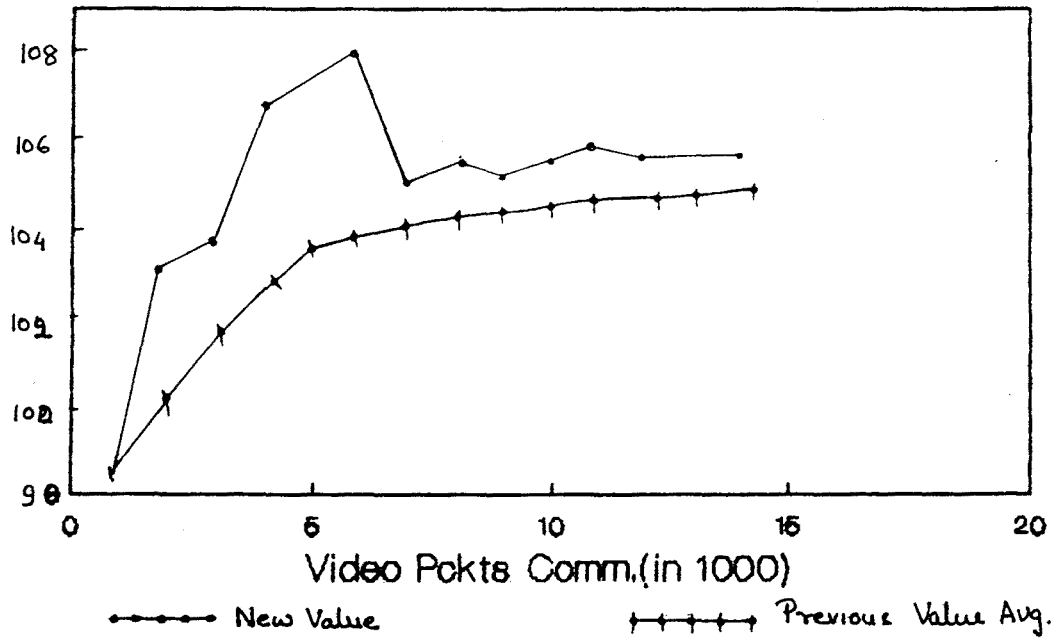
A simulation run is stopped when the control variable chosen is found to be stabilized. The control variable chosen for this simulation is the average transfer delay for the video packets. Since video traffic is the dominant traffic on this network, this choice for control variable is justified.

Graph #25: This graph shows the variation of average transfer delay for video data with the length of the simulation run taken. The length of the simulation run is expressed in terms of the number of video packets transferred across the network. The new value curve shows fluctuations initially and after 10,000 packets, it starts stabilizing. The average of previous values curve shows the average of all previous values obtained at steps of 1000 pckts. At 13,000 packets transferred, the new value graph stabilize close to the average value graph and at this point, the simulation is stopped.

SIMULATION RUN LENGTH

< CAP : ON >

Video Pokt Transfer Delay



Graph 25

CHAPTER SIX

CONCLUSION AND FUTURE DIRECTIONS

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

In this project I have designed a multimedia protocol for a high speed fiber optic bus network. The design was simulated using Borland C⁺⁺. Performance Analysis of the designed protocol was carried out and the software developed successfully predicted the network behavior under varying traffic conditions. The results of the simulation run give the different delays in the transfer of packets across the network, the throughput of the system, the packet loss, the network service to individual stations.

The network has a maximum load limit, to which it can provide satisfactory service. This limit depends on fairness strategy used. On analyzing the results it was found that when <CAP:OFF> is used, the network could support only 35 active stations whereas when <CAP:ON> was used the number of successful stations increased to 50. When <ETS:ON> with <CAP:ON> was used, the network could support only 32 stations. So we find that <ETS:ON> is not a useful strategy.

When the load is moderate (i.e. around 30 stations), video packets face the least delay under <CAP:OFF>. The delay faced in <CAP:ON> is higher than this and <ETS:ON> results in highest delays. Voice and ordinary data packets

are not much affected by the fairness strategies. The degree of fairness to upstream sources is highest when <CAP:ON> and <ETS:OFF> scheme is used. The degree of fairness to upstream sources is least when <CAP:OFF> is used. ETS scheme does not have much effect.

Upto 15 active stations, the upstream sources get a good service in <CAP:OFF> scheme along with faster overall traffic transfer. For loads ranging from 15 to 35 active stations, the degree of fairness to upstream sources decreases. The <CAP:OFF> provides fast service only to downstream sources, whereas <CAP:ON> provides fairness to upstream sources although the overall delay is slightly higher. For loads higher than 35 active stations <CAP:ON> with <ETS:OFF> is the best technique. Basically it is a trade off between delay and fairness. For loads less than 15 active stations <CAP:OFF> should be used for faster packet transfer and for higher loads <CAP:ON> with <ETS:OFF> should be used.

The prioritized queuing system used for output queues of multimedia sources has proved to be effective solution for meeting these constraints. The video data and voice data face delays much less than their respective upper limits. Moreover, the ordinary data does not suffer much because of the least priority given to it. It also gets satisfactory network service.

The Channel utilization of the network is high under all the three fairness schemes. the Channel utilization increases with increasing load. This is possible due to use of active optical taps. Under a collision situation, at least one packet is allowed to transmit, hence no slot is wasted. Thus the multimedia computer network can support applications like video conferencing, voice transmission and ordinary data transfer. It can support a maximum of 50 stations, all of which can offer all kinds of data.

Suggestions For Future Work

(1) LCSMA AND LCSMA-CD can result in better channel access strategies. Although their implementation is complex and expensive, they can improve the service provided by the network. Future work can be based on developing access strategies using these protocols.

(2) This network uses single wavelength for the signal transmission. An approach using multiple wavelengths can be implemented.

(3) A mechanism can be developed by which under low load conditions the network uses <CAP:OFF> technique and as the load increases the network adapts the <CAP:ON> technique.

APPENDIX A

MATHEMATICAL ANALYSIS OF CHANNEL UTILIZATION EFFICIENCY

A slot moving across the transmitting ends of the active sources goes unutilized only if none of the sources had packets to transmit in that slot. Therefore, under very heavy load conditions also, the channel utilization tends to one.

Suppose $N =$ number of active sources ≥ 1

$P_i =$ instantaneous probability of source "i" of transmitting a packet into slot being studied.

Probability of slot being unused is equal to the product of probabilities of not transmitting by individual sources

$$= \prod_{j=1}^N (1-P_j)$$

A slot is utilized if at least one source transmits a packet in that slot, so

$$\text{Slot Utilization Probability (S.U.P.)} = 1 - \prod_{j=1}^N (1-P_j) \quad N \geq 1$$

So as the number of active sources (N) increases, slot utilization probability increases and tends to one.

APPENDIX B
NETWORK SPECIFICATIONS

CHANNEL

Nature of channel	Optical Fiber
Bit Rate	150 Mbps
Transmission Speed	$2 \cdot 10^8$ m/sec

TAPS

Optical Taps	Active, Unidirectional
--------------	------------------------

TIMING

System Timing	Slotted
Slot Time	13.33 micro sec.
packet length	2000 bits

PROTOCOL

ALOHA

TRAFFIC CHARACTERISTICS

	VIDEO	VOICE	ORD. DATA
BANDWIDTH	1.5Mbps	64Kbps	64Kbps
NATURE	S	B	B
BITSTREAM	C	U	U
AVG. TALKSPURT	-	1.67 ms	-
AVG. SILENCE INTERVAL	-	1.33 ms	-

S = Steady

B = Bursty

C = Compressed

U = Uncompressed

BIBLIOGRAPHY

- [1] FRANCIS NEELAMKAVIL, "Computer Simulation and Modeling", John Wiley and Sons, Chichester, Britain, 1987.
- [2] NARSINGH DEO, " System Simulation with Digital Computer", Prentice Hall of India, New Delhi, 1991.
- [3] DIEDIER LE GALE, " MPEG : A Video Compression Standard for Multimedia Applications", ACM Communications, Vol.34, No. 4, pp. 47-58, April 1991.
- [4] GREGORY K. WALLACE, " The JPEG Still Picture Compression Standard ", ACM Comm., Vol. 34, No.4, pp. 31-44, April 1991.
- [5] ANDREW S. TANENBAUM, " Computer Networks ", Prentice Hall of India, New Delhi, 1990.
- [6] CHONG-WEI TSENG, BOR-UEI CHEN, " D-Net, A New Scheme for High Data Rate Optical Local Area Networks", IEEE Jour. on Special Areas in Communications (SAC), Vol. SAC-1, No.3, pp. 493-499, April 1983.
- [7] NICHOLAS F. MAXEMCHUK, " Twelve Random Access Strategies for Fiber Optic Networks", IEEE transactions on Communications, Vol.36, No.8, pp. 481-489, August 1988.

[8] AUREL A. LAZAR, JOHN S. WHITE, " Packetized Video on MAGNET", Optical Engineering (Journal), Vol.26, No.7, pp. 596-601, July 1987.

[9] COSMOS NICOLAOU, " An Architecture for Real Time Multimedia Communication Systems", IEEE Journal on SAC, Vol.8, No.3, pp. 391-397, April 1990.

[10] TATSUYA SUDA, TRACY T. BRADLEY, " Packetized Voice/Data Integrated Transmission on a Token Passing Ring Local Area Network", IEEE transactions on communications, Vol.37, No.3, pp. 238-246, March 1989.

SOURCE LISTING

```

// multimed.c          FINAL VERSION
// SOURCE CODE OF M.TECH PROJECT
// make sure whether BGI INITIALIZATION is proper !
//***** standard header files inclusion *****
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <alloc.h>

// *****
// This block contains the header declarations for the program

#ifndef NULL
#define NULL 0
#endif
#ifndef ZERO
#define ZERO 0
#endif

#define TWO 2
#define FOUR 4

#define MAX_STNS 64
#define NO_SLOTS_IN_RUN 1000
#define MAX_SLOTS_TO_RUN 65000

#define YES 1
#define NO 0
#define TRUE 1
#define FALSE 0

#ifndef DECIMAL
#define DECIMAL 10
#endif

#define ONE_SECONDS 1000

#define INVALID -1 // used in move_slots()
// used in initialization

#define NO_OF_DATA_TYPES 3 // no. of data types

#define EMPTY_SLOT 0
#define ORDINARY_DATA 1
#define VOICE_DATA 2
#define VIDEO_DATA 3

#define MAX_VOICE_DELAY 12750

```

```

#define MAX_VIDEO_DELAY 2500

#define NORMAL_Q_LEN_LIMIT 6
#define UPPER_Q_LEN_LIMIT 8
#define SERIAL_LENGTH 20

#define BASE_FOR_RANDOM 0.984 //meant for producing avg. 1 pkts per message
// used for ordinary and video traffics here.

#define MEAN_ORD 2340 // avg. gap between successive ord. data
// spurts. This and BASE_.. leads to
// avg. 1 pkt per 2000 slot

#define MEAN_VOICE 2340 // avg. gap between talk spurts that is
// avg. 1 pkt is produced per 225 slots.
// approximating 125 + 100

#define MEAN_VIDEO 100 // avg. gap between successive video data pkts
// this leads to avg. 1 pkt per 125 slots

#define NO_PKTS_IN_UNIT 1000 // the unit of the packet store
// the unit length shall also be dependent
// on the no. of slots in the unit run
// *****
// this block contains headers for the simulation verification part

#define Y_VALUE 1.65 // value used in the formula for n(i+1) from n(i)
#define CONFIDENCE_LIMIT 0.05 // desired confidence limit (interval)
#define INITIAL_TRIES 500 // no. of calls initially to random no. gnrt
//*****
// this block contains the header declarations for the graphics part

#define UPPER 10
#define LOWER 250
#define RIGHT 630
#define LEFT 15 //the four limits of the bus

#define LEFT_LOCO 25 // denotes the locomotive position's X.

#define GAP 3 // gap between real and virtual bus

#define SLOT_LENGTH 20 // denotes the length of the slot in pixels
#define MAX_SLOTS 74 // denotes the max no. of slots simultaneous
// on screen.

#define COLUMN_IN_PKT_STORE 5 // no. of columns of the packet store

#define CHANNEL_LENGTH (2*(RIGHT - 55 + GAP ) + (LOWER - UPPER + 2 * GAP ))
// the total length of the channel over which the sources are arranged
#define ends_gap(x) (CHANNEL_LENGTH - (18 * x))/SLOT_LENGTH

```

```

// macro to return the gap between tthe ends
// of a slot in terms of no. of slots

#define abs_gap(x1,x2) (floor((9.0 * (float)abs(x1-x2))/(float)SLOT_LENGTH))

#define stn_gap(xd,xs) (xd > xs) ? abs_gap(xd,xs) : (-abs_gap(xd,xs))
// these two macros are used to find the gap
// between a source and its destination stn
// in terms of number of slots seperating
// them.

#define d_point_offset (LOWER - UPPER)/SLOT_LENGTH
// this value is to be subtracted from the
// communication delay calculation

// now the dimensions of the result window and related objects start.
#define NO_BOX_PER_ROW 10 // in the result window

#define MARGIN_UPPER 5
#define MARGIN_LEFT 5 // margins between the boxes and result( or q) window

#define BOX_LENGTH 39
#define BOX_HEIGHT 20 // the dimensions of the slot & queue box

#define LEFT_R_WIN LEFT_LOCD
#define UP_R_WIN LOWER + 20 // gap between real bus and window =20
#define RIGHT_R_WIN LEFT_R_WIN + NO_BOX_PER_ROW * (BOX_LENGTH) + (2 * MARGIN_LEFT)
#define DOWN_R_WIN UP_R_WIN+209 // height of window = 209
// these are the four limits of the
// result window

// now the specific dimensions of the queue window and its objects start.

#define LEFT_Q_WIN RIGHT_R_WIN + (3* MARGIN_LEFT) // leaving the margin
#define UP_Q_WIN UP_R_WIN // same as result window
#define RIGHT_Q_WIN RIGHT // same as real bus
#define DOWN_Q_WIN DOWN_R_WIN // same as result window

#define SOURCES_SHOWN 4 // no of sources shown in the q window
#define Q_PACKETS_SHOWN 7 // no of packets in the queue of a source

#define REFRESH_COLOR GREEN // this color meant for clearing of the
// result window

#define ORDINARY_COLOR LIGHTGRAY
#define VOICE_COLOR LIGHTGREEN
#define VIDEO_COLOR LIGHTMAGENTA
#define EMPTY_COLOR BLACK // the colors to be filled in the result
// boxes in the result window.

```

```

// temp check with black ** white ** has been chosen with a special reason

// *****
// this block contains the global variable declarations in original

FILE *fp; // temporary file structure for checking

unsigned int f_act_sources[MAX_STNS]; // array for storing which sources
// are going to be active for a particular run
// initialized to 'NO'

unsigned int f_shown_sources[MAX_STNS]; // array for storing which sources are going
// to be displayed in the Q-window for this
// particular run
// initialized to 'NO'

unsigned int f_special_status[MAX_STNS] ; //flags to indicate special status
unsigned int f_killer_status[MAX_STNS] ; // flats to indicate killer status

unsigned int serial_special_status[SERIAL_LENGTH] ; // array to record serial in
// which the source enter and leave special status
unsigned int serial_killer_status[SERIAL_LENGTH] ; // array to record serial in
// which the source enter and leave killer status

unsigned int empty_special_entry = ZERO ;// indicators for empty positions
unsigned int empty_killer_entry = ZERO ;// ,,

int column_shown_sources[MAX_STNS]; // array for storing what are the columns to which
// a source has been mapped for the Q_WINDOW
// initialized to 'INVALID'

int shown_sources[SOURCES_SHOWN]; // array for storing wich sources are being
// displayed in the queue window
// initialized to the actual values
// these values are the internal values

// *****
int x_y_q_boxes[SOURCES_SHOWN][Q_PACKETS_SHOWN][2];
// 3-D array of integers to store the coords
// of the boxes in the rows of the Q window
int x_y_sname_box##[SOURCES_SHOWN][2];
// 2-D array of integers to store the coords
// of the boxes for displaying the names of
// the sources shown in the queue window.

int destn_stn[MAX_STNS]; // array for storing the destn station for
// a station.
// initialized to 'INVALID'

unsigned int no_of_pkts_in_q[MAX_STNS] ;// contains the no. of pkts in the
// queue of a source at a time instant

struct packet

```

```

unsigned int packet_id;           // packet identifier no. init to RAND_MAX
unsigned int src_stn;             // source stn no. init to ZERO
unsigned int type_of_pkt;        // type of pkt. (data - 0, voice - 1, video -2
// init to ZERO

unsigned int slot_no_gen;        // slot's no. in which this pkt was generated
unsigned int slot_no_reach_top;  // slot's no. in which this pkt reached 0 top
unsigned int slot_no_reach_destn; // slot's no. in which this pkt reached destn
// these three initialized to ZERO;
struct packet *next;            // pointer to the next packet in the queue
};                               // structure for a packet's details

struct slot
{
int packet_id;                  // packet identifier no.
int src_stn;                    // these two initialized to INVALID
int type_of_pkt;                // type of pkt. (data - 0, voice - 1, video -2
// initialized to 'EMPTY_SLOT'
};                               // structure for a slot's details

struct slot * head_slot_list[MAX_SLOTS]; // array of pointers to
// slot content details structure
// initialized to pointers to
// allocated nodes

struct packet * head_packet_queue[MAX_STNS]; // array of pointers to
// packet queues of sources
// initialized to NULL

int size_of_packet = sizeof(struct packet);
// global variable to store size of packet
// structure; saves many calls to sizeof()

// *****
// simulation specific variables are declared here in this block
unsigned int f_bwb_switch;
// flag used to indicate the status of
// the bandwidth balancing switch
// 1-'on' 0-'off'

unsigned int f_prob_assign;
// flag used to indicate the type of
// probability assignment;
// '0' - all sources have equal C. A. probability
// '1' - probability assignment is graded one

```

```

unsigned int f_try_limit_reached = NO ;
// flag used to denote whether the current try limit for random
// number generation routine is reached .

unsigned int f_simul_over = NO ;
// flag used to denote whether the simulation verification
// shows this is the time for ending of the simulation or not .
unsigned int f_mood_over = NO ;
// flag used to end the simulation by the user
unsigned int tried = ZERO ;
/// it is an unsigned variable to store the no. of times
// the random no. generator has been called for generating traffic

unsigned int try_upto = INITIAL_TRIES ;
// no. of times, the random number generator is initially called
// without a check on the standard deviation of its output

unsigned long sum_numbers = ZERO ;
// sum of the number of packets generated by the traffic generator

float deviation_sum = 0.0 ;
// long variable to store the sum of deviations

unsigned int no_of_act_stns =ZERO;
// this is the global variable to denote the number of stations
// which are active in a particular run

unsigned int slots_passed = ZERO ;
// this is an unsigned long integer variable which is used to store
// the number of slots that have passed through the network

int x_white_upper =INVALID,y_white_right = INVALID,x_white_lower= INVALID;
// the three white pixel positions;
// mind that they have been initialized to INVALID;

float prob_base[MAX_STNS]; // array of floating pt numbers to store the
// probability base values by virtue of the
// position of the source in the network

unsigned int normal_min_wait[MAX_STNS] ;
// unsigned int array to store the normal
// value of the minimum waits between successive
// retransmission attempts by a source

unsigned int min_wait[MAX_STNS] ;
// unsigned int array to store the numbers.
// of slots upto which the sources have to wait
// between successive transmission attempts.
// initialized to run-specific actual value in init_all()

unsigned int wait_counter[MAX_STNS][NO_OF_DATA_TYPES];
// global 2-D array for storing the wait_next_spurt values for

```



```

// different data types.
// [][0] - ord, [][1] - voice, [][2] - video
// initialized to ZERO

unsigned int access_wait[MAX_STNS];
// this counters will store the number of slots
// for which the source has to wait before trying
// to transmit again.

// ***** varibales for result calculations
unsigned long total_q_delay[MAX_STNS][NO_OF_DATA_TYPES] ;
// array of unsigned longs to store the total queuing delays
// of the different types of packets
unsigned long total_comm_delay[MAX_STNS][NO_OF_DATA_TYPES] ;
// array of unsigned longs to store the total communication delays
// of the different types of packets
unsigned int total_pkts_trans[MAX_STNS][NO_OF_DATA_TYPES] ;
// array of unsigned ints to store the total communication delays
// of the different types of packets

unsigned int no_of_slots_empty = ZERO ; // this variable holds the no. of slots
// that have passed empty across the bus

unsigned int delay_allowed[NO_OF_DATA_TYPES] ;
// array to hold the max. no. of slots by
// which a packet of particular data type
// could be allowed to get delayed

unsigned int no_of_pkts_delayed[MAX_STNS][NO_OF_DATA_TYPES] ;
// array to store the no. of packets of diff. types delayed beyond
// their limits .

unsigned int packet_store[NO_PKTS_IN_UNIT][COLUMN_IN_PKT_STORE];
// 2-D array of unsigned int-s to store the info about pkts
// finally for each unit run;
// initialized to ZERO

unsigned int curr_empty_entry;
// unsigned int variable to store the no. of the current
// empty entry in the packet store
// global variable take care;
// initialized to ZERO

// *****graphics declarations *****

unsigned int x_y_r_boxes[MAX_SLOTS][2];
// stores the x and y coordinates of the boxes in the
// result_window
// initialized by their values

unsigned int x_y_slot_no_box[2] ;

```

```

    // array to store the x and y pts of the upper left
    // corner of the slot no box
int pkt_color_array[4];
    // stores the colors of the pkts in the result window
    // initialized by color values

char filename[15] ; // to store the name of the data file
// *****
//                               MAIN STARTS
void main(int argc, char * argv[])
{
    /* ----- */
    // function declarations
void init_all(void);
void rand_act_stns(void);
void remove_lock_stepsync(void) ;
void init_graph(void);
void draw_net(void);
void draw_result_win(void);
void draw_q_win(void);
void move_slots(void);
void simulation (void) ;
    /* ----- */

// char filename[15] ; // array to store the name of the file in which
// the data generated by the simulation run is to
clrscr() ; // clear the screen
if(argc > 1)
    strcpy(filename,argv[1]);
else
{
    printf("Please <ENTER> the name of output data file <*.dat> : " ) ;
    scanf("%s",filename) ;
}
clrscr() ; // clear the screen
// fp = fopen(filename,"w");

// if(fp == NULL)
// {
//     printf("Could not open file!");
//     exit(1);
// }

    init_all(); // initializes all the global variables
    rand_act_stns(); // randomizing the stations to be active
    remove_lock_stepsync() ; // to remove lock step synchronization
    init_graph(); // initializing the graphics
    draw_net(); // drawing the network structure
    draw_result_win(); // drawing the result window and its objects
    draw_q_win(); // drawing the q window and its objects
    simulation() ;
// fclose(fp) ;

```

```

}
//
//                               MAIN ENDS
// ***** GLOBAL INITIALIZATION BLOCK STARTS *****
// this block contains routines for the various initializations
// this includes routines for allocating memory for nodes

// *****
// this function allocates memory initializes the slot list & all headers
// called from
// 1. main()
// *****
void init_all()
{
/* -----*/
// function declarations
void refresh_packet_store();
/* -----*/

register int i,j;
register int size_of_slot; // just meant to make fast routine

size_of_slot = sizeof (struct slot) ; // making it faster

for(i=0;i<MAX_STNS;i++)
{
f_act_sources[i] = NO; // initializing all sources as INACTIVE.
f_shown_sources[i] = NO; // initializing all sources as INACTIVE.
column_shown_sources[i] = INVALID;
destn_stn[i] = INVALID ;
// initializing columns to INVALID
}

for(i=0;i<MAX_SLOTS;i++) // allocating and intializing slot list
{
head_slot_list[i] = (struct slot *)malloc(size_of_slot);
if(head_slot_list[i] == NULL)
{
clrscr();
puts("Memory not allocated for node in init_all()!\n check it!");
exit(1);
} // checking for the case when memory is not allocated !

head_slot_list[i] -> packet_id = INVALID;
head_slot_list[i] -> src_stn = INVALID;
head_slot_list[i] -> type_of_pkt = EMPTY_SLOT;
// filling type as empty
}

// initializing the array of pointers to the stations' packet queues
for(i=0;i<MAX_STNS;i++)
head_packet_queue[i] = NULL;

```

```

// initializing the colors of the pkts array
pkt_color_array[EMPTY_SLOT] = EMPTY_COLOR;
pkt_color_array[ORDINARY_DATA] = ORDINARY_COLOR;
pkt_color_array[VOICE_DATA] = VOICE_COLOR;
pkt_color_array[VIDEO_DATA] = VIDEO_COLOR;

// initializing the wait_counters of stations to zero
for(i=0;i<MAX_STNS;i++) // for all stations
  for(j=0;j<NO_OF_DATA_TYPES;j++) // for all data types
    wait_counter[i][j] = ZERO;

// initializing the min_wait array and C. A. probability array
for(i=0; i<MAX_STNS ;i++)
{
  min_wait[i] = ZERO ;
  ndrmal_min_wait[i] = ZERO ;
  prob_base[i] = 0.0 ;
  access_wait[i] = ZERO ;
}
//initializing the delay variable to hold the result of a run
for(i=0;i<MAX_STNS;i++)
  for(j=0;j<NO_OF_DATA_TYPES ;j++)
  {
    total_q_delay[i][j] = ZERO ;
    total_comm_delay[i][j] = ZERO ;
    total_pkts_transf[i][j] = ZERO ;
    no_of_pkts_delayed[i][j] = ZERO ;
  }
// initializing the delay allowed values for the three data types
delay_allowed[ORDINARY_DATA -1] = RAND_MAX ; // just for formality
delay_allowed[VOICE_DATA -1] = MAX_VOICE_DELAY ; // max delay 170 ms
delay_allowed[VIDEO_DATA -1] = MAX_VIDEO_DELAY ; // refresh_rate = 50 per sec.

// source shown array initialized to ZERO
for(i=0;i<SOURCES_SHOWN ; i++)
  shown_sources[i] = ZERO ;

// initializing the packet store
refresh_packet_store(); // called to initialize the packet store
// initially as ZERO

for(i=0;i<MAX_STNS;i++)
{
  f_special_status[i] = NO ;
  f_killer_status[i] = NO ;
  no_of_pkts_in_q[i] = ZERO ;
} //initializations

for(i=0;i< SERIAL_LENGTH ;i++)
{
  serial_special_status[i] = ZERO ;

```

```

serial_killer_status[i] = ZERO ;
}

)
// ***** GLOBAL INITIALIZATION BLOCK ENDS *****

// ***** ACTIVE STATION RANDOMIZATION BLOCK *****

// *****
//this routine asks the no. of stations to be active for a particular run
// and randomizes to find the suitable configuration until the user is
// satisfied with the configuration.
// called from
// 1; main()
// *****
void rand_act_stns()
{
char resp;
int stations ;
int j ;
register int rand_no, i ;

printf("\n\n <ENTER> the no. of active stations [ 1 to %d] :",MAX_STNS) ;
scanf("%d",&no_of_act_stns) ;

printf("\n\n IS BAND-WIDTH BALANCING to be used ( 1=yes/0=no) ? : ");
scanf("%d",&f_bwb_switch) ;

printf("\n\n IS PROBABILITY ASSIGNMENT 'FLAT EQUAL' or 'GRADED' (0=flat/1=graded)? : ");
scanf("%d",&f_prob_assign) ;

clrscr() ;

randomize(); // called here for whole of the program

printf(" Warming up the Random Number Generator ....") ;
delay(ONE_SECOND) ;
for(i=0 ;i<1000;i++)
j = rand() ;

if(no_of_act_stns == ZERO || no_of_act_stns > MAX_STNS)
{
printf("ok !\n Goodbye!");
exit(1);
} // if the no. specified is invalid, exit

else if(no_of_act_stns == MAX_STNS)
{
for(i=0;i<MAX_STNS;i++)
f_act_sources[i] = YES;
}
}

```

```

// if all stations are active, need not randomize
else
{
    clrscr();
    stations = ZERO ;
    printf("Would you like to specify some stations (Y/N) ? ");
    fflush();
    scanf("%c",&resp) ;
    if(resp == 'y' || resp == 'Y')
    {
        printf("Welcome ! How many ? ");
        scanf("%d", &stations) ;
        if(stations > no_of_act_stns)
            exit(1) ; // if this no. specified is larger then exit
        for(i=0;i<stations ;i++)
        {
            printf("\n Enter the active station no. [%d] : ",i+1) ;
            scanf("%d",&j) ;
            if((j>0) && (j<= MAX_STNS))
                f_act_sources[j-1] = YES ;
        }
        stations = ZERO ;
        // now checking how many different stations were declared active !
        for(i=0;i<MAX_STNS ;i++)
            if(f_act_sources[i] == YES )
                stations ++ ;
        // printing this information to the user
        printf("\n\n No. of different active stations specified = %d !\n ",stations ) ;
    }
    puts("\nRandomizing . . .");
    delay(ONE_SECOND);

    i=0;
    while(i< ( no_of_act_stns - stations ))
    {
        rand_no = random(MAX_STNS);
        if(f_act_sources[rand_no] == NO)
        {
            f_act_sources[rand_no] = YES;
            i++;
        }
    } // Generate reqd. no. of diff.random no.s in range 0 - MAX_STNS
}

for(i=0;i<MAX_STNS;i++)
{
    if(f_act_sources[i]==YES)
    {
        do
        {
            rand_no = random(MAX_STNS);
        }while(f_act_sources[rand_no] == NO || rand_no == i);
    }
}

```

```

        destn_stn[i]= rand_no;
    }
    // generating the random destination station for all
    // active stations and storing the destns and their
    // gap from source in no. of slots

// filling the queue window sources arrays
i=0;
while(f_act_sources[i]!= YES)
    i++; // reaching the index of the first active source
f_shown_sources[1] = YES;
shown_sources[0] = i;
column_shown_sources[i++] = 0 ;
//making this source active for show window
// at the first column
while(f_act_sources[i] != YES)
    i++; // reaching the index of the second active source
// from upstream side
f_shown_sources[i] = YES;
shown_sources[1] = i;
column_shown_sources[i] = 1;
// making this source active for show window
// at the second column

i = MAX_STNS - 1 ; // now moving to the downstream side for other
// two sources to be displayed

while(f_act_sources[i]!= YES)
    i--; // reaching the index of the fourth active source
    f_shown_sources[i] = YES;
shown_sources[3] = i;
column_shown_sources[i--] = 3 ;
//making this source active for show window
// at the fourth column
while(f_act_sources[i] != YES)
    i--; // reaching the index of the second active source
// from downstream side
f_shown_sources[i] = YES;
shown_sources[2] = i;
column_shown_sources[i] = 2;
// making this source active for show window
// at the third column

// printing all the calculation results to the screen
printf("\n The stations chosen to be active are : \n");
for(i=0;i<MAX_STNS;i++)
    if(f_act_sources[i] == YES)
        if(f_shown_sources[i] == YES)
            printf("%d=stn\t%d=destn\t%d=column\n",i+1,destn_stn[i] +1,column_shown_sources[i]);
        else

```

```

printf("%d=stn\t%d=destn\n",i+1,destn_stn[i] +1);

printf("\n\n Press any key to continue ...");
getch();

// now assigning the probability base values for active sources
j = no_of_act_stns; // this is the denom. for probabilities
i = MAX_STNS - i; // starting from the most downstream source

while(i >= ZERO)
{
  if(f_act_sources[i] == YES)
  {
    if(f_prob_assign == YES )
      prob_base[i] = 1.0/(float)j ;
    else
      prob_base[i] = 1.0 ;

    j --;
  }
  i --;
}

if(f_bwb_switch == YES)
for(i=0;i<MAX_STNS; i++)
{
  if(f_act_sources[i] == YES)
  {
    min_wait[i] = ends_gap(shown_sources[0]) ;
    normal_min_wait[i] = min_wait[i] ;
  }
}
else
for(i=0;i<MAX_STNS; i++)
{
  if(f_act_sources[i] == YES)
  {
    min_wait[i] = ends_gap(i) ;
    normal_min_wait[i] = min_wait[i] ;
  }
}
} // assignment of the min_wait values based on the status of flag

printf("\n No. of active stations are : %d",no_of_act_stns);
puts(" The assigned probabilities are :");
for(i=0;i < MAX_STNS ; i++)
  if(f_act_sources[i] == YES)
    printf("prob[%d] = %f, main_wait[%d] = %d\n",i+1,prob_base[i],i+1, min_wait[i]) ;

printf("\n\n Press any key to continue ...");
getch();

```



```

}
// *****
// this function sets the wait counters
// for traffic generations in such a way
// that the observed lock step synchro-
// nization is removed
// *****
void remove_lock_stepsync()
{
    register int i,j=0 ;

    for(i=0;i<MAX_STNS;i++)
    {
        if(f_act_sources[i]!=YES)
        {
            wait_counter[i][0] = (no_of_act_stns-j) *(MEAN_ORD/no_of_act_stns) ;
            wait_counter[i][1] = j*(MEAN_VOICE/no_of_act_stns) ;
            wait_counter[i][2] = j*(MEAN_VIDEO/no_of_act_stns) ;
            j++ ;
        }
    }
    /* clrscr() ;
    printf("The effect of removing lock step sync : \n");
    for(i=0;i<MAX_STNS;i++)
    {
        if(f_act_sources[i] == YES)
        {
            printf("\n Source[%d]",i+1) ;
            for(j=0;j<NO_OF_DATA_TYPES;j++)
                printf(" %d",wait_counter[j]);
        }
    }*/
}
// ***** RANDOMIZATION BLOCK ENDS *****

// ***** GRAPHICS INITIALIZATION BLOCK *****
// *****
// routine for graphics initialization
// called from
// main()
// *****
void init_graph()
{
    int maxx,maxy;
    char resp;
    int graphdriver,graphmode;

    detectgraph(&graphdriver, &graphmode); // initially detecting
    if(graphdriver < 0)
    {
        puts("can't detect a graphics card!");
    }
}

```

```

    exit(1);
  )// exit, if there is no graphics card.

    // graphdriver and graphmode now set with highest
    // resolution mode on adaptor card

    printf("\n card detected is");
    printf("  %d, hi_res mode is %d",graphdriver,graphmode);
    printf("\nProceed to initialization");
    printf(" with these parameters?(Enter y/n)");
    fflush(); // flush the input stream
    scanf("%c",&resp);
    if(resp == 'n' || resp == 'N')  exit(1);

    initgraph(&graphdriver, &graphmode,"c:\\bcpp");// TAKE CARE
    setbkcolor(BLACK);
    setcolor(WHITE);
  }
  // *****
  // ***** END OF GRAPHICS INITIALIZATION BLOCK ROUTINES *****

  // ***** NETWORK DRAWING ROUTINES BLOCK*****
  // Routines for drawing the network on the screen
  // *****
  // this routine calls the routines for drawing components of the net
  // this has been called from
  // 1. main()
  // *****
  void draw_net()
  {
    /* -----*/
    // function declarations
    void draw_bus(void);
    void draw_loco(void);
    void draw_stns(void);
    /* -----*/

    draw_bus(); // draws the bus of the network
    draw_loco(); // draws the locomotive generator
    draw_stns(); // draws the stations in the net
  }

  // *****
  // this routine draws the real bus in the network
  // called from
  // 1. draw_net() in this block
  // *****
  void draw_bus()
  // this routine draws the network bus
  {
    rectangle(5,UPPER-10,15,UPPER+10);
  }

```

```

rectangle(5,LOWER-10,15,LOWER+10); // for the two ends of the bus

        setfillstyle(WIDE_DOT_FILL,WHITE);
floodfill(10,UPPER,WHITE);
floodfill(10,LOWER,WHITE);          // filling the two end boxes of the bus

        line(LEFT,UPPER,RIGHT,UPPER);
line(LEFT,LOWER,RIGHT,LOWER);
line(RIGHT,UPPER,RIGHT,LOWER);    // the three arms of the bus
}

// *****
// this routine draws the network locomotive generator
// called from
// 1. draw_net() in this block
// *****
void draw_loco()
{
        int y_centre;
        int radius;
        radius =15;
        y_centre = (UPPER+LOWER)/2;
        circle(LEFT_LOCO,y_centre,radius);
line(LEFT_LOCO,UPPER,LEFT_LOCO,y_centre - radius);
        line(LEFT_LOCO,y_centre + radius,LEFT_LOCO,LOWER);
        outtextxy(LEFT_LOCO -3,y_centre - 3,"G");
}
// *****
// this routine draws the station boxes in the net
// and fills the entries in the array
// meant for the purpose of displaying
// the access success star signs on
// the sources
// called from
// 1. draw_net() in this block
// *****
void draw_stns()
{
// this routine draws the network stations.
// this uses the global flag of active sources to draw their colors in them

register int i,j;

j=0;
// setcolor(DARKGRAY);
setcolor(LIGHTGRAY);
setfillstyle(SOLID_FILL,RED/*YELLOWGREENLIGHTRED*/); // color to fill with

for(i=1;i<=MAX_STNS;i=i+2)
{
        if(f_act_sources[i-1]==YES)
                // setcolor(YELLOW);

```

```

    setcolor(LIGHTRED);
    rectangle(50+(i-1)*5+j,UPPER + 10,50+(i-1)*5+10+j,UPPER +110);
    if(f_act_sources[i-1]==YES)
        floodfill(50+(i-1)*5 +j+2,UPPER + 12,LIGHTRED) ; // color of border

    line(50+(i*5)+j,UPPER,50+(i*5)+j,UPPER + 10);
// now filling the array entries

    if(f_act_sources[i-1]==YES)
        // setcolor(LIGHTCYAN);
        // setcolor(LIGHTGREEN);
        setcolor(LIGHTMAGENTA);
        line(50+(i*5)+j,UPPER+110 ,50+(i*5)+j,LOWER);

    if(f_act_sources[i]==YES)
        setcolor(LIGHTRED);
        // setcolor(YELLOW);
    else
        setcolor(LIGHTGRAY);
        // setcolor(DARKGRAY);

    rectangle(59+(i-1)*5+j,UPPER +130,59+(i-1)*5+10+j,LOWER-10);
    if(f_act_sources[i] == YES)
        floodfill(59+(i-1)*5 +j+2,UPPER + 132,LIGHTRED) ; // color of border
    line(59+(i*5)+j,UPPER,59+(i*5)+j,UPPER + 130);
// now filling the array entries

    if(f_act_sources[i]==YES)
        // setcolor(LIGHTCYAN);
        setcolor(LIGHTMAGENTA);
        // setcolor(LIGHTGREEN);

    line(59+(i*5)+j,LOWER - 10,59+(i*5)+j,LOWER);

    setcolor(LIGHTGRAY);
    // setcolor(DARKGRAY);
    j+=8;// j is being used to provide gaps between stations
}
setcolor(WHITE);
}
// ***** NETWORK DRAWING BLOCK ENDS *****

// ***** RESULT DISPLAY WINDOW ROUTINES *****
// BLOCK contains routines for preparing displays for showing results

// *****
// this routine draws the result window
// called from
// 1. main()
// 2. simulation control routine
// *****

```

```

void draw_result_win()
{
    /* -----*/
    // function declarations
    void set_x_y_r_boxes(void) ;
    void draw_all_r_boxes(void) ;
    void draw_data_type_boxes(void) ;
    void draw_slot_no_box(void) ;
    void draw_status_boxes(void) ;
    /* -----*/

    setcolor(WHITE);
    rectangle(LEFT_R_WIN,UP_R_WIN,RIGHT_R_WIN,DOWN_R_WIN); //result window
    setcolor(BLACK);
    rectangle(LEFT_R_WIN+1,UP_R_WIN+1,RIGHT_R_WIN-1,DOWN_R_WIN-1); //inner
        // only meant for refreshing of the window
    setcolor(WHITE);

    set_x_y_r_boxes(); // initialize the x,y array for box coordinates
    draw_all_r_boxes(); // draws all the slot boxes in the result window
    draw_data_type_boxes(); // draws the data type boxes at the bottom
        // of the result window
    draw_slot_no_box(); // draws a slot no box near the bottom of the
    // result window which shall continuously
    // display the no. of slots that have crossed
    // the network
    draw_status_boxes(); // draws boxes for displaying bwb status, prob. assignment
    // and the no. of active stations for a particular run

}
// *****
// this function initializes the array containing the x,y values for boxes
// accesses the global array x_y_r_boxes[][]
// called from
// draw_result_win() in this block
// *****
void set_x_y_r_boxes()
{
    /* -----*/
    // function declaration
    void rotate_x_y_r_boxes(void) ;
    /* -----*/
    register int i,
        row_box, // to store the row in the result win
        column_box, // to store the column in the result win
        x_coor, // to store the x value calculated
        y_coor; // to store the y value calculated

    for(i=0;i<MAX_SLOTS;i++)
    {
        row_box = i / NO_BOX_PER_ROW; // finding the row and column
        column_box = i % NO_BOX_PER_ROW; // of the box for slot 'i'
    }
}

```

```

x_coor = LEFT_R_WIN + MARGIN_LEFT + (column_box - 1) * (BOX_LENGTH);
y_coor = UP_R_WIN + MARGIN_UPPER + (row_box - 1) * (BOX_HEIGHT);

x_y_r_boxes[i][0] = x_coor; // storing the values into the
x_y_r_boxes[i][1] = y_coor; // global array x_y_r_boxes[][];
}
rotate_x_y_r_boxes();
}
// *****
// this function rotates the
// values in array x_y_r_boxes[]
// to the right by one place
// called from
// set_x_y_r_boxes() in this block
// *****
void rotate_x_y_r_boxes()
{
    register int i;
    register unsigned int temp_x, temp_y;

    temp_x = x_y_r_boxes[MAX_SLOTS - 1][0];
    temp_y = x_y_r_boxes[MAX_SLOTS - 1][1];
    // storing the leftmost value temp.y
    for(i=MAX_SLOTS - 1; i>0; i--)
    {
        x_y_r_boxes[i][0] = x_y_r_boxes[i-1][0];
        x_y_r_boxes[i][1] = x_y_r_boxes[i-1][1];
    }
    x_y_r_boxes[0][0] = temp_x;
    x_y_r_boxes[0][1] = temp_y;
}
// *****
// this function draws all the boxes using draw_a_box()
// called from
// 1. draw_result_win() in this block
// *****
void draw_all_r_boxes()
{
    /* -----*/
    // function declaration
    void draw_a_box(unsigned int, unsigned int);
    /* -----*/

    register unsigned int i;
    for(i=1; i< MAX_SLOTS; i++)
        draw_a_box(i, i); // draw the boxes in the result win
    draw_a_box(0, 0); // draw the last box
}

// *****
// this function draws a box at given row and column in result window.
// called from

```

```

// 1. move_slots() in slot movement block
// *****
void draw_a_box(unsigned int curr_slot_no , unsigned int actual_slot_no)
{
    char slot_detail[10],
    temp_string[10]; // to print the details of pkts in the slot box.

    register int content_type, x_coor,y_coor,color;
    // content_type stores the type of packet in the slot
    // color stores the corresponding filling color
    // other two store the coordinates of the box

    x_coor = x_y_r_boxes[actual_slot_no][0];
    y_coor = x_y_r_boxes[actual_slot_no][1]; // recovering the coords

    setcolor(WHITE) ;
    rectangle(x_coor,y_coor,x_coor + BOX_LENGTH -2,y_coor + BOX_HEIGHT - 2);
    // outer rectangle

    // now filling the space with yellow color
    setfillstyle(SOLID_FILL,YELLOW) ;
    floodfill(x_coor+2,y_coor+2,WHITE) ;

    // now filling the space with the packet color
    color = pkt_color_array[(head_slot_list[actual_slot_no] -> type_of_pkt)];
    setfillstyle(SOLID_FILL,color) ;
    floodfill(x_coor+2,y_coor+2,WHITE) ;

    // now putting th slot description into the result box
    if(color != EMPTY_COLOR)
    {
        setcolor(BLACK) ;
        ultoa((long)(head_slot_list[actual_slot_no]->src_stn + 1),slot_detail,DECIMAL);
        outtextxy(x_coor+2,y_coor+2,slot_detail);

        ultoa((long)(destn_stn[(head_slot_list[actual_slot_no]->src_stn)] + 1),slot_detail,DECIMAL);
        outtextxy(x_coor+20,y_coor+2,slot_detail);

        ultoa((long)curr_slot_no,slot_detail,DECIMAL);
        outtextxy(x_coor+2,y_coor+11,slot_detail);
    }
    setcolor(WHITE);
}
// *****
// this function draws the type color
// indicator boxes at the bottom of the
// result window
// called from
// 1. draw_result_win() in this block
// *****
void draw_data_type_boxes()

```

```

{
    register int i ;
    register int x_coor, y_coor ;
    register int color ;

    for(i=0; i< NO_OF_DATA_TYPES + 1;i++)
    {
        x_coor = LEFT_R_WIN + MARGIN_LEFT - 1 + (i * 2 * (BOX_LENGTH + 12)) ;
        y_coor = DOWN_R_WIN - BOX_HEIGHT + 4 ; // calc the upper left corner coords

        color = pkt_color_array[i] ;
        setcolor(color) ;
        setfillstyle(SOLID_FILL,color) ; // setting the color and fill style

        rectangle(x_coor, y_coor, x_coor + (2 * BOX_LENGTH )+ 5, y_coor + BOX_HEIGHT - MARGIN_UPPER - 4 ) ;
        floodfill(x_coor+2, y_coor + 2, color);
        // boxes drawn and filled with appropriate type color

        // now filling the name of the data type with black color
        setcolor(BLACK) ;

        switch(i)
        {
            case EMPTY_SLOT      : setcolor(WHITE) ;
                outtextxy(x_coor+2,y_coor+2,"EMPTY SLOT");
                setcolor(BLACK) ;
                break ;
            case ORDINARY_DATA   : outtextxy(x_coor+2,y_coor+2,"ORD. DATA");
                break ;
            case VOICE_DATA      : outtextxy(x_coor+2,y_coor+2,"VOICE DATA");
                break ;
            case VIDEO_DATA      : outtextxy(x_coor+2,y_coor+2,"VIDEO DATA");
                break ;
            default              : break ;
        }
    }
    setcolor(WHITE) ;
}
// *****
// this function draws a slot number
// box, near the bottom of the result
// window, which shall display the no.
// of the slots that have crossed the
// d-network
// called from
// 1. draw_result_win() in this block
// *****
void draw_slot_no_box()
{
    int x_coor, y_coor ;

    x_coor = LEFT_R_WIN + 4 * BOX_LENGTH + (2 * MARGIN_LEFT) ;

```



```

v_coor = UP_R_WIN + 7 * BOX_HEIGHT + (3*MARGIN_UPPER) ;

setcolor(WHITE) ; // to write text
outtextxy(x_coor,y_coor,"SLOTS PASSED :");

x_coor += (3 * BOX_LENGTH) - MARGIN_LEFT -1; // x value for the slot no. box
y_coor -= (2 * MARGIN_UPPER) - 2; // y value for the slot no. box
setcolor(BLUE); // background of the slot no box

x_y_slot_no_box[0] = x_coor ;
x_y_slot_no_box[1] = y_coor ; // x and y of the upper left corner of slot no box

rectangle(x_coor,y_coor,x_coor + 116 , y_coor + BOX_HEIGHT - 3 );
// 116 indicates the length of the slot no box
setfillstyle(SOLID_FILL,BLUE) ;
floodfill(x_coor +2,y_coor+2,BLUE);
setcolor(WHITE) ;
}
// *****
// this function updates the slot no
// box with the passed value of slots_passed
// called from
// move_slots() in slot movement block
// *****
void update_slot_no_box(unsigned int slots_passed_new)
{
char temp_buff[15] ; // temporary buffer to hold alphanumeric value

setcolor(BLUE) ; // setting the back ground color to the foreground

ultoa((long)(slots_passed_new - 1), temp_buff,DECIMAL) ;
outtextxy(x_y_slot_no_box[0]+ 20, x_y_slot_no_box[1]+7, temp_buff) ;
// since the array value are the pts of corner of the box
// so text is to be started at some space from the corner .

setcolor(YELLOW) ; // setting the foreground color again
ultoa((long)(slots_passed_new), temp_buff,DECIMAL) ;
outtextxy(x_y_slot_no_box[0]+20, x_y_slot_no_box[1]+7, temp_buff) ;
setcolor(WHITE) ;
}
// *****
// this function draws the boxes for
// indicating the status of the flags
// of BANDWIDTH BALANCING & PROBABILITY
// ASSIGNMENT & NO. OF ACTIVE STATIONS
// on the result window
// called from
// draw_result_win() in this block
// *****
void draw_status_boxes()
{
char temp_buff[10] ;

```

```

register int x_coor, y_coor, i ;

x_coor = LEFT_R_WIN + MARGIN_LEFT ;
y_coor = UP_R_WIN + MARGIN_UPPER + (8 * BOX_HEIGHT ) + (2 * MARGIN_UPPER);

setcolor(WHITE) ; // to write text
outtextxy(x_coor,y_coor,"B.W.B.") ;

x_coor += 60 ;
y_coor --=(2 * MARGIN_UPPER) - 2 ;

setcolor(BLUE) ;
setfillstyle(SOLID_FILL,BLUE) ;
rectangle(x_coor,y_coor,x_coor + 30,y_coor + BOX_HEIGHT - 2) ;
floodfill(x_coor + 1, y_coor + 1, BLUE ) ;

setcolor(YELLOW) ; // to write text in this bwb box

if(f_bwb_switch)
    outtextxy(x_coor+3, y_coor+8, " ON");
else
    outtextxy(x_coor+3, y_coor+8, "OFF");

setcolor(WHITE) ;
x_coor += 35 ;
y_coor = UP_R_WIN + MARGIN_UPPER + (8 * BOX_HEIGHT ) + (2 * MARGIN_UPPER);
outtextxy(x_coor,y_coor,"C.A. PROBS :");

x_coor = LEFT_R_WIN + (5 * BOX_LENGTH) + MARGIN_LEFT ;
y_coor --=(2 * MARGIN_UPPER) - 2 ;

setcolor(BLUE) ;
rectangle(x_coor,y_coor,x_coor + 55,y_coor + BOX_HEIGHT - 2) ;
floodfill(x_coor + 1, y_coor + 1, BLUE ) ;

setcolor(YELLOW) ; // to write text in this PROBABILITY box

if(f_prob_assign)
    outtextxy(x_coor+3, y_coor+8, "GRADED");
else
    outtextxy(x_coor+3, y_coor+8, " FLAT");

setcolor(WHITE) ;
x_coor += 62 ;
y_coor = UP_R_WIN + MARGIN_UPPER + (8 * BOX_HEIGHT ) + (2 * MARGIN_UPPER);
outtextxy(x_coor,y_coor,"ACTIVE STNS.") ;

x_coor = LEFT_R_WIN + (9 * BOX_LENGTH) + MARGIN_LEFT ;
y_coor --=(2 * MARGIN_UPPER) - 2 ;
setcolor(BLUE) ;
rectangle(x_coor,y_coor,x_coor + BOX_LENGTH-2 ,y_coor + BOX_HEIGHT - 2) ;

```

```

floodfill(x_coor + 1, y_coor + 1, BLUE ) ;
setcolor(YELLOW) ; // to write text in this STATIONS box

ultoa((long)no_of_act_stns,temp_buff,DECIMAL) ;
outtextxy(x_coor+15, y_coor+8, temp_buff) ;
setcolor(WHITE) ;

}

// *****
// clearing the result window
// called from
// 1. move_slots() in slot movement block
// 2. simulation control routine
// *****
void clear_r_win()
{
    setcolor(REFRESH_COLOR);
    setfillstyle(SOLID_FILL,REFRESH_COLOR) ;
    rectangle(LEFT_R_WIN +1, UP_R_WIN +1, RIGHT_R_WIN -1, DOWN_R_WIN -1);
    floodfill(LEFT_R_WIN +2, UP_R_WIN +2, REFRESH_COLOR);
    // filling the q_window with the refresh_color

    setcolor(BLACK) ;
    setfillstyle(SOLID_FILL,BLACK) ;
    rectangle(LEFT_R_WIN +1, UP_R_WIN +1, RIGHT_R_WIN -1, DOWN_R_WIN -1) ;
    floodfill(LEFT_R_WIN +2, DOWN_R_WIN -2, BLACK) ;
    setfillstyle(SOLID_FILL,WHITE) ;
    // filling the q_window with the BLACK
}

// ***** RESULT DISPLAY BLOCK ENDS *****

// ***** QUEUE DISPLAY BLOCK STARTS *****

// *****
// this function calls the various functions
// in this box
// called from
// 1. main()
// *****
void draw_q_win()
{
    /* -----*/
    // function declarations
    void set_sname_boxes(void);
    void set_x_y_q_boxes(void);
    void fill_snames(void);
    void draw_all_q_boxes(void);
    /* -----*/

    setcolor(WHITE);

```

```

rectangle(LEFT_Q_WIN,UP_Q_WIN,RIGHT_Q_WIN, DOWN_Q_WIN); // Q window

set_sname_boxes(); // this sets the values for x and y of the
// source name boxes in the queue window
set_x_y_q_boxes(); // setting the values of the boxes in the
// queue window
fill_snames(); // fills the names of the sources shown in the
// boxes for this purpose.
draw_all_q_boxes(); // draws all queues initially

}
// *****
// this function initializes the array containing the x,y values for source
// name boxes in the queue window
// these boxes are to be used for storing the coordinates of the name boxes
// called from
// l.draw_q_win() in this block
// *****
void set_sname_boxes()
{
    register int i;

    for(i=0;i<SOURCES_SHOWN;i++)
    {
        x_y_sname_boxes[i][0] = LEFT_Q_WIN + i * (BOX_LENGTH + 5 ) + (2*MARGIN_LEFT) ;
        x_y_sname_boxes[i][1] = UP_Q_WIN + MARGIN_UPPER;
    }
}

// *****
// this function initializes the array containing the x,y values for Q-boxes
// accesses the global array x_y_q_boxes[][]
// called from
// draw_q_win() in this block
// *****
void set_x_y_q_boxes()
{
    register int i,j,
    x_coor, // to store the x value calculated
    y_coor; // to store the y value calculated

    for(i=0;i<SOURCES_SHOWN;i++)
        for(j=0;j<Q_PACKETS_SHOWN;j++)
        {
            x_coor = LEFT_Q_WIN + (2 *MARGIN_LEFT) + i * (BOX_LENGTH + 5) ;
            y_coor = UP_Q_WIN + MARGIN_UPPER + (j+1) * (BOX_HEIGHT +5);
            x_y_q_boxes[i][j][0] = x_coor;
            x_y_q_boxes[i][j][1] = y_coor;
        }
}
// first dimension of the array represents the column
// second dimension represents the row in q_window

```

```

// third dimension represents whether it is 'X' or 'Y'
}
// *****
// this function draws the boxes for the names of the sources shown
// and fills them with the names
// called from
// 1. draw_q_win() in this block.
// *****
void fill_snames()
{
    register int i,
        x_coor,
        y_coor;
    char buffer[10];          // buffer for accepting itoa result
    char string[10];        // used for accumulating text before printing

    for(i=0;i<SOURCES_SHOWN;i++)
    {

        x_coor = x_y_sname_boxes[i][0];
        y_coor = x_y_sname_boxes[i][1];

        setcolor(BLUE);
        rectangle(x_coor,y_coor,x_coor + BOX_LENGTH ,y_coor +BOX_HEIGHT);
        // drawing the box in blue color

        strcpy(string,"S#");
        itoa(shown_sources[i]+1,buffer,DECIMAL);
        strcat(string,buffer);
        // convert the name integer to a string
        setfillstyle(SOLID_FILL,BLUE);
        // setting the fill style to solid with blue color
        floodfill(x_coor+1,y_coor+1,BLUE);
        // filling the box in blue color
        setcolor(YELLOW);
        outtextxy(x_coor+3,y_coor+5,string);
        // display the name of the source in the box in yellow
    }
    setcolor(WHITE);
}
// *****
// this function draws all the queues
// for the queue display window
// called from
// 1. draw_q_win() in this block
// *****
void draw_all_q_boxes()
{
    register int i,j,
        x_coor,
        y_coor;// for coordinates of the boxes

```

```

setcolor(WHITE); // to be drawn in WHITE color
for(i=0;i<SOURCES_SHOWN;i++)
for(j=0;j<Q_PACKETS_SHOWN;j++)
(
x_coor = x_y_q_boxes[i][j][0];
y_coor = x_y_q_boxes[i][j][1];

rectangle(x_coor,y_coor,x_coor + BOX_LENGTH,y_coor +BOX_HEIGHT);
// outer rectangle of the q_boxes
rectangle(x_coor+1,y_coor+1,x_coor + BOX_LENGTH-1,y_coor +BOX_HEIGHT-1);
// inner rectangle in the boxes for refreshing
// now made with WHITE color only
)// drawing all boxes of the queues in
// the q_window
)
//*****
// this function is meant for updation
// of source queue on the queue window
// called from
// 1. init_source_rcv()
// 2. init_source_trans()
// called only if the source is being
// displayed in the queue window
// *****
void refresh_queue(int actual_source_no)
(
register int x_coor,y_coor,
source_column_no, // the column in the q_window
color; // color of the packet
register int boxes_filled; // counter to keep track
struct packet *ptr_pkt; // for traversing the queue

source_column_no = column_shown_sources[actual_source_no];
// the column in the window is extracted
// the column no. in the window start from 0 and go upto SOURCES_SHOWN -1

ptr_pkt = head_packet_queue[actual_source_no];
// starting at the queue top

boxes_filled = ZERO;
// starting from q_top

while( ptr_pkt != NULL && (boxes_filled <Q_PACKETS_SHOWN ))
(
x_coor = x_y_q_boxes[source_column_no][boxes_filled][0];
y_coor = x_y_q_boxes[source_column_no][boxes_filled][1];
// filling the two coordinate values
boxes_filled ++;
color = pkt_color_array[ ptr_pkt -> type_of_pkt ];
// retrieving the color of the packet to be displayed
setcblor(color);

```

```

rectangle(x_coor+1, y_coor+1, x_coor + BOX_LENGTH -1,y_coor+BOX_HEIGHT -1);
setfillstyle(SOLID_FILL,color);
floodfill(x_coor +2,y_coor +2, color);
ptr_pkt = ptr_pkt -> next ; // moving forward in the queue
// fill the box with this color
)// filling the current queue status

while(boxes_filled < Q_PACKETS_SHOWN )
// if remaining boxes are to be painted empty
{
setcolor(BLACK);
x_coor = x_y_q_boxes[source_column_no][boxes_filled][0];
y_coor = x_y_q_boxes[source_column_no][boxes_filled][1];
// filling the two coordinate values
boxes_filled ++;

// now making the internal rectangle in BLACK color
rectangle(x_coor+1, y_coor+1, x_coor + BOX_LENGTH -1,y_coor+BOX_HEIGHT -1);
setfillstyle(SOLID_FILL,BLACK);
floodfill(x_coor +2,y_coor +2, BLACK);
}
}

// *****          SLOT MOVEMENT BLOCK STARTS          *****

// *****
// this function actually moves slots in the network and
// also serves the purpose of control centre
// called from
// i. main()
// *****
void move_slots()
{
/* -----*/
// function declarations
void init_source_recv(unsigned int,int) ;
void init_source_trans(unsigned int,int) ;
void draw_a_box(unsigned int,unsigned int) ;
void update_slot_no_box(unsigned int) ;
void clear_r_win(void) ;
void draw_result_win(void) ;
/* -----*/

register int curr_x, curr_y;           // store current pixel coords
register int x_l_limit,
           x_r_limit,
           y_l_limit,
           y_u_limit;                // four limits of the virtual bus

register unsigned long int i ;        // simple counter

register unsigned int curr_slot_no,   // stores the current slots being used

```

```

    actual_slot_no; // used for making calcs faster

    register int rel_x = -1;          // stores the relative (w.r.t. SS)'x'
        //      of a pixel

//***** initialization

x_l_limit = LEFT_LOCO;
x_r_limit = RIGHT + GAP;
y_l_limit = LOWER + GAP;
y_u_limit = UPPER - GAP; // initializing the four limits

    curr_slot_no = slots_passed ; // setting up to current value of
        // no. of slots passed

// ***** loop starts
for(i=0 ; i< ( NO_SLOTS_IN_RUN ) * SLOT_LENGTH ; i++)
{

    curr_slot_no = slots_passed ; // starting new traversal of bus

    // now moving on the lower horizontal arm of virtual bus

    curr_x = x_l_limit - 1;

    if(x_white_lower == INVALID)
    {
        while(getpixel(curr_x +1,y_l_limit) != WHITE && curr_x < x_r_limit)
            curr_x ++;

        if(curr_x == x_r_limit)
            x_white_lower= INVALID;
        else
            x_white_lower = curr_x +1;
    } // finding out the x_white_lower, if it is invalid

    if(x_white_lower > x_l_limit )
    {
        curr_x =x_white_lower -1;
        x_white_lower --;
    }
    else if(x_white_lower == x_l_limit)
    {
        curr_x = x_white_lower + SLOT_LENGTH -1;
        x_white_lower =x_white_lower +SLOT_LENGTH -1;
        putpixel(x_l_limit,y_l_limit,BLACK);

        slots_passed ++; // one more slot passed
        curr_slot_no ++; // entering a new slot
        if((head_slot_list[curr_slot_no % MAX_SLOTS] ->src_stn) == INVALID )
no_of_slots_empty ++ ;
        update_slot_no_box(slots_passed) ; // refreshes the slot

```



```

// no. box with this new value
}
if(x_white_lower !=INVALID)
while(curr_x < x_r_limit)
{
    putpixel(curr_x +1,y_l_limit,BLACK);
    putpixel(curr_x,y_l_limit,WHITE);

    curr_slot_no ++ ; // entering a new slot

    if(curr_x + 1 >= 55) // 55 = 'x' of source 1
    {
        rel_x = curr_x + 1 - 55;
        if(((rel_x / 9) * 9 == rel_x) && ((rel_x/9) < MAX_STNS)) //if rel_x is int. mult of 9
            if(f_act_sources[rel_x / 9] == YES)
                if(head_slot_list[curr_slot_no % MAX_SLOTS] -> src_stn == (rel_x/9))
                    // fprintf(fp,"R\t%d\t%d\n",rel_x/9,curr_slot_no);
                    // init_source_recv(curr_slot_no, (rel_x /9) );
                    // actual source no is passed to this routine
                    // if 'x' is between sources, check for ACTIVE source, and
                    // if it is found, call the source routine for receiving
                    // the slot. the source no. passed is internal no.

        curr_x +=SLOT_LENGTH;
    }
// completed the lower arm of the virtual bus

// now on the right vertical arm of the virtual bus

        curr_y = y_l_limit + 1; // reaching the lower crnr of right arm

        if(y_white_right == INVALID)
        {
            while(getpixel(x_r_limit,curr_y-1) != WHITE && curr_y >y_u_limit)

curr_y --;

        if(curr_y == y_u_limit)
            y_white_right= INVALID;
            else
                y_white_right = curr_y -1;
        } // finding out the y_white_right, if it is invalid

        if(y_white_right < y_l_limit && y_white_right !=INVALID)
        {
            curr_y = y_white_right + 1;
            y_white_right ++;
        }
    else if(y_white_right == y_l_limit)

```

```

        curr_y = y_white_right - SLOT_LENGTH + 1;
        y_white_right = y_white_right - SLOT_LENGTH + 1;
        // putpixel(x_r_limit, y_l_limit, BLACK);
    }
    if(y_white_right != INVALID)
while(curr_y > y_u_limit)
{
    putpixel(x_r_limit, curr_y-1, BLACK);
    putpixel(x_r_limit, curr_y, WHITE);

    curr_slot_no ++ ; // entering a new slot

        curr_y -= SLOT_LENGTH;
    }
// completed the right arm of the virtual bus

// now moving to the upper arm of the virtual bus
curr_x = x_r_limit + 1;

    if(x_white_upper == INVALID)
    {
        while(getpixel(curr_x - 1, y_u_limit) != WHITE && curr_x > x_l_limit)
            curr_x --;
        if(curr_x == x_l_limit)
            x_white_upper = INVALID;
    }
    else
        x_white_upper = curr_x - 1;
} // finding out the x_white_upper, if it is invalid

    if(x_white_upper < x_r_limit && x_white_upper != INVALID)
    {
        curr_x = x_white_upper + 1;
        x_white_upper ++ ;
    }
    else if(x_white_upper == x_r_limit)
    {
curr_x = x_white_upper - SLOT_LENGTH + 1;
        x_white_upper = x_white_upper - SLOT_LENGTH + 1;

        // putpixel(x_r_limit, y_u_limit, BLACK);
    }
    if(x_white_upper != INVALID)
while(curr_x > x_l_limit)
{
    putpixel(curr_x - 1, y_u_limit, BLACK);
    putpixel(curr_x, y_u_limit, WHITE);

    curr_slot_no ++ ; // entering a new slot

    if(curr_x - 1 >= 55) // 55 = 'x' of source 1
    {

```

```

rel_x = curr_x - 1 - 55;
if(((rel_x / 9) * 9 == rel_x) && ((rel_x/9) < MAX_STNS)) //if rel_x is int. mult of 9
    if(f_act_sources[rel_x / 9] == YES)
//      fprintf(fp,"T\t%d\t%d\n",rel_x/9,curr_slot_no);
        init_source_trans(curr_slot_no,(rel_x / 9) );
) // if 'x' is between sources, check for ACTIVE source, and
// if it is found, call the source routine for transmitting
// the slot. the source no. passed is internal no.

curr_x -=SLOT_LENGTH;
) // completed the upper arm of the virtual bus
if( (i/SLOT_LENGTH)*SLOT_LENGTH == i)
(
// generate a new pulse and put a new slot structure
putpixel(x_l_limit,y_u_limit,WHITE);
// initialization of the corresponding slot structure has also to
// done at this very place.

//      curr_slot_no ++ ;
actual_slot_no = curr_slot_no % MAX_SLOTS;

head_slot_list[actual_slot_no] -> packet_id = INVALID;
head_slot_list[actual_slot_no] -> src_stn = INVALID;
head_slot_list[actual_slot_no] -> type_of_pkt = EMPTY_SLOT;

//      draw_a_box(curr_slot_no,actual_slot_no); // refreshing the r_win
)

) // ***** end of loop
)
// ***** SLOT MOVEMENT BLOCK ENDS *****

// ***** STATIONS' INTERNAL ROUTINES BLOCK *****
// this block contains routines for the various source distributions
// this includes routines for invoking them, and managing queues of
// the ACTIVE sources

// *****
// this function invokes a source's receiving packet function
// called from
// 1. mov_slots() from slot movement block
// *****
// This routine is activated only when the curr slot has the data transmitted
// by this very source . Now only to copy back the data onto the packet store
// *****
void init_source_recv( unsigned int curr_slot_no, int actual_source_no )
{
/* -----*/
// function declarations

```

```

void send_to_packet_store(struct packet *);
struct packet * search_remove_return_pkt(struct slot *,int,unsigned int) ;
void refresh_queue(int) ;
/* -----*/

register unsigned int actual_slot_no ; // (internal value)
struct packet * ptr_temp;

ptr_temp = head_packet_queue[actual_source_no];

// if the head_packet_queue is NULL , since this packet has already
// been acknowledged and thus removed from the source and so simply
// return to the calling routine .
if(ptr_temp == NULL)
    return ; // because of reason given above ;

actual_slot_no = curr_slot_no % MAX_SLOTS ;

ptr_temp = search_remove_return_pkt(head_slot_list[actual_slot_no],actual_source_no, curr_slot_no) ;
// trying to find, remove and extract the packet in the queue which is
// filled in this slot

if(ptr_temp == NULL)
    return ; // this packet not found in the packet queue
else
(
    ptr_temp -> slot_no_reach_destn = curr_slot_no + ends_gap(actual_source_no) ;
    send_to_packet_store(ptr_temp) ; // send the data to backup
//    if(f_shown_sources[actual_source_no] == YES )
// refresh_queue(actual_source_no) ; // if this source active, display the queue
    free(ptr_temp) ;

    no_of_pkts_in_q[actual_source_no] -- ;
    if(no_of_pkts_in_q[actual_source_no] == ZERO )
    (
if(f_special_status[actual_source_no] == YES )
(
    f_special_status[actual_source_no] = NO ;
    if(empty_special_entry < SERIAL_LENGTH )
        serial_special_status[empty_special_entry++] = actual_source_no ;
    )
else if(f_killer_status[actual_source_no] == YES )
(
    f_killer_status[actual_source_no] = NO ;
    if(empty_killer_entry < SERIAL_LENGTH )
        serial_killer_status[empty_killer_entry ++] = actual_source_no ;
    )
)

min_wait[actual_source_no] = normal_min_wait[actual_source_no] ;
)
)

```

```

}
// *****
// this function searches a pkt in the
// queue of a source which is present
// in the slot passed ; it removes that
// packet from the queue ; updates the
// access wait counter ; returns this
// packet to the calling routine
// called from
// 1. init_source_recv() in this block
// *****
struct packet *search_remove_return_pkt(struct slot *ptr_slot , int actual_source_no, unsigned int curr_slot_no)
{
    struct packet *ptr_temp1, *ptr_temp2 ; // temp ptrs to the list

    ptr_temp1 = head_packet_queue[actual_source_no] ;
    // null condition already checked in calling routine init_source_recv()

    if(( ptr_temp1 -> type_of_pkt ) == (ptr_slot -> type_of_pkt) )
        if((ptr_temp1 ->packet_id) == (ptr_slot -> packet_id) )
        {
            // now checking and updating reach top entry in the
            // next packet in the queue, if any !
            ptr_temp2 = ptr_temp1 -> next ;
            if(ptr_temp2 != NULL)
                if(ptr_temp2 -> slot_no_reach_top == ZERO )
                    ptr_temp2 -> slot_no_reach_top = curr_slot_no + ends_gap(actual_source_no) ;
            // moving the second node to the top of the queue
            // and resetting the access wait counter to zero
            head_packet_queue[actual_source_no] = ptr_temp2 ;
            access_wait[actual_source_no] = ZERO ;
            return(ptr_temp1) ;
        }

    // Now start checking further in the list
    // access wait now need not be reset
    // starting again from the top point

    ptr_temp2 = ptr_temp1 = head_packet_queue[actual_source_no] ;

    while((ptr_temp1 != NULL) &&(((ptr_temp1->type_of_pkt) !=(ptr_slot ->type_of_pkt))
        ||((ptr_temp1 ->packet_id) !=(ptr_slot->packet_id))))
    {
        ptr_temp2 = ptr_temp1 ;
        ptr_temp1 = ptr_temp1 -> next ;
    }

    if((ptr_temp1 != NULL) &&((ptr_temp1->type_of_pkt) ==(ptr_slot ->type_of_pkt))
        && ((ptr_temp1 ->packet_id) ==(ptr_slot->packet_id)))
    {
        ptr_temp2 -> next = ptr_temp1 -> next ;
        return(ptr_temp1) ;
    }
}

```

```

    )
    else
        return((struct packet *)NULL) ;
}

// *****
// this function copies the contents of a node
// onto the packet store
// called from
// 1. init_source_recv() in this block
// *****
void send_to_packet_store(struct packet * ptr_pkt)
{
    /* -----*/
    // function declaration
    void refresh_packet_store(void) ;
    void calc_delays_in_unit() ;
    /* -----*/
    // trying to check whether the storage is already full
    // but this is not a fool proof check
    // modify this !

    register int i; // just for printing packet onto file

    if(curr_empty_entry == NO_PKTS_IN_UNIT)
    {
        calc_delays_in_unit() ;
        refresh_packet_store() ;
    } // If the packet store is full, calc delays, refresh the store and continue

    // copying the data in the packet onto the packet store
    packet_store[curr_empty_entry][0] = ptr_pkt -> src_stn;
    packet_store[curr_empty_entry][1] = ptr_pkt -> type_of_pkt;
    packet_store[curr_empty_entry][2] = ptr_pkt -> slot_no_gen;
    packet_store[curr_empty_entry][3] = ptr_pkt -> slot_no_reach_top;
    packet_store[curr_empty_entry][4] = ptr_pkt -> slot_no_reach_destn;

    curr_empty_entry ++;
    if(curr_empty_entry == NO_PKTS_IN_UNIT)
    {
        calc_delays_in_unit() ;
        refresh_packet_store() ;
    } // Repeat the earlier checking for the storage
}

// *****
// this function only refreshes the packet store
// called from
// 1. init_all() in initialization block
// 2. send_to_packet_store() in dsttrbtn block
// 3. simulation() in the simulation control routine
// *****

```

```

void refresh_packet_store()
{
    register int i,j;

    for(i=0;i<NO_PKTS_IN_UNIT;i++) // for all rows
        for(j=0;j<COLUMN_IN_PKT_STORE;j++) // for all columns
            packet_store[i][j] = ZERO;
    curr_empty_entry = ZERO;
}

// *****
// this function invokes a source's generating packet function
// called from
// 1. mov_slots() from slot movement block
//*****
void init_source_trans(unsigned int curr_slot_no,int actual_source_no)
{
    /* -----*/
    // function declarations
    void put_in_queue(struct packet *,int,unsigned int) ;
    int give_random_no(void) ;
    // int exponential(int) ;
    void put_in_slot(int,unsigned int) ;
    struct packet * create_packet(void) ;
    void refresh_queue(int) ;
    /* -----*/

    register int i, j, no_of_pkts_gen =0, rand_no ;
    float rand_prob ; // random no. for probab. of transmission
    int temp_buff;
    struct packet *ptr_temp;

    // if all of the counters are non-zero return after
    // decrementing them

    if(access_wait[actual_source_no] !=ZERO)
    if(wait_counter[actual_source_no][VIDEO_DATA -1] != ZERO)
    if(wait_counter[actual_source_no][VOICE_DATA -1] != ZERO)
    if(wait_counter[actual_source_no][ORDINARY_DATA -1] != ZERO)
    {
        // now at this pt., all these counters have non zero
        // values, which means that neither any new traffic
        // is to be generated nor any attempt of transmission
        // is to be made . So return after decrementing these
        // four counters

        // decrementing the three type counters
        // and the access wait counter
        access_wait[actual_source_no] --;
        for(i=0;i<NO_OF_DATA_TYPES;i++)
            wait_counter[actual_source_no][i] --;
    }
}

```

```

    return;
}

// if above check fails, we proceed further

// now the channel access attempt shall be made
// this is done prior to generation of new packets.
// since the packets can be generated at any point in the slot
// while the channel can only be accessed at the start of the slot

// TRANSMISSION ATTEMPT

ptr_temp = head_packet_queue[actual_source_no];
// accessing the q_top packet for this source
if( access_wait[actual_source_no] != ZERO )
    access_wait[actual_source_no] -- ;
else
{
    if(ptr_temp !=NULL)
    {
        rand_prob = ((float)rand())/RAND_MAX;

        if( rand_prob <= prob_base[actual_source_no] )
        {
            put_in_slot(actual_source_no,curr_slot_no);
            // put the packet in the slot ; itself updates te result window slot box
            access_wait[actual_source_no] = min_wait[actual_source_no];
        }
    }
}
// if access wait is zero, try to send and set the wait to min_wait
// else decrement the wait counter by 1.
// TRANSMISSION ATTEMPT OVER

// now the individual traffic distributions
// shall be called as follows
// if the wait counter for a particular type of traffic
// becomes zero, invoke the random number generator for that
// traffic and generate as many pkts of that type. Generate
// a random number to serve the purpose of packet id for
// identification of the packets . The video traffic is steady
// traffic as against the voice and data traffics, which are bursty
// in nature and so video traffic is represented by one packet
// generated per fixed time interval (represented by MEAN_VIDEO)

// ORDINARY DATA GENERATION
if(wait_counter[actual_source_no][ORDINARY_DATA -1] != ZERO)
    wait_counter[actual_source_no][ORDINARY_DATA -1] -- ;
else
{
    no_of_pkts_gen = give_random_no(); // generate no.(rand) of pkts in this data spurt
}

```



```

if(no_of_pkts_gen != ZERO)
{
    rand_no = rand() ; // random packet id
    for(j=0;j<no_of_pkts_gen;j++)
    {
        ptr_temp = create_packet();
        // create pkts, fill entries and put them in queue
        ptr_temp -> packet_id = rand_no + j;
        ptr_temp -> src_stn = actual_source_no ;
        // the src stored in the packet is internal value
        // please make this sure again !
        ptr_temp -> type_of_pkt = ORDINARY_DATA;
        ptr_temp -> slot_no_gen = curr_slot_no ;
        put_in_queue(ptr_temp,actual_source_no,curr_slot_no);
    }
}
wait_counter[actual_source_no][ORDINARY_DATA -1] = MEAN_ORD;

no_of_pkts_in_q[actual_source_no] += no_of_pkts_gen ; // at every generation
} //ORDINARY DATA GENERATION OVER

// VIDEO DATA GENERATION
if(wait_counter[actual_source_no][VIDEO_DATA -1] != ZERO)
    wait_counter[actual_source_no][VIDEO_DATA -1] -- ;
else
{
    // generate a video packet, fill it
    ptr_temp = create_packet() ;
    rand_no = rand() ; // random packet id

    ptr_temp -> packet_id = rand_no ;
    ptr_temp -> src_stn = actual_source_no ;
    ptr_temp -> type_of_pkt = VIDEO_DATA ;
    ptr_temp -> slot_no_gen = curr_slot_no ;
    put_in_queue(ptr_temp,actual_source_no,curr_slot_no);
    wait_counter[actual_source_no][VIDEO_DATA -1] = MEAN_VIDEO ;
    no_of_pkts_in_q[actual_source_no] ++ ;
}

// VIDEO DATA GENERATION OVER

// VOICE DATA GENERATION
if(wait_counter[actual_source_no][VOICE_DATA -1] != ZERO)
    wait_counter[actual_source_no][VOICE_DATA -1] -- ;
else
{
    no_of_pkts_gen = give_random_no(); // generate no.(rand) of pkts in this data spurt
    no_of_pkts_in_q[actual_source_no] += no_of_pkts_gen ;
    if(no_of_pkts_gen != ZERO)
    {
        rand_no = rand() ; // random packet id
    }
}

```

```

for(j=0;j<no_of_pkts_gen;j++)
(
ptr_temp = create_packet();
// create pkts, fill entries and put them in queue
ptr_temp -> packet_id = rand_no + j;
ptr_temp -> src_stn = actual_source_no ;
// the src stored in the packet is internal value
// please make this sure again !
ptr_temp -> type_of_pkt = VOICE_DATA;
ptr_temp -> slot_no_gen = curr_slot_no ;
put_in_queue(ptr_temp,actual_source_no,curr_slot_no);
)
)
do // ensuring the spurt gap is not differing by > 10
(
rand_no = random(MEAN_VOICE + 10);
) while(abs(rand_no - MEAN_VOICE) > 10);
wait_counter[actual_source_no][VOICE_DATA - 1] = rand_no;
)
// VOICE DATA GENERATION OVER

// REFRESHING THE RESULTING QUEUE

// if(f_shown_sources[actual_source_no] == YES )
// refresh_queue(actual_source_no) ;

if(no_of_pkts_in_q[actual_source_no] >= NORMAL_Q_LEN_LIMIT)
(
if((f_special_status[actual_source_no] == NO) && (f_killer_status[actual_source_no] == NO))
(
f_special_status[actual_source_no] = YES ;
if(empty_special_entry < SERIAL_LENGTH )
serial_special_status[empty_special_entry ++] = actual_source_no ;
min_wait[actual_source_no] = normal_min_wait[actual_source_no]/ TWO ;
)
if(no_of_pkts_in_q[actual_source_no] >= UPPER_Q_LEN_LIMIT)
if(f_killer_status[actual_source_no] == NO)
(
f_special_status[actual_source_no] = NO ;
if(empty_special_entry < SERIAL_LENGTH )
serial_special_status[empty_special_entry ++] = actual_source_no ;
if(empty_killer_entry < SERIAL_LENGTH )
serial_killer_status[empty_killer_entry ++] = actual_source_no ;
f_killer_status[actual_source_no] = YES ;
min_wait[actual_source_no] = (normal_min_wait[actual_source_no])/ FOUR ;
)
)
)
)
// *****

```

```

// this function creates a new packet node and initializes it
// called from
// 1. init_source_trans() in dstribtn block
// *****
struct packet *create_packet()
{
    struct packet *ptr_temp;

    ptr_temp=(struct packet *) malloc(size_of_packet);
    if(ptr_temp == NULL)
    {
        closegraph();
        clrscr();
        puts(" Memory not allocated in module create_packet()\n");
        puts(" System fault !\n Please check it!\n Goodbye!");
        exit(1);
    }
    // checking for memory allocation failure
    ptr_temp -> src_stn = RAND_MAX;
    ptr_temp -> type_of_pkt = ZERO;
    ptr_temp -> slot_no_gen = ZERO;
    ptr_temp -> slot_no_reach_top = ZERO;
    ptr_temp -> slot_no_reach_destn = ZERO;

    ptr_temp ->next = NULL;

    return(ptr_temp);          // no comments needed, it is simple initialization
}

// *****
// this function puts the packet on the
// top of the queue of the source passed
// on the current slot
// called from
// 1. init_source_trans() in this block
// *****
void put_in_slot(int actual_source_no, unsigned int curr_slot_no)
{
    /* -----*/
    // function declarations
    void draw_a_box(unsigned int, unsigned int) ;
    /* -----*/
    struct packet * ptr_pkt;
    register unsigned int actual_slot_no;

    ptr_pkt = head_packet_queue[actual_source_no];
    // accessing the q_top packet
    actual_slot_no = curr_slot_no % MAX_SLOTS;
    // finding the internal slot corresponding to this current slot

    // now copying the data from packet to slot

```

```

head_slot_list[actual_slot_no] -> packet_id = ptr_pkt -> packet_id;
head_slot_list[actual_slot_no] -> src_stn = ptr_pkt -> src_stn;
head_slot_list[actual_slot_no] -> type_of_pkt = ptr_pkt -> type_of_pkt;

// now updating this slot in the result window
// draw_a_box(curr_slot_no,actual_slot_no) ;
}

// *****
// this function puts the packet node passed
// to it in appro. place in the Q of
// the source ->actual_source_no
// called from
// 1. init_source_trans() in this block
// *****
void put_in_queue(struct packet *ptr_pkt,int actual_source_no, unsigned int curr_slot_no)
{

register int this_pkt_type; // to keep new pkt type for fast comparisons
struct packet * ptr1_temp, *ptr2_temp; //temporary pointers for traversal

// checking for the queue empty case!
if(head_packet_queue[actual_source_no] == NULL)
{
ptr_pkt -> slot_no_reach_top = curr_slot_no ;
head_packet_queue[actual_source_no] = ptr_pkt ;
access_wait[actual_source_no] = ZERO ;
return;
} // if the queue is empty, put the packet at the
// top of the queue and fill the entry for reaching
// the top of the queue; reset the access_wait counter
// to ZERO value

this_pkt_type = ptr_pkt -> type_of_pkt; // otherwise, starting operation

ptr1_temp = ptr2_temp = head_packet_queue[actual_source_no];
// starting giving value of HEAD

while(ptr1_temp ->next != NULL && ptr1_temp ->type_of_pkt >=this_pkt_type)
{
ptr2_temp = ptr1_temp;
ptr1_temp = ptr1_temp -> next;
} // move in the link list till the pt. of insertion comes -> by ptr1_temp

// when list has only one node or say pt. of insertion is at the
// head itself, then the two pointers are equal
if(ptr1_temp == ptr2_temp)
{
if((ptr1_temp -> type_of_pkt) < this_pkt_type)
{
ptr_pkt -> slot_no_reach_top = curr_slot_no ;
ptr_pkt -> next = ptr1_temp;
}
}
}

```

```

    head_packet_queue[actual_source_no] = ptr_pkt;
    access_wait[actual_source_no] = ZERO ;
    return;
} // fixing the new packet at the top of queue
// resetting the access_wait counter to ZERO
// filling the entry for reaching the top of the queue
else
    ptr1_temp ->next = ptr_pkt;
}
else
{
    if((ptr1_temp -> type_of_pkt) < this_pkt_type)
    {
        ptr_pkt ->next = ptr1_temp;
        ptr2_temp ->next = ptr_pkt;
    }
    else
        ptr1_temp -> next = ptr_pkt;
}
}

// *****
// this function gives the no. of pkts per
// message for the ord and video traffic
// according to the flag type
// called from
// 1. init_source_trans() in this block
// *****
int give_random_no()
{
    double avg_numbers = 0.0 ;
    unsigned int pkts ;
    double prod = 1.0 , p_final, rand_float_no ;
    double variance ;

    p_final = exp( - BASE_FOR_RANDOM );

    pkts = ZERO ;

    while( prod > p_final )
    {
        rand_float_no = ((double)rand())/RAND_MAX;
        prod *= rand_float_no ;
        pkts++;
    }

    tried += 1 ; // increase the number of times this function has been
                // invoked

    sum_numbers += pkts ; // increment the no. of pkts generated so far

    avg_numbers = (float) sum_numbers / (float)tried ; // ( X_BAR )

```

```

deviation_sum += ((float)pkts - avg_numbers) * ((float)pkts - avg_numbers ); // ( SUM += X_i square )
if( tried >= try_upto )
{
    f_try_limit_reached = YES ;

    variance = (double)(deviation_sum/try_upto) ;

    try_upto = Y_VALUE * Y_VALUE * variance / (CONFIDENCE_LIMIT * CONFIDENCE_LIMIT) ;

    if( tried >= try_upto )
        f_simul_over = YES ;
    else
        f_try_limit_reached = NO ;
}

return( pkts - 1 ); // now 'pkts - 1' is being returned ,so modify
// other routines after seperate testing
}
#endif
// *****
// this function gives a exponentially
// distributed nummber with a mean as
// passed to this routine
// called from
// 1. init_source_trans() in this block
// *****
int exponential(int mean)
{
    double rn, retval ;
    int ret;

    // rand() returns a random no. from 0 to RAND_MAX
    do
    {
        rn = ((double)rand()) / RAND_MAX;
    } while( rn == 0.0 );

    retval = ( mean ) * log(rn);
    ret = (int) retval;

    return(-ret);
}
#endif
// ***** STATION DISTRIBUTION BLOCK ENDS *****
// ***** RESULT CALCULATION BLOCK STARTS *****
// *****
// this function calculates all the
// delays needed for the result over
// the current unit of packet store

```

```

// called from
// 1. master control routine
// 2. simulation control routine
// make it clearer
// *****
void calc_delays_in_unit()
{
    register int i,j;
    register unsigned int src,type.slot_gen,slot_q_top,slot_destn;
    register int q_delay, comm_delay ;

    for(i=0;i<NO_PKTS_IN_UNIT && i<curr_empty_entry ;i++)
    {
        src = packet_store[i][0];

        if((src >= ZERO) && ( src < MAX_STNS))
        {
            type = packet_store[i][1] ;
            total_pkts_trans[src][type-1] ++ ;
            q_delay = packet_store[i][3] - packet_store[i][2] ;
            total_q_delay[src][type -1] += (long)q_delay ;
            comm_delay = packet_store[i][4] - packet_store[i][3] - d_point_offset ;
            total_comm_delay[src][type -1] += (long)comm_delay ;
            if((q_delay + comm_delay) > delay_allowed[type-1] )
                no_of_pkts_delayed[src][type-1] ++ ;
        }
    }
}

fp = fopen(filename,"a") ;
fprintf(fp,"<Results-RUN: Stn= %d,",no_of_act_stns) ;
fprintf(fp,"BWB= %d,Prob= %d>\n",f_bwb_switch,f_prob_assign) ;

    for(i=0;i<MAX_STNS;i++)
    {
        if(f_act_sources[i] == YES)
        {
            fprintf(fp,"\n<Src-%d>,", i+1) ;
            for(j=0;j<NO_OF_DATA_TYPES;j++)
            {
                fprintf(fp,"<Type-%d,Pkts-%d,Q_d-%ld,",j+1,total_pkts_trans[i][j],total_q_delay[i][j]) ;
                fprintf(fp,"C_d-%ld,Deld-%d>",total_comm_delay[i][j],no_of_pkts_delayed[i][j]) ;
            }
        }
    }
    fprintf(fp,"\n");
    fclose(fp) ;
}

// *****RESULT CALCULATION BLOCK OVER *****
// ***** simulation control block *****

```

```

// *****
// This function takes the charge of running
// the simulation after the main() has drawn
// the screen and its components after the
// initialization of the variables .
// called from
// main() in the main block
// *****
void simulation(void)
{
    /* -----*/
    // function declarations
    void move_slots(void) ;
    void calc_results(void) ;
    void calc_delays_in_unit(void) ;
    void refresh_packet_store(void) ;
    void show_std_results(void) ;
    void clear_r_win(void) ;
    void draw_result_win(void) ;
    /* -----*/
    register int i,j ;

    // do
    // {
    //   move_slots() ; // warming up the network
    //   // but the calculations should not include
    //   // the packets reached within this time interval. SO
    //   refresh_packet_store() ;// not done for results
    do
    {
        move_slots() ;
    } while((slots_passed < MAX_SLOTS_TO_RUN)&&(f_mood_over == NO)) ;
    // } while(!f_simul_over) ;
    calc_delays_in_unit() ;
    for(i=0;i<50;i++)
        printf("\a");
    fp=fopen(filename,"a") ;
    fprintf(fp,"\n<THESE WERE THE FINAL RESULTS>");
    fprintf(fp,"<THE CALCULATED R ESULTS ARE >\n");
    for(i=0;i<MAX_STNS;i++)
    {
        if(f_act_sources[i] == YES )
        {
            for(j=0;j<NO_OF_DATA_TYPES;j++)
            {
                if(total_pkts_trans[i][j] !=ZERO)
                {
                    fprintf(fp,"\nq_delay[%d][%d]=%f",i+1,j,(float)total_q_delay[i][j]/(float)total_pkts_trans[i][j]
                    fprintf(fp,"| comm_delay[%d][%d]=%f",i+1,j,(float)total_comm_delay[i][j]/(float)total_pkts_tran
                ]]); ;
                }
            }
        }
    }
}

```



```
)  
fclose(fp) ;  
)  
// *****simulation block control over *****
```