Simulation of Gigabit Ethernet and Measuring its Performance against Fast Ethernet

Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree

of

Master of Engineering

in

COMPUTER TECHNOLOGY & APPLICATION

By

PAWAN

(10/CO/01)

Under the supervision of

DR. S.K.SAXENA

Department of Computer Engineering

Delhi College of Engineering of

Delhi University

Delhi -110 042

CERTIFICATE

It is certify that the work which is being presented in this dissertation entitled "**SIMULATION OF GIGABIT ETHERNET AND MEASURING ITS PERFORMANCE AGAINST FAST ETHERNET"** in partial fulfillment of the requirement for the award of degree of **'Master of Engineering'** in 'Computer Technology and Applications' submitted by **Pawan** Roll Number 10/CO/01 to the Department of Computer Engineering, **Delhi College of Engineering**. The work embodied in this dissertation has not been submitted for the award of any other degree under the supervision and guidance of **Dr. S.K. Saxena, Department** of Computer Engineering, and Delhi College of Engineering. The work embodied in this dissertation has not been submitted for award of any other degree by the student to the best of our Knowledge..

Dr. D. Roy Choudhury
Professor and Head
Department of Computer Engineering
Delhi College of Engineering
Delhi University.

Dr. S. K. Saxena
Department of Computer Engineering
Delhi College of Engineering

Place: Delhi
Date :

Acknowledgements

It is my great pleasure to acknowledge a debt of gratitude to my project guide, **Dr. S. K. Saxena,** Delhi College of Engineering, Delhi for his expert guidance, valuable advice, continued encouragement during all stages of the project. The help, co-ordination and patience with which he attended to my problems and solved them, have contributed a lot to the completion of my project..

I would also like to thank HOD Prof. D.R. Choudhury and all the professors of Computer Technology & Application Department, of Delhi College of Engineering, Delhi, and also the Librarian of IIT Delhi, along with all my classmates and other students, whose suggestions and help in this regard proved to be very valuable. I would like to thank all my classmates for the discussions, which I made with them and for their cooperation throughout this work.

Finally, I thank for Lab assistant for their coordination in the Computer Laboratory.

Date:

PAWAN
(10/CO/01)
M.E. (CTA)

Contents

Abstract

The accelerating growth of LAN traffic is pushing network administrators to look for even higher-speed backbone network technology, which can be compatible with existing Ethernet installation. Gigabit Ethernet is one such higher speed network technology. . Gigabit Ethernet uses the same frame format, and functions as its 10Mbps and 100Mbps precursors. It offers1000Mbps bandwidth and uses the same CSMA/CD protocol. This works evaluates the performance of Gigabit Ethernet and also presents a comparative study of Gigabit Ethernet against Fast Ethernet. It is shown that half-duplex Gigabit Ethernet achieves over 70% throughput with a 100% offered load. It is also shown that as packet size increases, the performance of Gigabit Ethernet Exceeds the performance of Fast Ethernet.

Chapter-1

INTRODUCTION

1.1. Evolution of Gigabit Ethernet

As companies rely on application like electronic mail and database management for core business operations, computer networking becomes increasingly more important. A network is a set of communication links for interconnection a collection of terminals, computers, telephones, printer or other type of data handling devices. These devices are refereed to as stations. A local Area network (LAN) allows multiple stations within a building or campus area to exchange information among themselves. In the late 1970s and early 1980s LANs made a dramatic entrance into the communication scene. Now Ethernet is the world's most pervasive LAN technology. It operates at the data rate of 10Mbps, and employs Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol to access the common channel.

Since modern organization depend upon their LANs to provide connection for a growing number of complex desktop computer applications, the volume of the network traffic increases. As the volume of network increases, the bandwidth offered by a 10Mbps Ethernet LAN becomes inadequate to maintain acceptable performance. This created a need for high-speed networks.

As the high-speed LAN becomes the leading choice. Fast Ethernet provides 10-times higher bandwidth than 10-Mbps Ethernet and is compatible with existing Ethernet This establishing Ethernet as a scalable technology. But the growing use of Fast Ethernet connection to servers and desktops, however is creating a clear need for an even higher-speed network technology at the backbone and server level.

Before Gigabit Ethernet appeared, ATM was the ultimate promise for handling the traffic at the backbone level. But ATM consumes 25-30 percent bandwidth overhead as LAN frames are fragmented and turned into cells. Its ever-increasing complexities delayed atm's real world debut. Even now, fundamental ATN standards are still being defined, and the technology is just beginning to roll out with capabilities needed for general use.

But the newly and timely arrived, Gigabit Ethernet is a 1000Mbps (1Gbps) extension of the same CSMA/CD protocol for accessing the channel and uses the same Ethernet frame format. From 10Mbps Ethernet to 100Mbps Fast Ethernet, there is an enormous installed base of Ethernet drivers, and adapters that Gigabit Ethernet perfectly fits as a backbone core.

1.2Organization of the Thesis

The objective of this work is to

1. Simulate Gigabit Ethernet using Discrete Event Simulation Technique.
2. Analyze Gigabit Ethernet's performance using simulator.
3. Compare the performance of Gigabit Ethernet with that of Fast Ethernet.

1.3 Organization of The Thesis

The thesis has six chapters. Chapter-1 is introductory. Ethernet and it's operation are given in Chapter-2. An overview of Fast Ethernet is also included in this chapter. Chapter – 3 gives a detailed view of Gigabit Ethernet. Chapter-4 introduces some of the important discrete event simulation concepts and the implementation details of Gigabit Ethernet simulator for various inputs. It also gives an interpretation to these results. The conclusion and future scope of work have been included in Chapter-6.

Chapter-2

Ethernet and Fast Ethernet

2.1 Introduction

Ethernet is a multi-access, packet –switched communication system for carrying digital data among locally distributed systems. Ethernet is characterized by use of specified packet format and set of rules for transmitting packets.

This chapter deals in detail about Ethernet and Fast Ethernet. This chapter begins with an important family of standards for LAN developed by IEEE.Section 2.3 gives information about Ethernet's packet format. Section 2.4 gives Ethernet's operation in detail. Section 2.6 presents an overview of Fast Ethernet. Finally, the chapter ends with a conclusion in section 2.7.

2.2 IEEE LAN Standards

The shared communication channel in an Ethernet is a passive broadcast medium with no central control; packet address recognition is used in each to take packets from the channel. Access to the channel by stations wishing to transmit is coordinated in distributed fashion by stations themselves, using a statistical arbitration scheme. IEEE developed a family of standards for LANs to enable equipment of variety of manufactures to interface. This family is called the IEEE 802 family. The components of the IEEE-802 standard are as follows:

802.1 Overview, Internetworking, and Systems management.

802.2 Logical Link Control (LLC)

802.3 CSMA/CD bus

802.4 Token bus.

802.5 Token ring.

802.6 Metropolitan Area Networks (MANs)

802.7 Advisory group for broadband transmission.

802.8 Advisory group for fiber optics.

802.9 Integrated voice and data LANs

802.10 Standard for Interoperable LAN security(SILS) WG

802.11 Wireless LAN WG

802.12 Demand Priority WG

A comparison of the IEEE-802standards to the OSI Model is shown here:

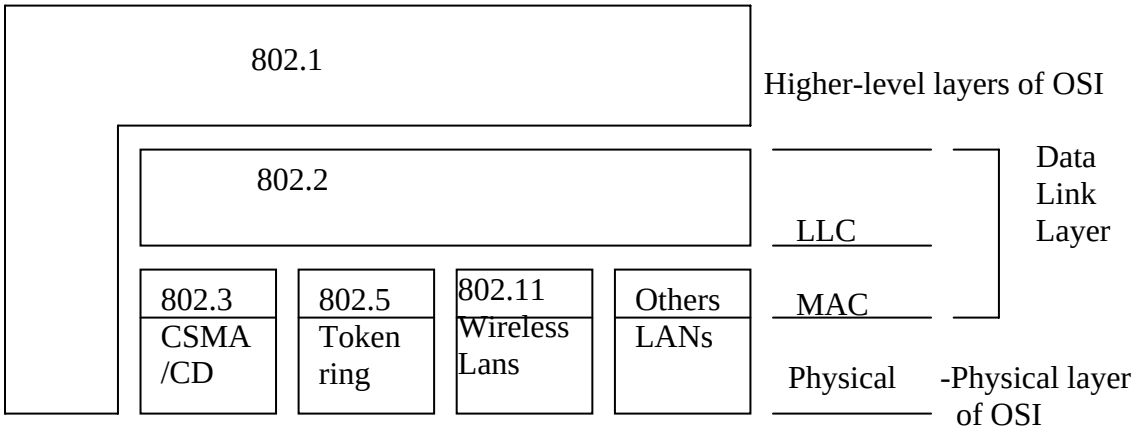

Fig.2.1 IEEE family of protocols

The IEEE-802 data link layer is divided into following two sublayers:

1.Logical Link Control.

2. Medium Access Control.

The MAC layer is responsible for using either a random or a token procedure. For controlling access to the channel, and it handles frame delimiting, address recognition and error-checking functions. The LLC layer, which is above the MAC layer, provides two types of user services: an unacknowledged connectionless service and a set of connection-oriented services.

2.3 Ethernet Packet Format

Stations using Ethernet must be to transmit and receive packets

On the common channel with the packet format and spacing as shown here:

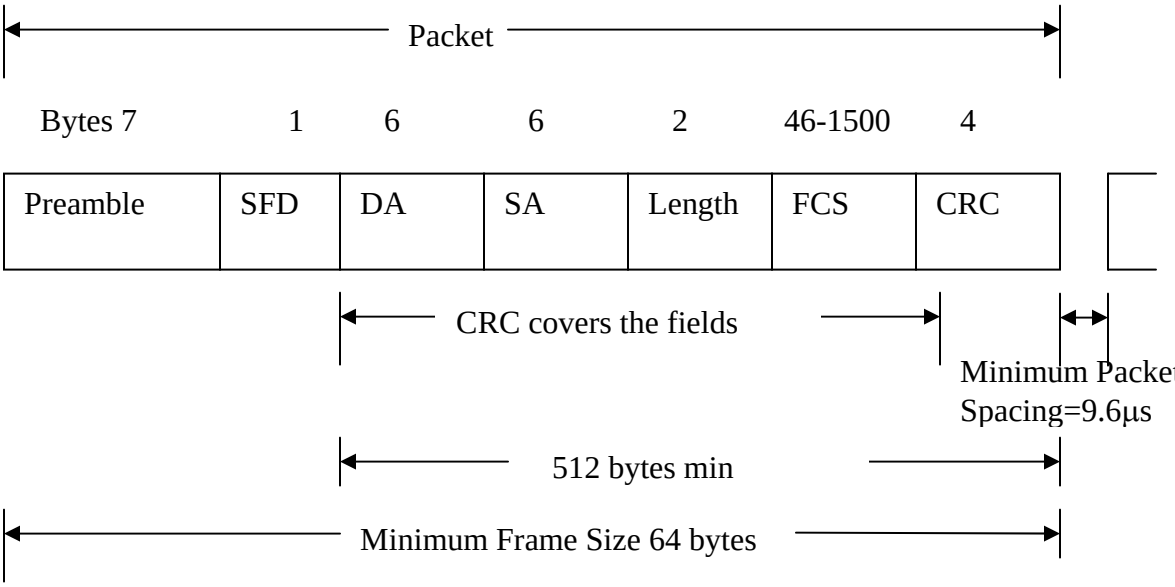

Fig: 2.2 Ethernet Packet Format

Each packet is a sequence of 8-bit bytes. The least significant bit of each byte is transmitted first. The fields are defined as follows:

Preamble:

This is a 56-bit (7-byte) synchronization pattern consisting of alternating ones and Zeros, which is used to ensure receiver synchronization.

Start-Frame Delimiter:

The 1-byte start-frame delimiter is similar to the preamble, except that it ends with two consecutive one bits. Thus, 10101011 give the start-frame delimiter.

Destination Address:

This 48-bit field specifies to which station the packet is addressed. The address Can be either a unique station on the Ethernet, or it can be a multidestination address.

Source Address:

The two-byte field indicates the number of information bytes being supplied

by the LLC layer for the data field.

Data field:

The IEEE-802 Standard recommends that the data field has a length between 46 and 1500 bytes. The 46-bytes minimum ensures that valid packet will distinguishable from collision fragment. Thus Ethernet has a minimum ensures that valid packet will be distinguishable from collision fragment. Thus Ethernet has minimum frame size of 64 – bytes(excluding preamble 7-bytes and start-frame delimiter 1-byte).Another important reason for having a minimum frame size is to prevent a station from completing the transmission of a frame before the first bit has reached the far end of the cable , where it may collide with another frame. Therefore, the minimum time to detect a collision is the time it takes for the signal to propagate from one end of cable to the other. This minimum time is called the Slot time. The number of bytes that can be transmitted in a slot time is called the Slot size. In Ethernet, the slot size is 64 bytes. Therefore for a 10Mbps Ethernet with a minimum length of 2500 meters and 4 repeaters, the minimum allowed frame must take 51.2 μsec.

Frame check Sequence:

This part of the frame contains a 32-bit (4-byte) cyclic redundancy check (CRC) code for error detection. The CRC covers the destination address, source address, length, and data fields.

Maximum Packet Size: 1526 bytes

Minimum Packet Size: 72 bytes.

Minimum Packet Spacing:

The minimum time that must elapse after one transmission before another transmission is started is 9.6μsec.

Collision Filtering:

Any received bit sequence smaller than the minimum valid packet size is discarded as a collision fragment.

2.4 Ethernet Operation

Access to a Ethernet's shared communications channel is managed with a distributed control policy specified by IEEE 802.3 standard known as 1 Persistent Carrier Sense Multiple Access with Collision Detection or 1-persistant CSMA/CD. A station wishing to transmit is said to "contend" for use of the common shared communications channel (called Ether) until it "acquires" the channel, once the channel is acquired the station uses it to transmit a packet.

To acquire the channel, station check whether the network is busy (Carrier Sense) and defer transmission of their packet until the Ether is quiet (no transmission occurring). When quiet is detected, the deferring station begins to transmit with probability one (1-persistent).

If the channel is busy, and there is a packet ready to be sent out, the MAC sublayer defers to the passing frame by delaying the transmission of its own waiting packet. After the last bit of the frame from other station has passed by, the MAC sublayer continues to defer for a certain time period called an interframe spacing. Transmission of any waiting packet is initiated at the end of this time. When the transmission is completed (or immediately at the end of the interframe spacing time if no packet is waiting in the transmission buffer), the MAC sublayer resumes monitoring the carrier-sense signal.

During transmission, the transmitting station listen for a collision. In a correctly functioning system, collision occur only within a short time interval following the start of transmission, since after this interval all stations will detect carrier and defer transmission. This time interval is called collision window or the collisions interval and is a function of the end-to-end propagation delay. If no collisions occur during this time , a transmitter has acquired the Ether and continues transmission of the packet. If a station detects collision, the transmission of the rest of the packet is immediately aborted, and a jamming signal is transmitted. To minimize repeated collision, each station involved in a collision tries to retransmit at a different time by scheduling the retransmission to take place after a random delay period. By using random delay period, the stations, which were involved in the collisions, are not likely to have another collision on the next transmission attempt.

However, to ensure that this backoff technique maintains stability, a method known as truncated binary exponential backoff is used in Ethernet.

2.4.1 Binary Exponential Backoff (BEB) Algorithm:

A station that finds it participating in a collision aborts its transmission and then goes to sleep for a random period of time, called the backoff time, before attempting to retransmit its packet. This present IEEE-802.3 standard prescribe the BEB algorithm as the method for station to determine its backoff time. The BEB algorithm is given by:

K=Min (attempts, 10)

Backoff=Random (0,2^k)

Wait for backoff number of slot times

Before proceeding to retry.

The function Random (0, 2^k) returns an integer random value uniformly distributed from 0 to 2^k-1 slot times. The variable is incremented each time the given packet is transmitted. Thus the average backoff delay doubles after each collision that a station experience on a given packet. If the packet remains undelivered when attempts reaches 16, the , the packet is discarded with an excessive collision error.

2.4.2 Ethernet Addressing

IEEE-802.3 standard allow 2-byte or 6-byte addresses, but Ethernet uses only the 6-byte addresses. The higher-order bit of the destination address is 0 for ordinary addresses and 1 for group address. Group addresses allow multiple stations to listen to a single address. When a frame is sent to a group address, all the station in the group receive it. Sending to a group of station is called multicast. The address consisting of all 1 bits is reserved for broadcast. A frame consisting of all 1s in the destination field is delivered to all station on the network.

The 46^{th} bit (adjacent to the high-order bit) is used to distinguish local and global addresses. Local addresses are assigned by each network administrator and have no significance outside the local network. Global addresses is contrast, are assigned by IEEE to ensure that no two station anywhere in world have the same global addresses. Therefore any station can uniquely address any other station by just giving the right 48-bit number.

The basic operating characteristics of the CSMA/CD protocol are outlined in the following diagram.

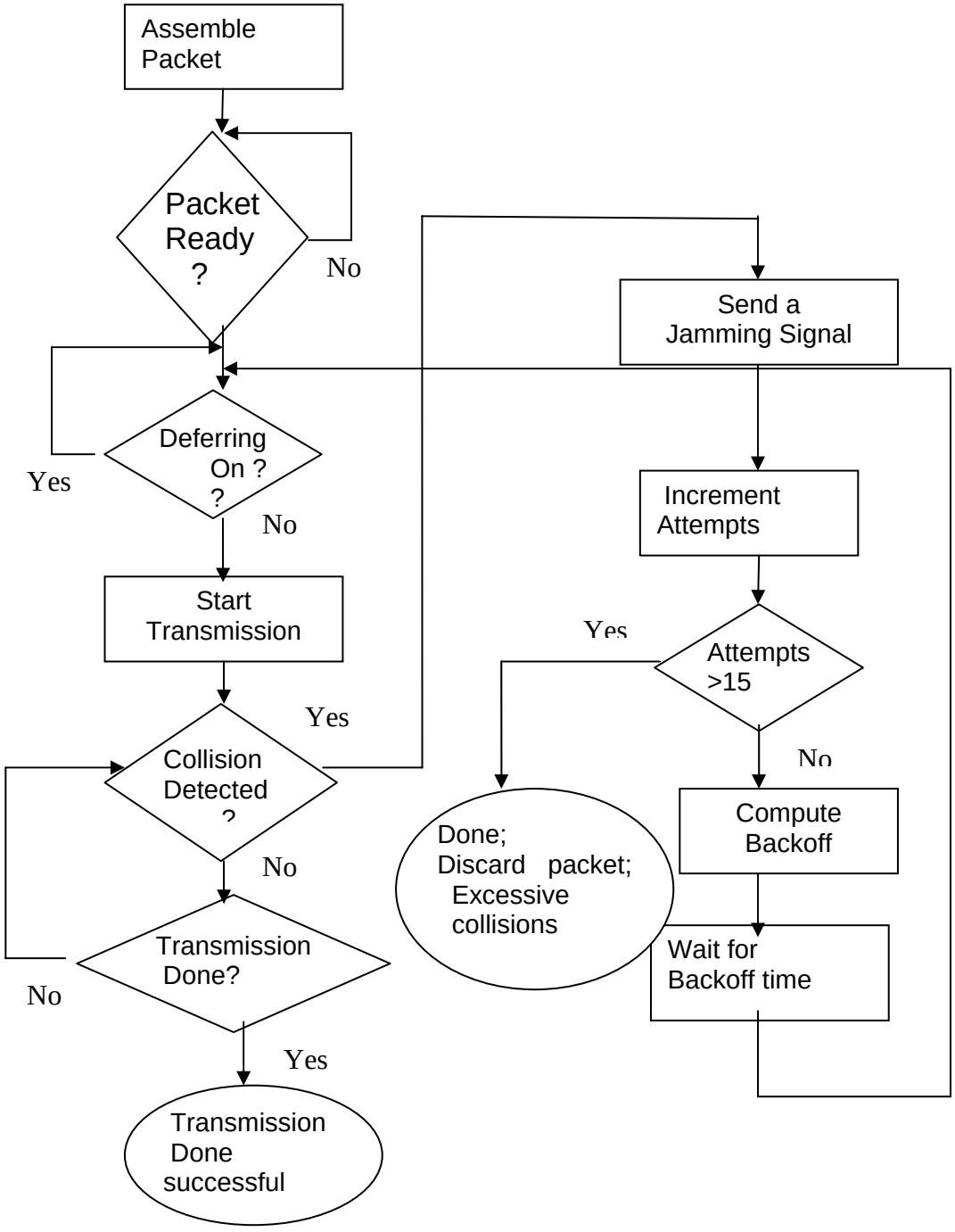

Fig: 2.3 Flow Diagram for the CSMA/CD access scheme

Capture Effect:

Ethernet channel capture is a phenomenon in which the Ethernet media access control (MAC) system can become biased for a short term toward one station on a heavily loaded network. Under certain circumstances, this allows a station to more frequently win the contention for the channel, or “capture” the channel, while that station has something to send.

The time required to acquire the channel to send the next packet in the transmit queue resembles a “reverse lottery”: most packets get sent as soon as they reach the head of the transmit queue, but a few suffer spectacularly large delays.

Under BEB, the updates to the collision counter at each host are done independently and only in response to actual transmission attempts by the given host. Thus, in particular, only the “winner” gets to reset its collision counter after a successful packet transmission.

This asymmetry in the treatment of the collision counters can permit a single busy host to “capture” the network for an extended period of time , in the following way. If we examine the system during a contention interval, when several action hosts are competing for control of the channel, we would expect each of them to possess a non-zero collision counter. Eventually, one of those hosts will acquire the channel and deliver its packet. At the next end-of-carrier event, the remaining host will still have non-zero collision values, but the “winning” host will reset its collision counter to zero before returning to competition. If the “winning” host has more packets in its transmit queue (and its network interface is fast enough), it is free to transmit its next packet immediately. Conversely, the rest of the hosts may be delayed until their latest backoff interval expires. Furthermore, should any of them collide with the “winning” host, Observe that the “winner” randomize their next retry over a (much) larger interval. Thus, the same host is likely to “win” a second time, in which case the same situation will be repeated at the next end-of-carrier event except the other hosts collision counters have gotten larger. Thus, it is even more likely that the winner’s “run” of good luck will continue until its transmit queue is emptied, or some other especially-unlucky host’s collision counter “wrap around” after 16 failed attempts---causing it to compete more aggressively for control of the channel after reporting an excessive collision error.

Although the Ethernet capture effects can cause significant short- term unfairness (in terms of the channel access delays experienced by different hosts), one not forgets the fact that it can also help performance under some circumstances. In particular, capture increases the capacity of the network by allowing a host to spread the “cost” of acquiring the channel during an Ethernet MAC layer contention period over multiple packet transmissions.

2.5 Transmission Media:

An important part of designing and installing an Ethernet is selecting the appropriate Ethernet medium. There are four major types of media in use today. Thickwire for 10Base5 (10Mbps, uses baseband signaling and can support segments of up to 500 meters) networks, thin coax for 10 Base2 networks and fiber optic for 10BaseFL networks. Thickwire was one of the first cabling systems used in Ethernet, but was expensive and difficult to use one of first cabling systems used in Ethernet, but it excellent noise immunity and is the method of choice when running between building or widely separated hubs.

2.6 FAST ETHERNET

Fast Ethernet is the generic term for the Ethernet network technology that runs at 100Mbps over either UTP or fiber-optic cable and is the successor to 10Mbps (10BaseT). It works on the basis of IEEE-802.3u standard approved in June 1995. 802.3u is not a new standard, but an addendum to the existing 802.3 standard. The Ethernet networks uses the same CSMA/CD protocol that normal 10Mbps Ethernet uses.

The maximum cable length permitted in Ethernet is 2.5km (with a maximum of four repeaters on any path). As the bit rate increases, the sender transmits the frame faster. As a result, if the frame sizes and cable lengths are maintained, then a station may transmit a frame too fast and not detect a collision at other end of the cable. So one of the two things has to be done:

(i) Keep the maximum cable length and increase the slot time (OR)

(ii) Keep the slot time same and decrease the maximum cable length(OR)

(iii) Both.

In the Fast Ethernet, the maximum cable length is reduced to only 200 meters, leaving the minimum frame size and slot time intact.

In particular, the parameter table for 100Mbps Fast Ethernet was changed as follows:

Parameters	**Values**
Slot time	512 bit times
Inter Frame Gap	0.96μs
Attempt Limit	16
Backoff Limit	10
Jam Size	32 bit
Max Frame Size	1518 bytes
Min Frame Size	512 bits
Burst Limit	Not applicable

Technically, it would have been possible to copy 10Base-5 or 10Base-2 and still detect collision on time by just reducing the maximum cable length by a factor of 10. However, the advantage of 10Base-t wiring so overwhelming, that Fast Ethernet is based entirely on this design. Thus all Fast Ethernet systems use hubs. Beside the operating rate the main difference concern type of network cabling, the data encoding method and the network spans.

2.6.1 Media Independent Interface of Fast Ethernet:

The various layers of Fast Ethernet protocol architecture are shown here in the following figure:

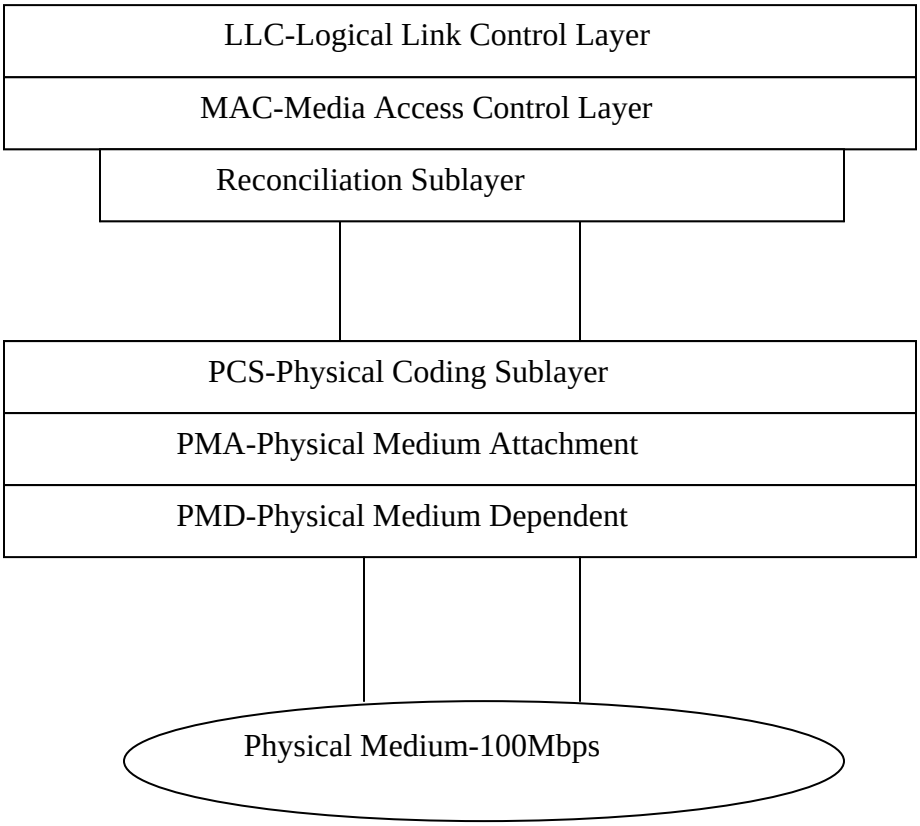

Fig.2.4 Fast Ethernet Protocol Architecture.

The Media Independent Interface (MII) is the interface between the MAC layer and the Physical layer. It allows any physical layer to be used with MAC layer. MII can accommodate two specific data rates of 10 and 100 Mbps, thereby permitting the support of 10-Base-T nodes at Fast Ethernet hubs. To reconcile the MII signal with MAC signal , the reconciliation sublayer is used .That is it provides the mapping between the 1-bit data stream at the MAC interface and the nibblewide transmit and receive interface at the MII.

The physical layer is structured in three sublayers:

Physical Coding Medium Attachment – provide a medium-independent means for the PCS to support various bit-oriented physical media.

Physical Medium Dependent – responsible for transmission and reception and reception of the electrical signal.

2.6.2 Physical Media for 100BASE-T

The four physical medium alternative for Fast Ethernet are shown here in the following figure

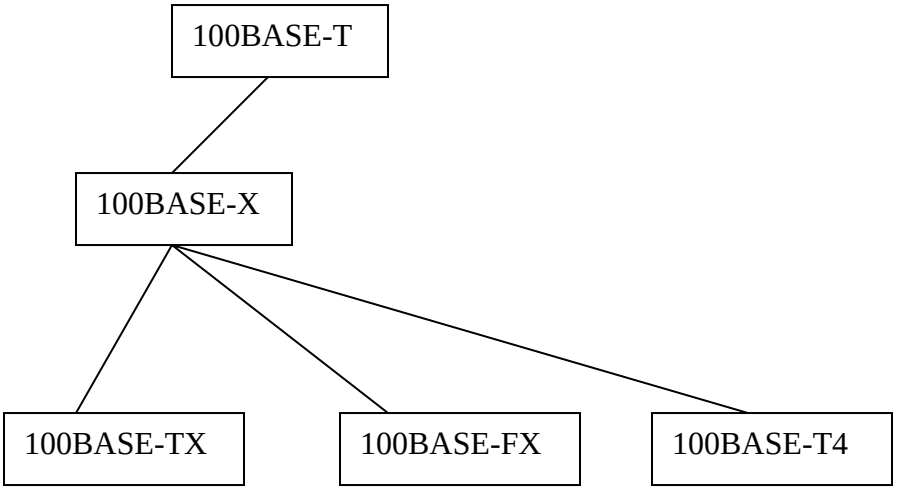

Fig 2.5. Media option for the IEEE-802.3u Standard

The various 100BASE-X options all use two physical links between devices. One link is used for transmission and other is used for reception and collision detection. 100BASE-X uses either two pairs of shielded twisted-pair(STP) wire or two pairs of high quality Cat-5 UTP cable, the latter being the most widely used in practice. The DTE-to-repeater distance is limited to 100 m. The 100BASE –FX option is similar to 100BASE-TX, except that it uses two multimode optical fibers. With these fibers transmission distances of up to 2 km are possible.

2.7 Conclusion:

Ethernet is the word's most pervasive LAN technology. Ethernet is intended primarily for use in such areas as office automation, distributed data processing, terminal access, and other situation requiring economical connections to local communication medium carrying bursts of traffic at high peak data rates. Now they are the information infrastructure of entire organizations, serving a missile-critical role and recognized as strategic business assets.

The arrival of Fast Ethernet established Ethernet as a scalable technology. Fast Ethernet is being deployed to the desktops and servers. Today most network traffic leaves its locality and has to cross a backbone link to get where it is going. A hundred Ethernet or Fast Ethernet users can easily generate a gigabit of backbone of traffic. To accommodate this traffic Gigabit Ethernet has to come.

Chapter-3

Gigabit Ethernet

3.1 Introduction

Gigabit Ethernet is an extension of the 10Mbps and 100Mbps IEEE-802.3. Ethernet standards that increase the transmission speed to 1000Mbps, ten times that of Fast Ethernet. Gigabit Ethernet standards are fully compatible with existing Ethernet installations. Since the Gigabit Ethernet specifications are based on the current Ethernet technology and keeping the layer above the MAC intact, managing and maintaining the Gigabit network should be easier than that with totally a new technology. The topology of the network that Gigabit Ethernet supports is star.

This Chapter gives an overview of Gigabit Ethernet. In this chapter importance is gives an overview of Gigabit Ethernet. In this chapter importance is given to half-duplex Gigabit Ethernet. The next section deals with the MAC Layer of Gigabit Ethernet. Section 3.3 describes Carrier Extension technique used in Gigabit Ethernet. The chapter ends up with a conclusion in Section 3.6.

3.2. THE MAC Layer of IEEE-802.3z:

Gigabit Ethernet works on the basis of IEEE-802.3z standard retains Carrier Sense Multiple Access/Collision Detection (CSMA/CD) protocol as the access method.

The MAC layer of Gigabit Ethernet is similar to that of 10Mbps Ethernet and 100Mbps Fast Ethernet. Scaling Ethernet to 1000Mbps creates some problem. Gigabit Ethernet maintains the minimum and maximum frame sizes of Ethernet. Since Gigabit Ethernet maintains the minimum and maximum frame sizes of Ethernet. Since Gigabit Ethernet is 10 times faster than Fast Ethernet, to maintain the same slot size, the maximum cable length would have to about 20 meters, which would make for a very unpractical topology. Instead, Gigabit Ethernet uses a bigger slot size of 512 bytes.

3.2.1 Transmission Media

The physical layer alternatives fall under realm of IEEE 802.3z, which specifies using 8B 10B encoding on the line include the following **:**

(i) 1000BASE-LX supports optical fiber backbone using wavelength in the 1310nm window (1270 to 1355 nm). For short-span backbone the maximum lengths are 550 m using multimode fibers with either 50 or 62.5-µm core diameters. Long-span campus-type backbone can have spans up to 5 km with single mode fibers.

(ii) 1000BASE-SX is a lower-cost short-wavelength (770-to 860-nm range) option used in short span backbones. The link length can be up to 275 m with a maximum length of 25 m.

(iii) 1000BASE-CX supports short 1-Gbps spans between devices clustered within a single room or an equipment rack. The cables are specialized shielded twisted-pair (STP) copper wire with maximum length of 25m.

3.2.2 Gigabit Ethernet Encoding Schemes

The 8B10B coding method arose from Fiber Channel and other fiber oriented protocol. In this scheme an 8-bit group of data from MAC is converted into a larger group of 10 signals .The figure shown below illustrates the operation of this code. The encoding is actually accomplished as a combination of two separate 5B6B and 3B4B block codes. However, these two codes are not necessarily performed as separate functions since they generally serve as a tool that simplifies the mapping and implementation functions. Before encoding, the bits are given designations A,B,C,D,E,F,G,H. After encoding, the bits are designated by the letters a through j.

The 8B10B code has a good transition density for easier clock recovery; it is well DC-balanced to prevent baseline wander and provide a good error detection capability. As part of encoding process, the control line input K indicates whether the bit A through H is data or control bit s. If they are control bits, a special 10-bit block is generated. There are 12 of these unique non-data blocks, which can be used for synchronization, signal control instructions, or other functions. The disparity control function monitor the number of 1 and 0 bits, where the disparity is the excess of one type of bit over another. The disparity

controller determines the proper 8B10B encoding of next data byte so that an equal balance of 1 nad 0 bits is maintained.

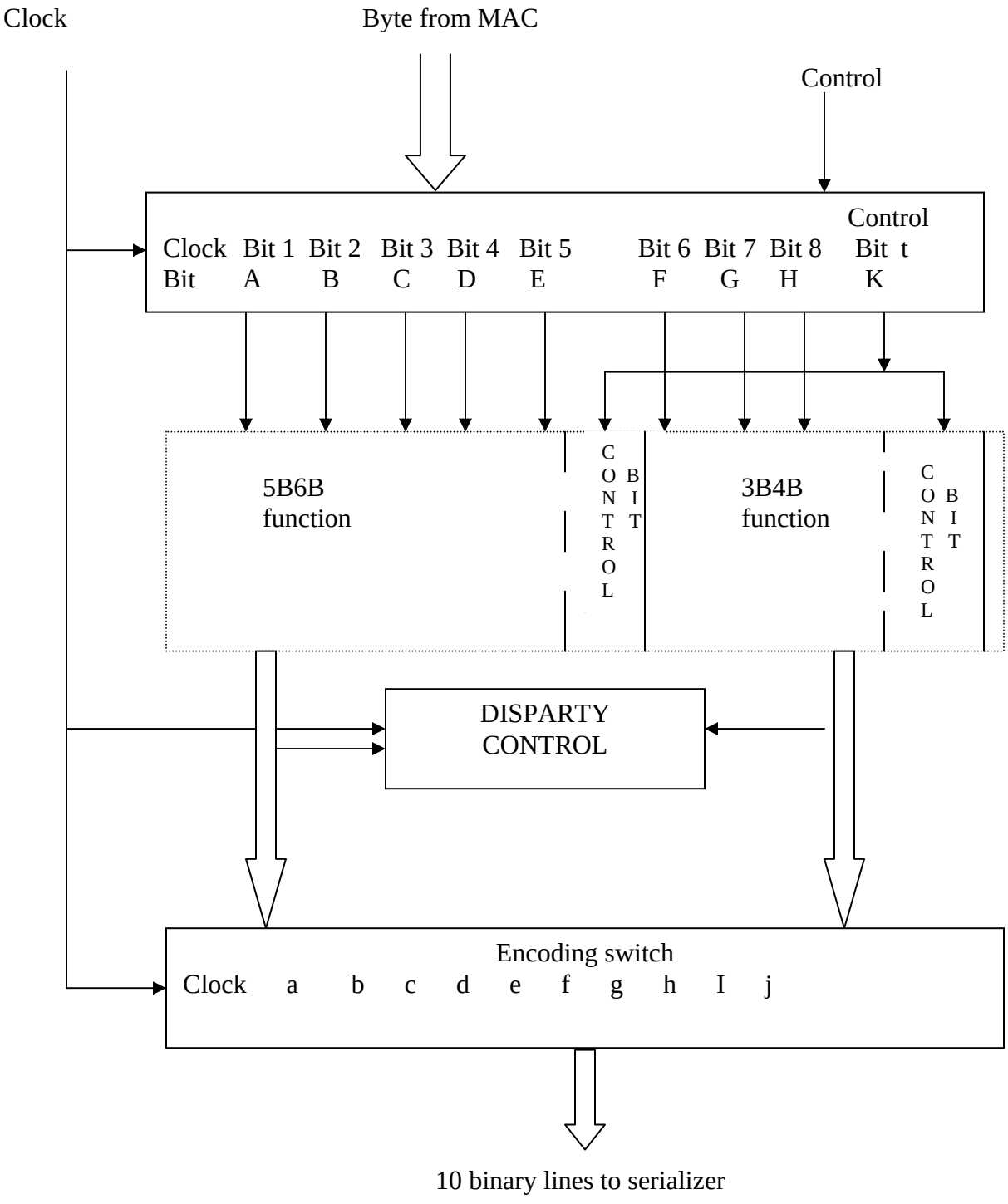

Fig. Operation of 8B 10B –encoding

3.3 Carrier Extension:

To maintain compatibility with Ethernet, the minimum frame size of Gigabit Ethernet is not increased, but the “carrier event” is extended. If the frame size is less than 512 bytes, then the frame is padded with extension symbols. These are special symbols, which cannot occur in the payload. This process is called carrier extension. Carrier extension is a way of maintaining 802.3 minimum and maximum frame size with meaningful cabling distances.

For carrier extended frames, the non-data extension frames, the non-data extension symbols are included in the collision and dropped. However, the Frame Check Sequence (FCS) is calculated only on the original (without extension symbols) frame. The extension symbols are removed before the receiver checks the FCS. So the LLC (Logical Link Control) layer is not even aware of the carrier extension. The following figure shows the Ethernet frame format when carrier extension is used.

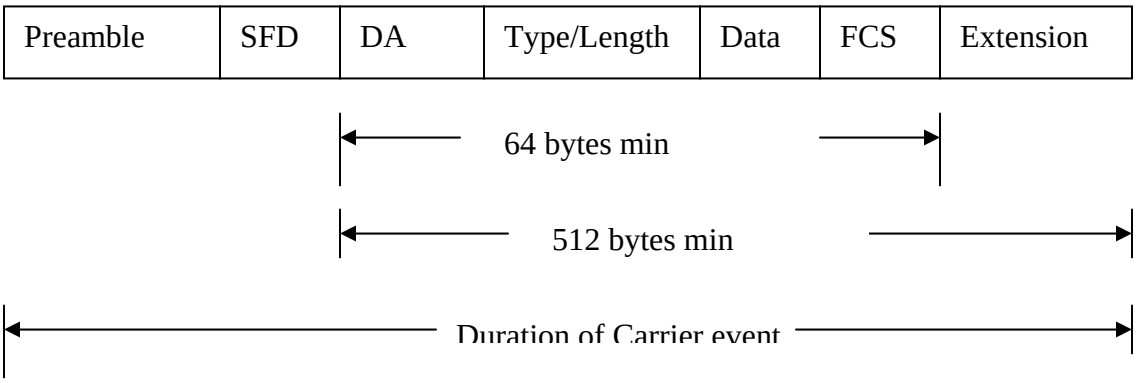

Fig3.1 Gigabit Ethernet Frame format

To accommodate the new frame, a number of modifications were made to the MAC. In particular, the parameter table was changed as follows:

Parameters	Values
Slot time	4096 bit times
Inter Frame Gap	96ns
Attempt Limit	16
Backoff Limit	10
Jam Size	32 bit
Max Frame Size	1518 bytes
Address Size	48 bits
Extended Size	448 bytes

Table3.1: MAC parameters

Carrier Extension has some undesirable side effects.

3.3.1 Implication of Carrier Extension:

- End of packet may precede end of carrier.

  If the packet size is less than 512 bytes, then it will be padded with extension symbols to make carrier 512 bytes and then transmitted. On the receiving side the end of this packet precedes the end of carrier. But the transmission is said to be successful only if whole carrier event of 512 bytes is received.

- Reduced network efficiency:

  Consider the following

Station A transmits short packet with carrier extension. Station B (at maximum distance from station A) collides with A. Now the fragment received by station C (distance (C,A) is much less than distance(A,B)) may contain the entire packet but must still be discarded to avoid duplication of data when A retransmits.

Also carrier extension on small packets increases overhead. Therefore carrier extension reduced network efficiency when transmitting small packets and when collision occurs.

3.4 Packet Bursting:

Most frames in the network are small and the carrier extension adds significant overhead and result in an effective throughput much less than 1000 Mbps. This throughput problem was addressed by feature called Packet Bursting is an extension of carrier extension.

Packet Bursting is nothing but a “Carrier Extension + Burst of packets”. After padding a frame with carrier extension symbols, a station does not relinquish control, instead it transmits whatever remaining packet it has, one after the other, filling the interframe gap (IPG) with carrier extension symbols, such that the line never appears free to any other station. This continues until a Packet Bursting timer of 1500 bytes expires. In the worst possible scenario a station may hold control for two 1500-byte packets. Packet Bursting substantially increases the throughput. The following figure shows how packet Bursting works.

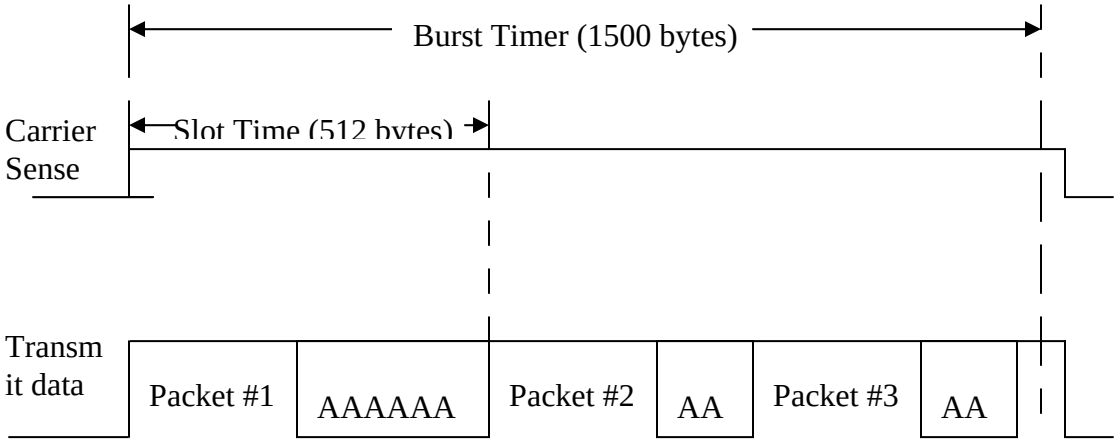

Fig 3.2 Packet Bursting

3.5 Gigabit Ethernet Architecture

As we already mentioned that IEEE 802.3z retains the CSMA/CD protocol from Ethernet. It supports half-duplex as well as full-duplex mode of operation. For the Physical layer, 802.3z uses the physical signaling used in Fiber channel to support gigabit rates over optical fibers. As it stand today, 802.3z define transmissions via single-mode and multimode fiber (long wave and short wave) , short haul twinax cabled (balanced two-pair shielded coax). Gigabit Ethernet supports shared as well as switched architecture.

Gigabit Ethernet technology brings Fast Ethernet and Fiber channel as follows:

From Fast Ethernet

1. Maintains the same technology alternatives and constraints.
2. Medium Access Control (CSMA/CD and Full-duplex).
3. Frame structure.
4. Minimum frame size of 64 bytes.
5. Maximum frame size of 1518 bytes.
6. Management Independent Interface.

From Fiber Channel

1. 8b/10B coding (PCS Layer)
2. Serializer/Deserializer (PMA layer)
3. Physical Medium Dependent (PMD)
4. Medium Dependent Interfaces.
5. Physical Media and Signaling.

3.5.1 Protocol Architecture:

Two technologies were merged to create the Gigabit Ethernet standard: IEEE 802.3 Ethernet and ANSI Fiber Channel. The following figure shows the components that were brought together to form the new Gigabit Ethernet technology.

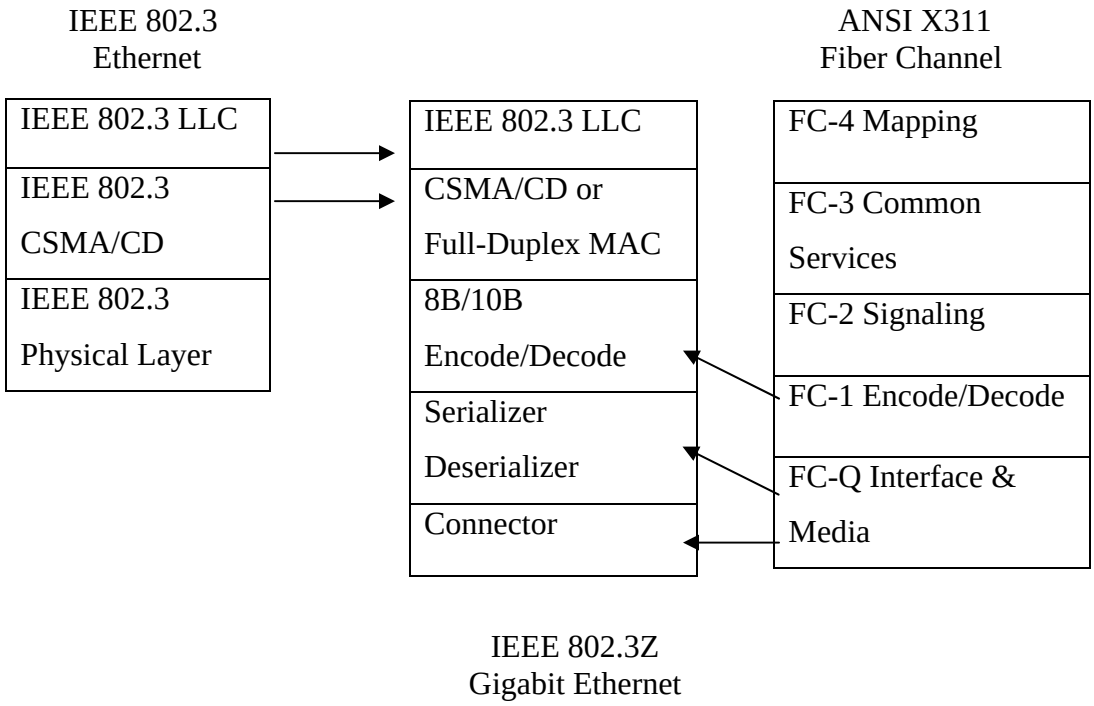

Fig3.3 Gigabit Ethernet Protocol Stack

3.6 Conclusion:

By bringing Fast Ethernet and Fiber channel technologies together, Gigabit Ethernet takes advantage of the high-speed physical interface technology of Fiber Channel while maintaining the IEEE 802.3 Ethernet frame format. This ensures backward compatibility for installed media, use of full or full-duplex transmissions, and at the end this guarantees interoperability with existing Ethernet and Fast Ethernet applications.

Now Gigabit Ethernet is being deployed as a backbone in existing networks. It can be used to aggregate traffic between clients and “server level”, and for connecting Fast Ethernet switches. It can also be used for connecting workstations and servers for high-bandwidth applications such as medical imaging or CAD.

Chapter-4

Simulation and Gigabit Ethernet Simulator

4.1 Introduction

In order to study a system to understand the relationship between its components or to predict how the system will operate under a new policy, it is sometimes possible to experiment with the system itself. However, this is not always possible. A new system may not yet exist or even if the system exists, it may be impractical to experiment with it. For example, it may not be possible to keep the length of all the packets in a big Ethernet LAN to be fixed and analyze the effect of it on the throughput of the network. Studies of such type of systems are accomplished through simulation.

A simulation is the imitation of the operation of a real-world process or system over time. In this chapter we discuss some of the basic concepts of simulation and the implementation details of the simulator.

4.2 Components of a System

In order to understand and analyze a system, a number of terms need to be defined.

System: A collection of entities that interact together over time to accomplish one or more goals.

System State: A collection of variables th**at** contain all the information necessary to describe the system at any time.

Event: A simultaneous occurrence that changes the state of a system.

Event notice: A record of an event to occur at the current or some future time, along with any associated data necessary to execute the event.

Event list: A list of event notices for future events, ordered by time of occurrence; also Known as the future event list (FEL).

Clock: A variable representing simulation time.

4.3 Discrete Event Simulation

A discrete-event simulation is the modeling over time of a system all of whose state changes occur at discrete points in time – those points when an event occurs. A discrete-event simulation proceeds by producing a sequence of systems that represent the evolution of the system through time. A given snapshot at a given time t include not only the system state at time t, but also a event list of all activities currently in progress and when each such activity will end, the status of all entities and current membership of all sets, plus the current values of cumulative statistics and counters that will be used to calculate summary statistics and counter that will be used to calculate summary statistics
at the end of simulation.

4.4 The Event Scheduling/Time Advance Algorithm

The mechanism for advancing simulation time and guaranteeing that all events occur in correct chronological order is based on the future event list (FEL). This list contains all event notices for events that have been scheduled to occur at future time. Scheduling a future event means that at the instant an activity begins, its duration is computed or drawn as a sample from a statistical distribution and the end-activity event, together with its event time, is placed on the future event list. In the real world, most future events are not scheduled but merely happen- such as random breakdown or random arrivals. In the model, the end of some activity, which in turn is represented by a statistical distribution, represents such random events.

At any given time t, the FEL contains all previously scheduled future events and their associated event times. The FEL is ordered by event time, meaning that the events are arranged chronologically; that is the event times satisfy

$t<t_1<=t_2<=t_3<=\ldots<=tn$

Time t is the value of CLOCK, the current value of simulated time. The event associated with time t_1 is called the imminent event; that is, it is the next event that will occur. After the system snapshot at simulation time CLOCK=t has been updated the CLOCK is advanced to simulation time CLOCK=t_1 and the imminent event notice is

removed from the FEL and the event executed. Execution of the imminent event means that a new system snapshot for time t_1 is created based on the old snapshot at time t and the nature of the imminent event. At time t_1, new future events may or may not be generated, but if any are scheduled by created event notices and putting them in their position on the FEL. After the new system snapshot for time t_1 has been updated, the clock is advanced to the time of the new imminent event and that event is executed. This process repeats until the simulator must perform to advance the clock and build a new system snapshot is called the event-scheduling/time-advance algorithm.

4.4.1 Algorithm steps:

Step1. Remove the event notice for the imminent event from the FEL.

Step2. Advance CLOCK to imminent event time (i.e., advance CLOCK from t to t_1).

Step3. Execute imminent event, update system state, change entity attribute, and set membership as needed.

Step4. Generate future events and place their event notices on FEL ranked by event time.

Step5. Update cumulative statistics and counters.

4.5 Performance Measures

There are many measures to evaluate the performance of a network. We have used some of the important performance measures like throughput, average_delay, Collision/Packet and Collision Rate. A description of these above quantities is as follows:

Throughput:

The throughput of a local network is a measure in bits per second of the successful traffic being transmitted between stations; that is since packets can become corrupted in traveling station to station, it is customary to only count the error-free bits (error free packets) when measuring throughput.

Average delay:

The average delay experienced by packets transmitted in the experiment. This quantity is calculated by dividing the total delay for the experiment by the number of successful packet transmissions.

Collision per Packet:

The average number of collisions experienced by packets before successful transmission.

Collision Rate:

The overall collision rate in collisions per second.

4.6 Gigabit Ethernet Simulator

The Gigabit Ethernet Simulator is a single process discrete event simulator. The program simulates a finite population network. This gives a better picture of the behavior of the network than simulations based on assumptions similar to those used in infinite-population modeling studies. The program models packet buffering at each individual station with a buffer length of one. First, we simulated Fast, we simulated Fast Ethernet, then we increased the speed to 1000Mbps(1Gbps), the slot time to 512 bytes and implemented Carrier Extension to get “Gigabit Ethernet without Packet bursting” we got the final half-duplex Gigabit Ethernet simulator.

4.7 Important Data Structures:

Some of the important data structures used in the simulation are given here.

Description of s host and the statistics kept for that host.

```
typedef struct
{
        long status;//Indication of host status
        long hostid;//This is used to reconize host
        double time;//Time of next packet arrival
```

```
} host_desc;
```

Statistics collected for the experiment as a whole.

```
typedef struct
{
        long thruput;
        double delay;// Total delay experienced*/
        long  success;//Successful transmission*/
        long collision;//Total overall collision experienced*/
        long discards;
} expt_stats ;
```

Description of a simulated network event.

```
struct event
{
        struct event *ptr;
        double time; //Time at which struct event occur
        short hostid;//Location of the host ehich generated the struct event
        short size;//Length of the packet
        short retries;//Attempt before success
};
```

4.8 Input Parameters

The main parameters for the simulator program are:

1. Number of hosts.
2. Packet distribution in terms of four different packet sizes and their percentage in total load generated.

3. Offered load for the network. The packets are generated using a uniform distribution according to the offered load.

4.9 EVENT QUEUE:

A single event queue is used for the simulation, which at any given time contains one entry for each station on the network. The entries in the event queue are event structures. Entries in the queue are arranged in the ascending order with respect to the simulation time of the next transmission attempt by the station. Therefore the head of the queue contains an event whose next transmission attempts time is less than all the other events in the queue.

4.10 Main Routines of the simulator

1. init_run
    (i) Calculates total number of packets to be generated based on offered load, packet sizes & their percentage in total load.
    (ii) Generates first batch of packet arrivals.
    (iii) Initializes host descriptions, queues and overall experiment statistics.
2. addPackets
    (i) Generates a batch of packets with different packet sizes based on their percentage in total load. Generate packet arrival time based on the uniform distribution starting from the last arrival time in the previous batch of packets. Adds packets to the respective queues based on the *hostid* of generated packet.
    (ii) Remove one packet each from all host queues and add to event queue, so that event queue has only one packet for each host.
3. en_send
    (i) Send a packet. First, see if the packet caused a collision. Any enqueued transition, which begins with one propagation delay of the beginning of

this packet, will collide. If collision is detected then call en_collision, else the transmission is successful. Call en_successful if the network is Fast Ethernet, en_successful_g if the network is Gigabit Ethernet without packet bursting, en_successful_g_pb if the network is Gigabit Ethernet with packet bursting.

(ii) If it the last arrival in the current batch of packets, then generate a new batch of packets.

(iii) Scan event queue for arrivals and backoffs with in current transmission and move them into boundary. Call en_defer.

4. en_defer

(i) As long as the arrival times are less that the end time of current transmission, deferral mechanism means these will wait until the current transmission finishes. Their arrival time is updated, and they are moved to the appropriate position in the queue.

5. en_successful

(i) In case of successful transmission, size of packet will be used to calculate the throughput. Update total throughput.

(ii) Remove one more packet from the host queue for which the transmission is successful, and add to the event queue.

(iii) If the arrival time of this packet is less than current time, update arrival time to current time. Update total delay for the experiment.

6. en_successful_g

(i) In case of successful transmission, size of packet will be used to calculate the throughput. Update total throughput. If the packet size is less than 512 bytes then the current time is calculated based on the time taken to transmit 512 bytes.

(ii) Remove one more packet from the host queue for which the transmission is successful, and add to the event queue.

(iii) If the arrival time of this packet is less than current time, update arrival time to current time. Update total delay for the experiment.

7. en_successful_g_pb

   (i) In case of successful transmission, size of packet will be used to calculate the throughput. Update total throughput. If the packet size is less than 512 bytes then the current time is calculated based on the time taken to transmit 512 bytes (career extension bytes).

   (ii) If there are more packets in the host queue for which arrival time is less than current time and the total transmission size (including career extension bytes) is less than 1500 bytes, remove those from the host queue. Update current time and throughput based on the packet sizes of these packets.

   (iii) Remove one more packet from the host queue for which the transmission is successful, and add to the event queue.

   (iv) If the arrival time of this packet is less than current time, update arrival time to current time. Update total delay for the experiment.

8. en_collision.

   (i) Count this collision.

   (i) Call backoff for each station involved in collision using actual end of collision at each station. The time from which the backoff is computed is the later of

   The time at which the station detects the collision plus CD (collidion detection time for the host) plus JAM (Jamming signal time)

   The last time at which the station is busy from the transmission of some other station.

   This does not take into account non-configuration collisions, where the backoff may start earlier.

   (ii) Update total delay for the experiment and move the packets to appropriate positions in the queue.

9. backoff

(i) Calculates the backoff for a station involved in a collision based on the retry count.

4.11 Conclusion

The implementation details of a half-duplex Gigabit Ethernet simulator are presented in this chapter. Through this simulator we can investigate the effects of the packet arrival process, size of packets transmitted, Carrier Extension and Packet Bursting. We can also get results on ta number of different performance metrics, such as throughput, average packet delay, collision rate, collision/packet and the actual arrival rate. Based on these metrics we can analyze the performance of Gigabit Ethernet and we can compare that with Fast Ethernet.

Chapter-5

Result and Discussions

5.1 Introduction

There are many ways to analyze the performance of a network. We have analyzed the performance considering variations of (i) throughput verses offered load; (ii) average delay versus offered load; (ii) Collision per packet (Collision/pkt) versus offered load. We have computed the results obtained for “Gigabit Ethernet with Packet Bursting” with that of “Gigabit Ethernet without Packet Bursting” and “Fast Ethernet”. This gives a clear picture about the performance of Gigabit Ethernet. Analyze is done for a bus network. Since hub is logically equivalent to bus network, analyze the made for bus works holds for hubs also.

5.2 Input for the Simulator

For the total analysis the following assumption were made:

1. The cable length is the maximum allowed cable length.
2. Distribution of the inter-arrival times is uniformly distributed.

For mixed packet size load, two types of inputs were considered for the analysis. They are as follows:

Input 1:

Packet Size Distribution

40% of the packets are 76 bytes in length, 39% are 180 bytes, 3% are 975 bytes, and 18% are 1526 bytes.

Number of hosts is 32.

Input 2:

Packet Size Distribution

40% of the packets are 72 bytes in length, 20% are 180 bytes, 22% are 975 bytes and 18% are 1526 bytes.

Number of hosts is 16.

5.3 Measured Performance

To measure the performance the metrics considered are throughput, collision/packet and average packet delay. Comparisons are made between Gigabit Ethernet with Carrier Extension and without Packet Bursting, Gigabit Ethernet with Carrier Extension and with Packet bursting, and Fast Ethernet. The following are the terminologies used to present in a simpler manner:

- By Gigabit Ethernet without PB, we mean Gigabit Ethernet with Carrier Extension only (Packet Bursting is not implemented).
- By Gigabit Ethernet with PB or by simply Gigabit Ethernet we mean Gigabit Ethernet with Carrier Extension and with Packet Bursting.

Here by three networks we mean Fast Ethernet, Gigabit Ethernet without PB and Gigabit Ethernet with PB.

We now present the results obtained from simulation for measuring the performance of Fast Ethernet, Gigabit Ethernet with Packet Bursting. In the Graphs plotted 'FE' represents Fast Ethernet, 'GE' represents Gigabit Ethernet without PB, and 'GE_PB' represents Gigabit Ethernet with PB.

1. Throughput:

(ii) Figure 5.3.1 shows the throughput as a function of offered load for Input-1 for the three networks. It can be easily seen that Gigabit Ethernet without PB achieved only throughput of over 399.3Mbps with 100 percent Offered Load, whereas Gigabit Ethernet with PB achieved a throughput of over 648Mbps compared to Fast Ethernet's 87Mbps throughput with 100 percent Offered Load. This means that the performance of Gigabit Ethernet without PB is only marginally better than Fast Ethernet. But this is due to reason that 79% of the Packets in the network are of size less than 512 bytes and the carrier Extension on each of these small packets increases the overload on the packets. After implementing Packet Bursting the throughput of Gigabit Ethernet increases to 648Mbps. This is because of the reason that for all the small packets following the first one within a burst timer the overload is almost nil.

Figure 5.3.2 shows the throughput as a function of Offered Load for the Inout-2 for the three networks. Here it can be seen that the throughput achieved by Gigabit Ethernet is 752Mbps. The main reason behind this is the number of station in the network considered for Input-2 is 16, and the number of small packets (of size less than 512 bytes) is less than that of the number of small packets in Input-1. Since the number of station is less, the number of collisions is less than and hence the throughput is more.

(ii) The following table gives the throughput achieved by Gigabit Ethernet with and without PB for two types of Packet length 256 and 512 bytes and for the packet size distribution mentioned in Input-1 for 100% offered load.

Packet Size	**Gigabit Ethernet without PB**	**Gigabit Ethernet with PB**
256 bytes	324.8Mbps	605Mbps
512 bytes	638.4Mbps	732Mbps
Distribution in Input-1	399.3Mbps	648.66Mbps
Distribution in Input-2	557.76Mbps	752.36Mbps

Table 5.3

From the above table and from Figure 5.3.3, it is clear that the use of Packet Bursting resulted in significant improvement in the throughput of small packets over Carrier Extension. The same result was obtained with packet size distribution specified in the Input-1. There is not much difference in figure 5.3.4 because at size of 512 bytes career extension overload is nil.

With Packet Bursting, up to 448 bytes “wasted space” is amortized over a large number of useful data bits. Without it, every frame below 512 bytes would need to be padded with extension bits.

2. Collision/packet:

Figure 5.3.5, Figure 5.3.6, Figure 5.3.7 and Figure 5.3.8 show Collision/pkt as a function of the offered load for fixed packet lengths of Input-1, Input-2, 256 bytes and 512 bytes respectively for the three networks. The number of stations considered in the network is 32.

It is clearly observable from the figures that the collision/pkt peaks at lower percent offered load for Gigabit Ethernet compared to Fast Ethernet then decreases due to increased number of packet discards (resulting in an increasing in capture effect).

It is also visible that for small packets, collision/pkt for Gigabit Ethernet without PB is higher. If the packet are small, then many packets can be accommodated within a burst timer, and if the packet within the burst timer is transmitted successfully, then the subsequent packets within that burst timer are guaranteed not to collide.

Collision/pkt for Input-2 is less compared to collision/pkt for Input-1 for all the three networks. The reason is that the number of station in the stations in Input-2 is less than that of the number of stations in Input-1.

3. Average a Packet Delay:

Figure 5.3.9, 5.3.10, 5.3.11 & 5.3.12 show average packet delay as a function of offered load for the three networks. Here we have considered 32 stations in the network.

The graphs show that average packet delay for Gigabit Ethernet without PB for small packets is even more than that of average packet delay for Fast Ethernet. The reason is increase in slot time to 4096 bit times.

But for Gigabit Ethernet with PB, even for small packets the average packet delay is much less than that of Fast Ethernet's.

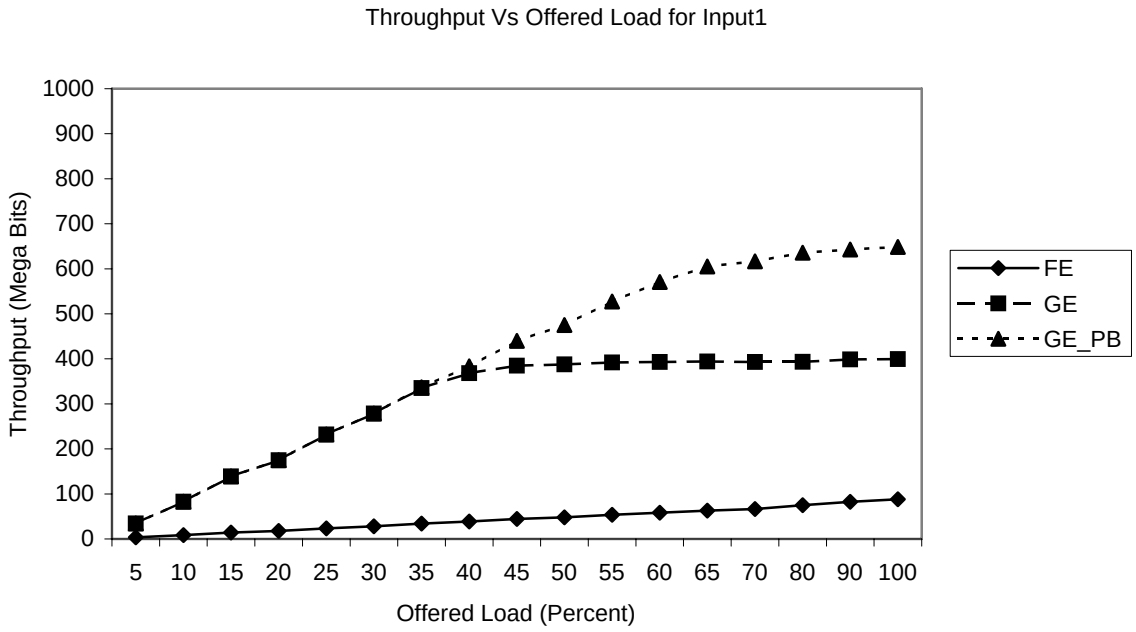

Figure 5.3.1

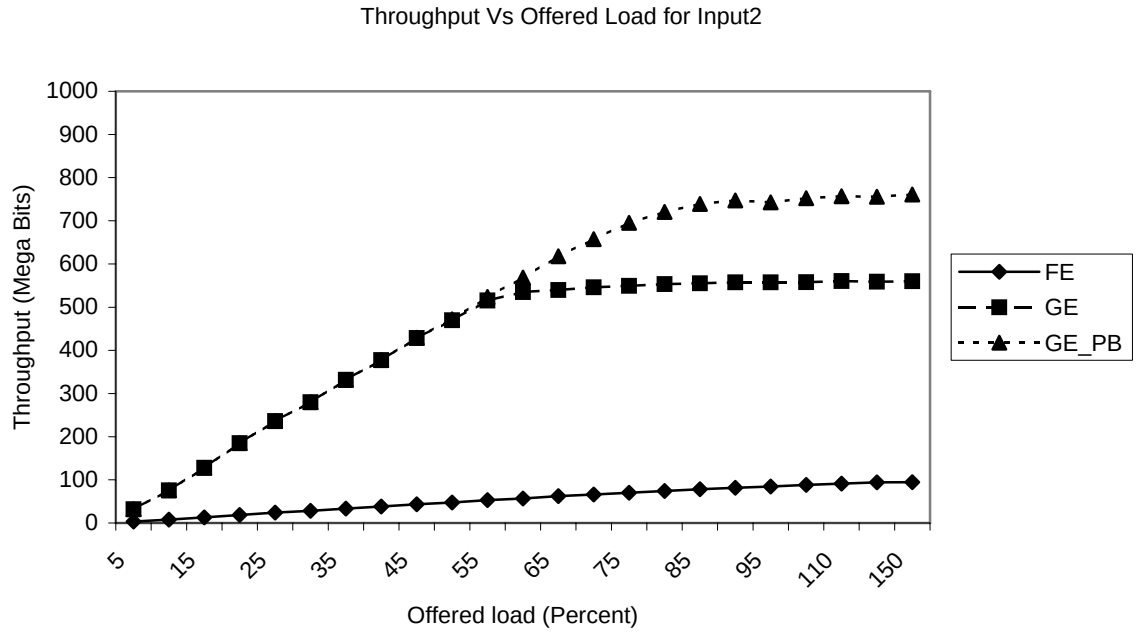

Figure 5.3.2

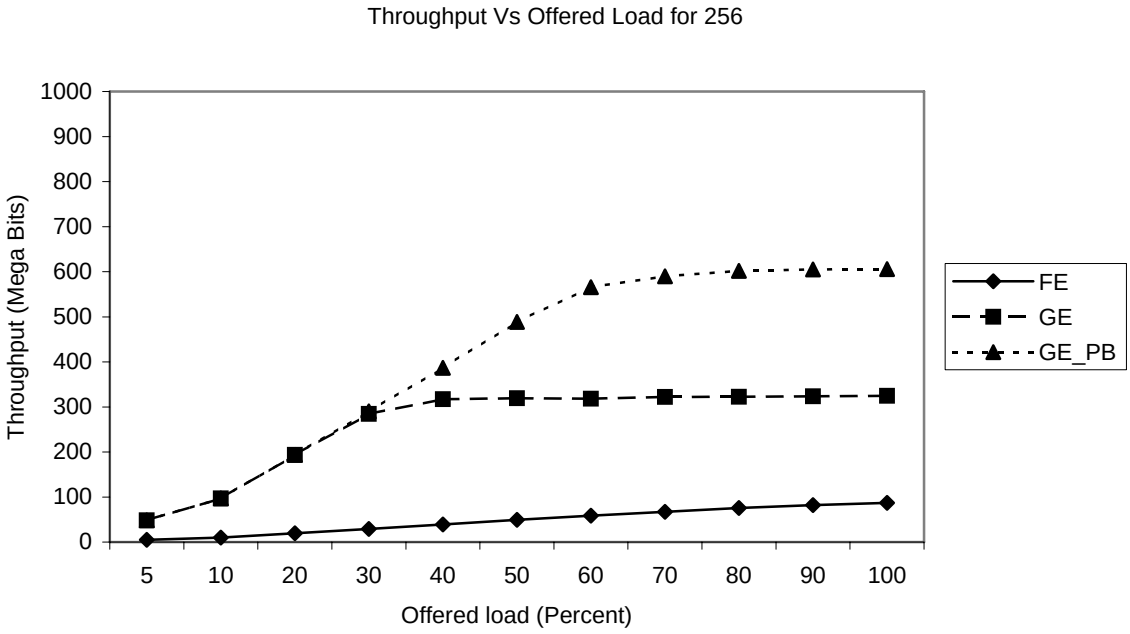

Figure 5.3.3

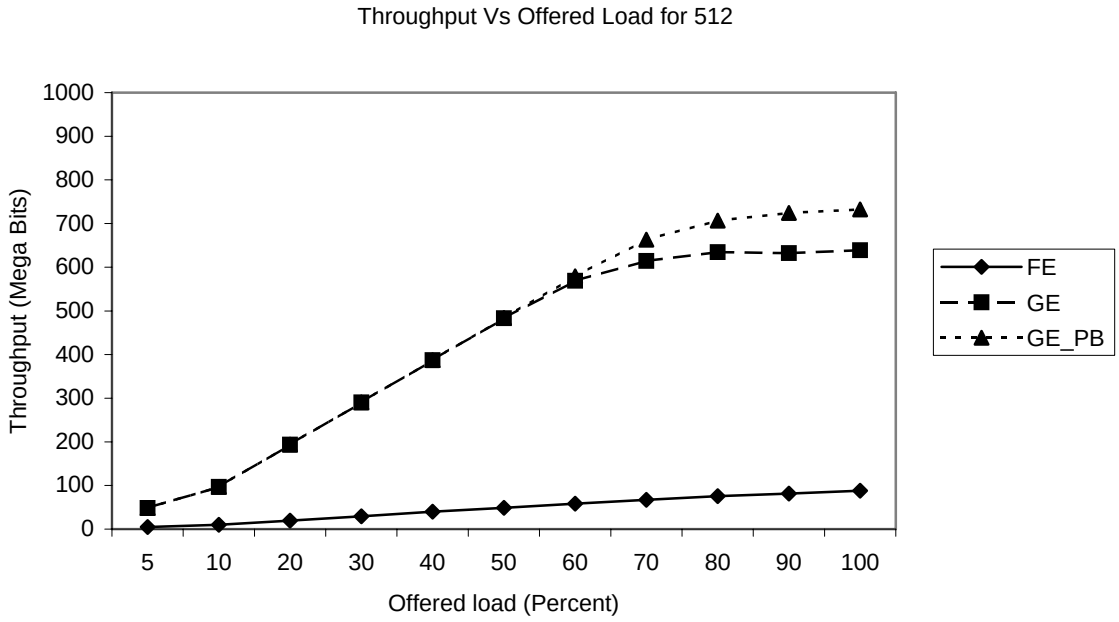

Figure 5.3.4

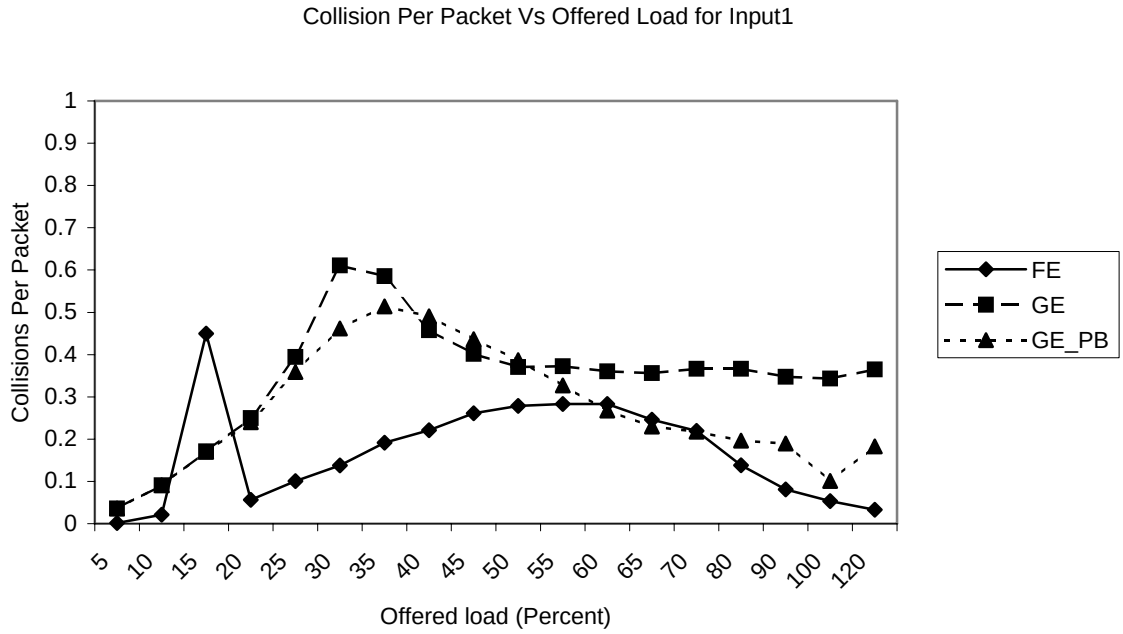

Figure 5.3.5

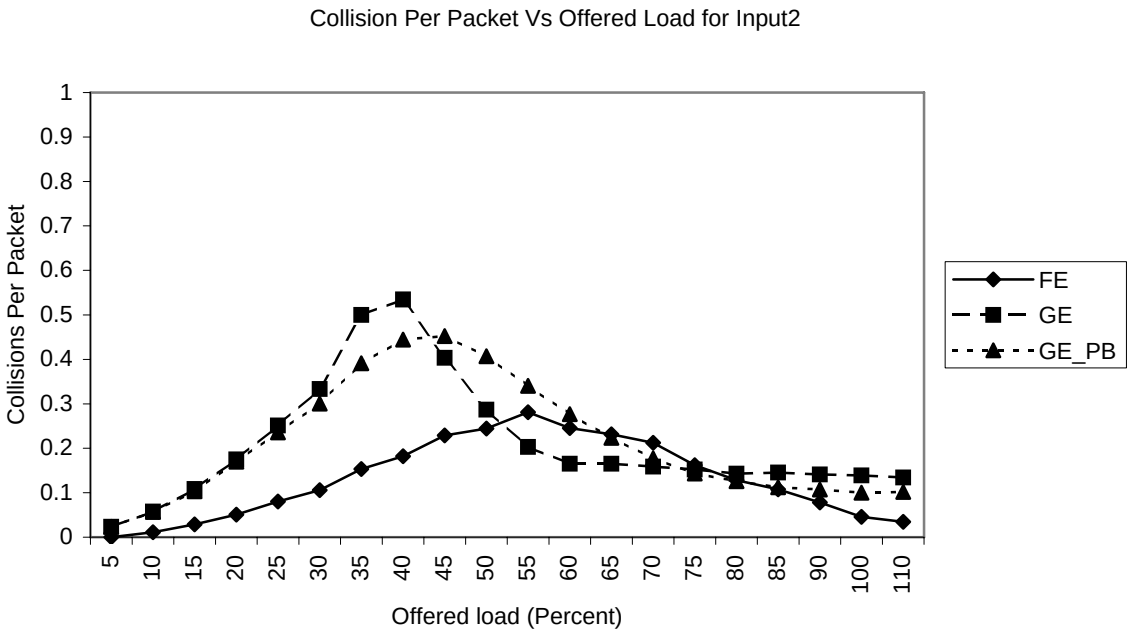

Figure 5.3.6

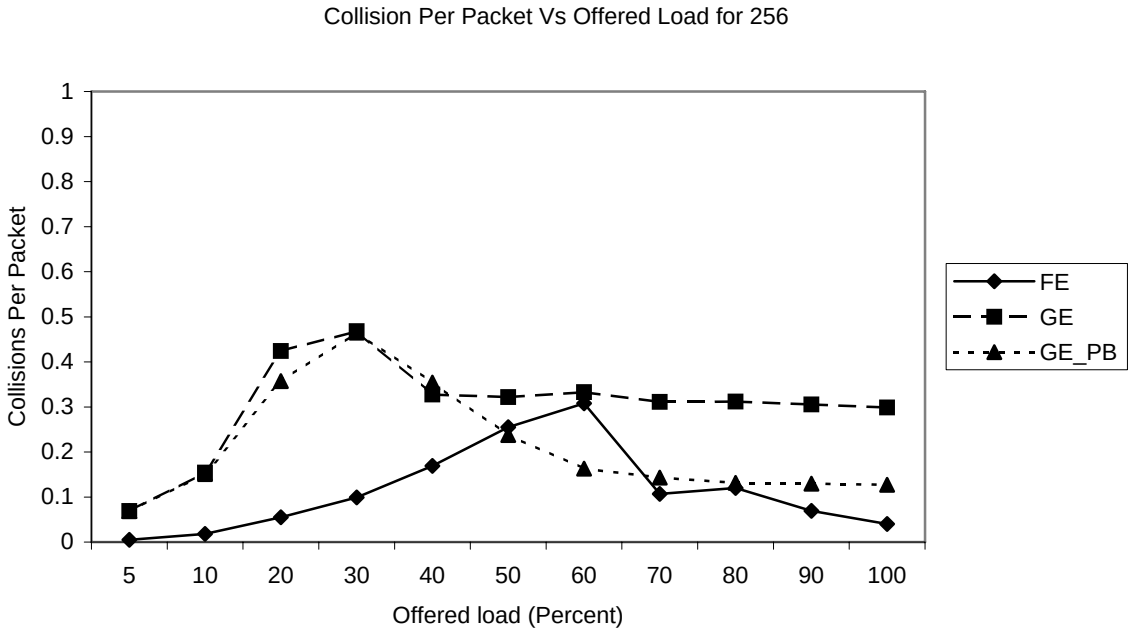

Figure 5.3.7

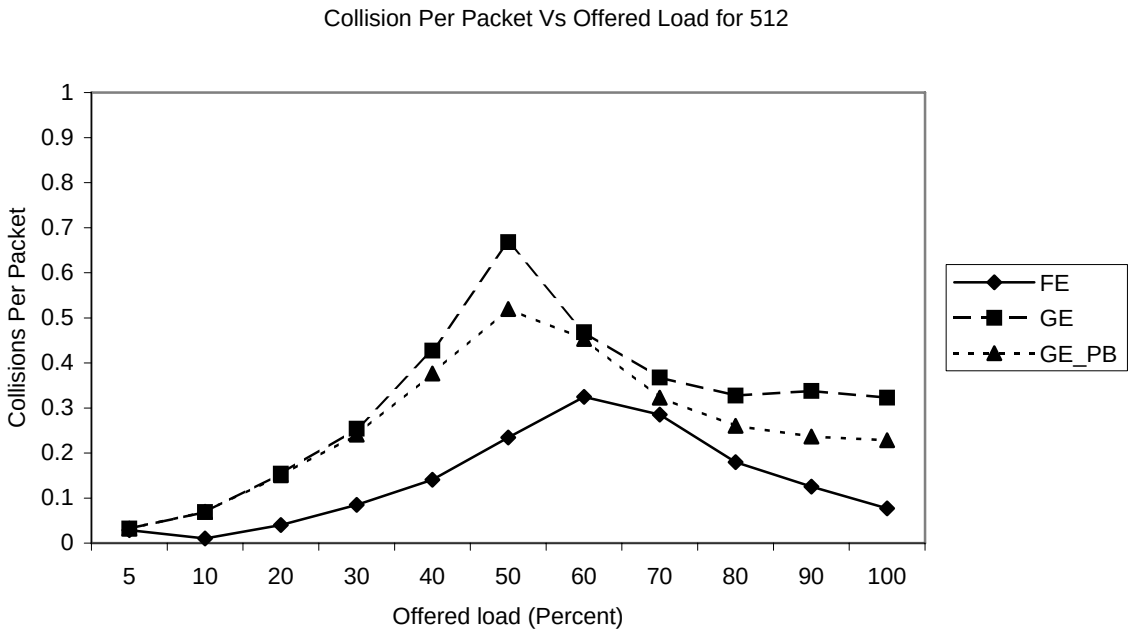

Figure 5.3.8

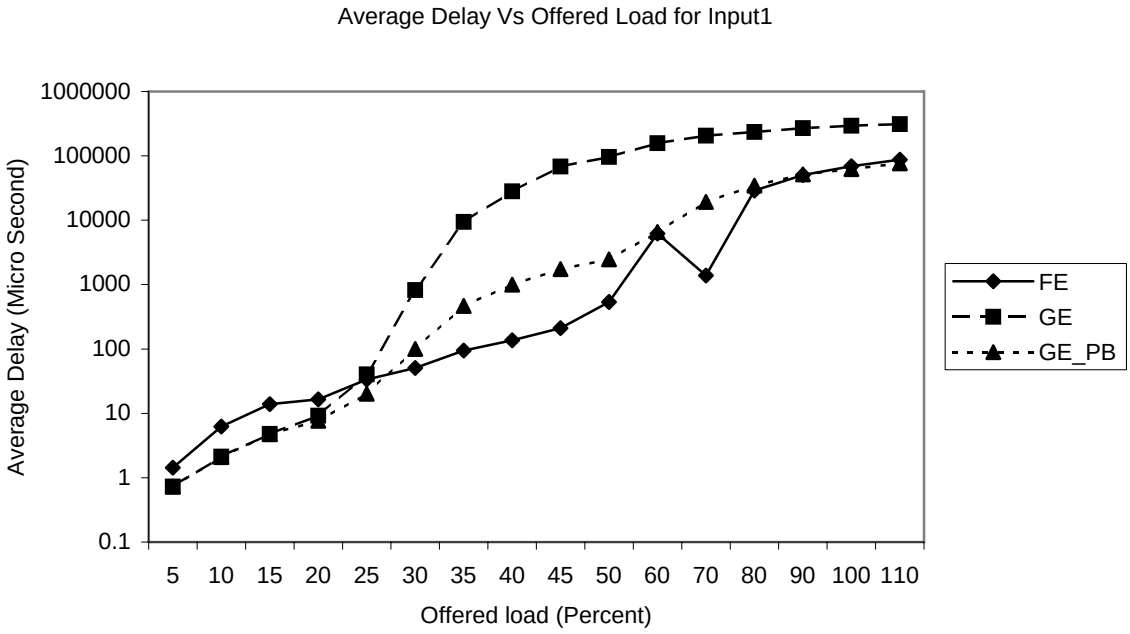

Figure 5.3.9

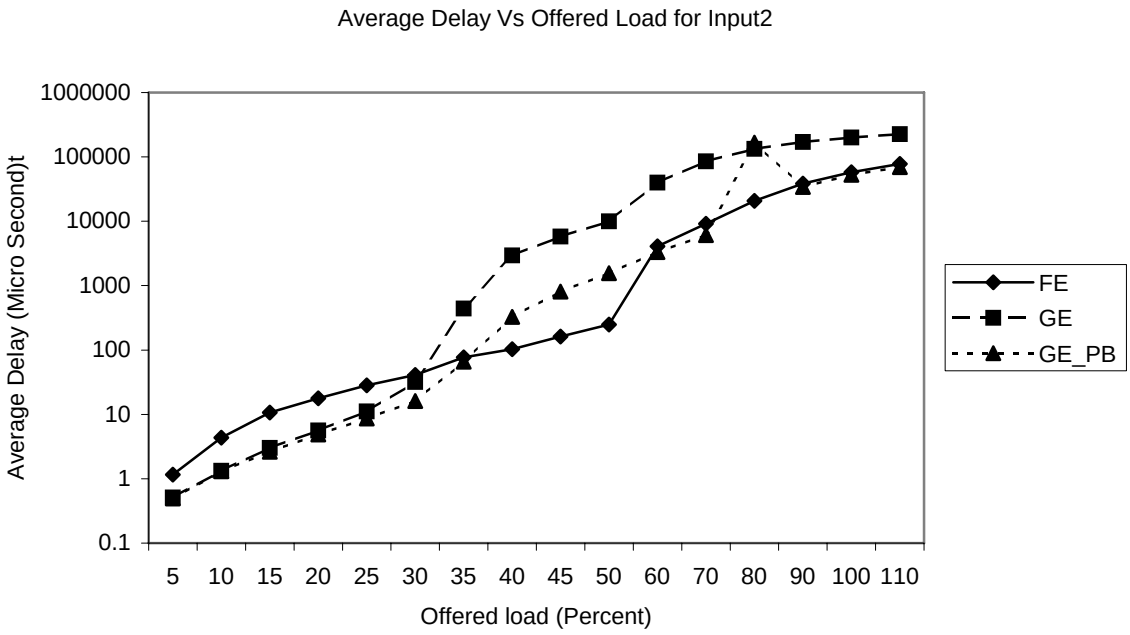

Figure 5.3.10

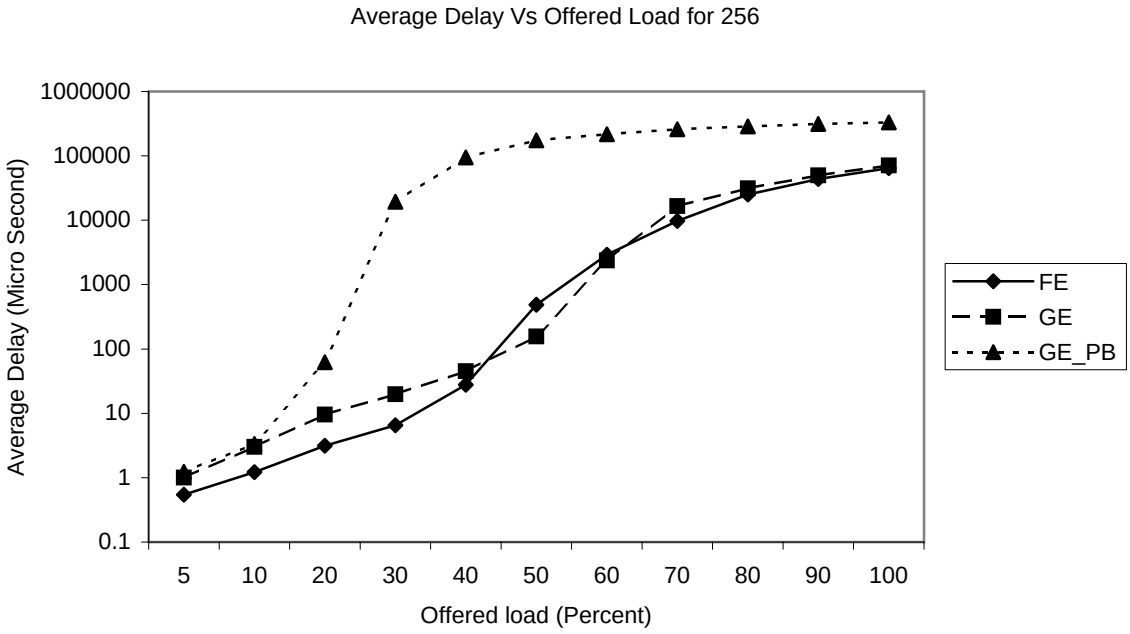

Figure 5.3.11

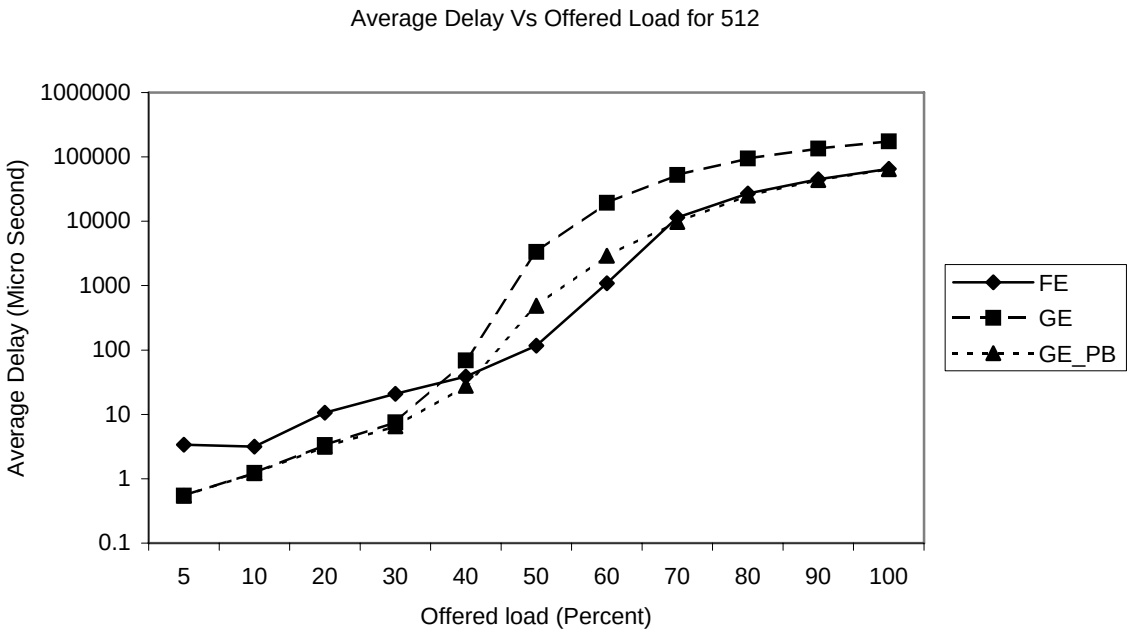

Figure 5.3.12

5.4 Conclusion

Analysis has been done based on these performance metrices throughput, collision/pkt and average packet delays. Observations made from the analysis are as follows:

1. The Carrier Extension allows maintaining a 200 meters network diameter for CSMA/CD operation, however it reduces network efficiency.
2. Carrier extension plus Packet Bursting has approximation 30% higher throughput than without Packet Bursting.
3. Collision/pkt peaks at lower percent offered load for Gigabit Ethernet compared to Fast Ethernet, then decreases due to increased number of packet discards.
4. Average packet delay decreases because of Packet Bursting for Gigabit Ethernet, and much less than that of Fast Ethernet.

From the above observation it is obvious that Packet Bursting improves all measure of performance. Also we can see that Gigabit Ethernet approaches the performance of Fast Ethernet as packet size increases, in term of throughput percent. After the implementing of Packet Bursting, Gigabit Ethernet has become even more efficient at handling small packets.

Chapter-6

Conclusion and Future Scope of the Work

3.1 Conclusions

Performance of Gigabit Ethernet (half-duplex) has been analyzed through discrete event simulation technique. A comparative study has been made against the performance of Fast Ethernet. In the study thus carried on, Gigabit Ethernet has proven to exceed the performance of Fast Ethernet as packet Ethernet as packet size increases. Gigabit Ethernet is also found to be efficient at handling small packets. Simulation results show that in half-duplex topology with collision, Gigabit Ethernet achieves a throughput of over 70% with a 100% offered load. Thus we can conclude from the analysis that Gigabit Ethernet is suitable for handling gigabit of backbone traffic, and hence can be installed as a backbone.

3.2 Future Scope of the work

In this work analysis has been made for half-duplex Gigabit Ethernet with collisions. Full-duplex Gigabit Ethernet switches without collision exist. Therefore a similar simulation study can be made for full-duplex Gigabit Ethernet switches to analyze their performance.

Code

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<malloc.h>
#include<math.h>

#define MODULS 2147483647
#define MULT1 24112
#define MULT2 26143

double clock_time = 0.0;
double tprop, bitrate, jamtime, interfgap, slottime;
double max_arrival = 0.0;
long iteration = 0, tot_iteration, network_type, host_count;
long p1_packets, p2_packets, p3_packets, p4_packets, sz1, sz2, sz3,
sz4;

typedef struct
{
      unsigned long seed;
      long minimum;
      long maximum;
} random_data;

random_data random_hostid, random_slot, random_backoff;

double getRandom(random_data *data)
{

      long tempSeed, lowprd, hi31;

      tempSeed = data->seed;
```

```
	lowprd = (tempSeed & 65535) * MULT1;
	hi31 = (tempSeed >> 16) * MULT1 + (lowprd >> 16);
	tempSeed   = ((lowprd & 65535)- MODULS) + ((hi31 & 32767) <<
16) + (hi31 >> 15);

	if (tempSeed < 0)
		tempSeed += MODULS;

	lowprd = (tempSeed & 65535)*MULT2;
	hi31 = (tempSeed>>16) * MULT2 + (lowprd >> 16);
	tempSeed   = ((lowprd & 65535) - MODULS ) + (( hi31 & 32767)
<< 16) + ( hi31 >> 15);

	if (tempSeed < 0)
		tempSeed += MODULS;

	data->seed = tempSeed ;

	return fabs((((tempSeed >> 7) | 1) + 1) / 16777216.0);
}

long getUniformRandom(random_data *data)
{

	double u;
	do
	{
		u = getRandom(data);
	}
	while (u <= 0.0);
	return (long)((data->maximum - data->minimum) * u + data-
>minimum + 0.5);

}

long getRandomBackoff(long maximum)
```

```
{
      random_backoff.maximum = maximum;
      return getUniformRandom(&random_backoff);
}

void init_random(long hosts, long slots)
{
      double seed = 1973272912.0;
      random_hostid.minimum = 1;
      random_hostid.maximum = hosts;
      random_slot.minimum = 0;
      random_slot.maximum = slots;
      random_backoff.minimum = 0;
      seed = fmod(715.0 *seed, 2147483647.0);
      seed = fmod(1058.0*seed, 2147483647.0);
      seed = fmod(1385.0*seed, 2147483647.0);
      random_hostid.seed = (unsigned long)seed;
      seed = fmod(715.0 *seed, 2147483647.0);
      seed = fmod(1058.0*seed, 2147483647.0);
      seed = fmod(1385.0*seed, 2147483647.0);
      random_slot.seed = (unsigned long)seed;
      seed = fmod(715.0 *seed, 2147483647.0);
      seed = fmod(1058.0*seed, 2147483647.0);
      seed = fmod(1385.0*seed, 2147483647.0);
      random_backoff.seed = (unsigned long)seed;
}

long getRandomHost()
{
      return getUniformRandom(&random_hostid);
}

double getRandomArrivalTime()
{
      return getUniformRandom(&random_slot) * slottime;
}
```

```
typedef struct
{
      long status;//Indication of host status
      long hostid;//This is used to reconize host
      double time;//Time of next packet arrival
} host_desc;

host_desc host[66];

typedef struct
{
      long thruput;
      double delay;// Total delay experienced*/
      long  success;//Successful transmission*/
      long collision;//Total overall collision experienced*/
      long discards;
} expt_stats ;

expt_stats stats;

struct event
{
      struct event *ptr;
      double time; //Time at which struct event occur
      short  hostid;//Location  of  the  host  ehich  generated  the
struct event
      short size;//Length of the packet
      short retries;//Attempt before success
};

// The global struct event queue front and rear
struct event *host_front[66], *host_rear[66], *front, *rear;
```

```
void queue_add(struct event **f, struct event **r, struct event *
e)
{
      struct event *temp = *f;
      if(*f==NULL)
      {
            e->ptr = NULL;
            *f=*r=e;
      }
      else if((*f)->time > e->time)
      {
            e->ptr = *f;
            *f=e;
      }
      else
      {
            /*traverse the entire  the queue to search the position
to insert the new node*/
            while(temp!=NULL)
            {
                  if(temp->time <= e->time && (temp->ptr==NULL ||
temp->ptr->time > e->time))
                  {
                        e->ptr=temp->ptr;
                        temp->ptr=e;
                        if(e->ptr==NULL)
                              *r=e;
                        return;
                  }
                  temp=temp->ptr; /*go to next struct event*/
            }
            printf("\nMust Never Reach Here");
      }
}

struct event * queue_remove(struct event **f, struct event **r)
```

```
{
	struct event *q;
	if(*f==NULL)
	{
		printf("queue is empty");
		return NULL;
	}
	else
	{
		q=*f;
		*f=q->ptr;
		if(*f==NULL)
			*r=NULL;
		return q;
	}
}

/*Function to delete struct event queue*/
void deleventq(struct event **f, struct event **r)
{
	struct event *q;
	q = queue_remove(f, r);
	if(q != NULL)
		free(q);
}

/*adds a new element to the queue*/
void addq(double time, short hostid, short size, short retries)
{
	struct event *e;
	/*create new node*/
	e = (struct event *) malloc(sizeof(struct event));
	if(e == NULL)
	{
		printf(" M_A ");
		return;
```

```
	}
	e->time=time;
	e->hostid=hostid;
	e->size=size;
	e->retries=retries;

	queue_add(&(host_front[hostid]), &(host_rear[hostid]), e);
}

/*displays all element of the queue*/
void q_display(struct event *q)
{
	/*traverse the entire linked list*/
	while(q != NULL)
	{
		printf("\n Time = %f HostId = %d PacketSize = %d
Retries = %d", q->time, q->hostid, q->size, q->retries);
		q=q->ptr;
	}
}

/*count the number of nodes present in the lined list representing
a queue*/
long count(struct event *q)
{
	long c=0;
	/*traverse the entire linked list*/
	while(q!=NULL)
	{
		q=q->ptr;
		c++;
	}
	return c;
}

void addPackets(double start_time)
```

```
{
	long pc;
	for(pc = 0; pc < p1_packets; pc++)
	{
		double	arrival_time	=	start_time	+
getRandomArrivalTime();
		addq(arrival_time, getRandomHost(), sz1, 0);
		if(arrival_time > max_arrival)
			max_arrival = arrival_time;
	}
	//printf("Added SZ1 = %ld", p1_packets);
	for(pc = 0; pc < p2_packets; pc++)
	{
		double	arrival_time	=	start_time	+
getRandomArrivalTime();
		addq(arrival_time, getRandomHost(), sz2, 0);
		if(arrival_time > max_arrival)
			max_arrival = arrival_time;
	}
	//printf("Added SZ2 = %ld", p2_packets);
	for(pc = 0; pc < p3_packets; pc++)
	{
		double	arrival_time	=	start_time	+
getRandomArrivalTime();
		addq(arrival_time, getRandomHost(), sz3, 0);
		if(arrival_time > max_arrival)
			max_arrival = arrival_time;
	}
	//printf("Added SZ3 = %ld", p3_packets);
	for(pc = 0; pc < p4_packets; pc++)
	{
		double	arrival_time	=	start_time	+
getRandomArrivalTime();
		addq(arrival_time, getRandomHost(), sz4, 0);
		if(arrival_time > max_arrival)
			max_arrival = arrival_time;
```

```
	}

	for(pc = 1; pc < host_count; pc++)
	{
		if(host_front[pc] != NULL)
		{
			struct event *temp = front;
			int found = 0;
			while(temp != NULL)
			{
				if(temp->hostid == pc)
				{
					found = 1;
					break;
				}
				temp = temp->ptr;
			}
			if(!found)
			{
				struct		event		*e		=
queue_remove(&(host_front[pc]), &(host_rear[pc]));
				if(e->time < clock_time)
				{
					stats.delay += clock_time - e->time;
					e->time = clock_time;
				}
				queue_add(&front, &rear, e);
			}
		}
	}
	//printf("Added  SZ4 = %ld", p4_packets);
	//printf("Count : %ld", count(front));
	//getch();
	//q_display(front);
}
```

```
//Function to compute backoff time
double backoff(long nattempt)
{
	double backoff_time;
	long k, minvalue;

	if(nattempt<=10)
		minvalue=nattempt;
	else
		minvalue=10;

	k=(long)pow(2,minvalue);
	backoff_time = getRandomBackoff(k) * (4096 / bitrate);
	return(backoff_time);
}

//On successful transmission of packet
void en_successful(host_desc *h1, long packets)
{
	stats.success++;
	stats.thruput = stats.thruput + (packets * 8);
	h1->status = 0;
	clock_time += ((packets * 8.0) / bitrate) + tprop;
	deleventq(&front,&rear);
	if(host_front[h1->hostid] != NULL)
	{
		struct	event	*e	=	queue_remove(&(host_front[h1-
>hostid]), &(host_rear[h1->hostid]));
		if(e->time < clock_time)
		{
			stats.delay += clock_time - e->time;
			e->time = clock_time;
		}
		queue_add(&front, &rear, e);
	}
}
```

```
//On successful transmission of packet
void en_successful_g(host_desc *h1, long packets)
{
	stats.success++;
	stats.thruput = stats.thruput + (packets * 8);
	h1->status = 0;
	if(packets >= 512)
		clock_time += ((packets * 8.0) / bitrate) + tprop;
	else
		clock_time += ((512 * 8.0) / bitrate) + tprop;
	deleventq(&front,&rear);
	if(host_front[h1->hostid] != NULL)
	{
		struct event *e = queue_remove(&(host_front[h1->hostid]), &(host_rear[h1->hostid]));
		if(e->time < clock_time)
		{
			stats.delay += clock_time - e->time;
			e->time = clock_time;
		}
		queue_add(&front, &rear, e);
	}
}

void deleventq_int(struct event **f, struct event **r, struct event *d)
{
	if(*f == d)
		deleventq(f,r);
	else
	{
		struct event *temp = *f;
		while(temp != NULL)
		{
			if(temp->ptr == d)
```

```
                {
                        temp->ptr = d->ptr;
                        if(*r == d)
                        {
                                *r = temp;
                        }
                        free(d);
                        break;
                }
                temp = temp->ptr;
        }
    }
}

//On successful transmission of packet
void en_successful_g_pb(host_desc *h1, long packets)
{
    struct event *temp;
    stats.success++;
    stats.thruput = stats.thruput + (packets * 8);
    h1->status = 0;
    if(packets >= 512)
        clock_time += ((packets * 8.0) / bitrate);
    else
        clock_time += ((512 * 8.0) / bitrate);
    deleventq(&front, &rear);
    temp = host_front[h1->hostid];
    if(packets < 512)
        packets = 512;
    while(temp != NULL && temp->time < clock_time)
    {
        if(temp->time   >=   max_arrival   &&   iteration   <
tot_iteration)
        {
            iteration++;
            addPackets(max_arrival);
```

```
            }

            if((temp->size + packets) < 1500)
            {
                  stats.success++;
                  stats.thruput = stats.thruput + (temp->size * 8);
                  clock_time += ((temp->size * 8) / bitrate);
                  packets += temp->size;
                  deleventq(&(host_front[h1->hostid]),
&(host_rear[h1->hostid]));
            }
            else
            {
                  break;
            }
            temp = host_front[h1->hostid];
      }
      //printf("Burst : %d", burstCount);
      //getchar();
      clock_time += tprop;
      if(host_front[h1->hostid] != NULL)
      {
            struct   event   *e   =   queue_remove(&(host_front[h1-
>hostid]), &(host_rear[h1->hostid]));
            if(e->time < clock_time)
            {
                  stats.delay += clock_time - e->time;
                  e->time = clock_time;
            }
            queue_add(&front, &rear, e);
      }
}

//Function to defer the packet
//long     en_defer(host_desc     *hd1,host_desc     *hd2,expt_status
*esd,double   timed1,double   timed2,double   tpd1,long   psized1,long
```

```
psized2,double    delayd1,double    delayd2,long    retriesd1,long
retriesd2,long typed1,long typed2,double tpropd,double interfgapd)
void en_defer()
{
      double time;//Time at which struct event occur
      struct event *temp;

      time = clock_time + interfgap;
      while(front != NULL && front->time < clock_time)
      {
            temp = queue_remove(&front, &rear);
            stats.delay += time - temp->time;
            temp->time = time;
            queue_add(&front, &rear, temp);
      }
}

void en_collision(double max_detection_time)
{
      double collision_time;
      struct event *temp;
      double end_time;
      end_time = max_detection_time + jamtime;
      collision_time = clock_time + tprop;
      clock_time = end_time;
      stats.collision++;
      while(front != NULL && front->time < collision_time)
      {
            double temp_time;
            temp = queue_remove(&front, &rear);
            temp->retries++;
            temp_time  =  end_time  +  interfgap  +  backoff(temp-
>retries);
            host[temp->hostid].status=1;
            host[temp->hostid].time = temp->time;
            stats.delay += (temp_time - temp->time);
```

```
            temp->time = temp_time;
            if(temp->retries < 16)
                  queue_add(&front, &rear, temp);
            else
            {
                  host[temp->hostid].status=0;
                  stats.discards++;
                  if(host_front[temp->hostid] != NULL)
                  {
                        struct          event          *e          =
queue_remove(&(host_front[temp->hostid]),          &(host_rear[temp-
>hostid]));
                        if(e->time < clock_time)
                        {
                              stats.delay += clock_time - e->time;
                              e->time = clock_time;
                        }
                        queue_add(&front, &rear, e);
                  }
                  free(temp);
            }
      }
}

//Function for attempt to send packet
void en_send(long ipi_dist_idx1,long len_dist_idx1,long nhost)
{
      double delay1,delay2,time1,time2,tp1;
      short
psize1,psize2,hid1,hid2,retries1,retries2,type1,type2,i=1;


      //initialization  host_desc
      for(i=1;i<=nhost;i++)
      {
            host[i].status=0;
```

```
            host[i].hostid=i;
            host[i].time=0.0;
            host_front[i] = host_rear[i] = NULL;
      }

      stats.thruput=0;
      stats.delay=0.0;
      stats.success=0;
      stats.collision=0;
      //end of initialization of expt_status
      addPackets(0.0);
      //printf("Starting Simulation");
      while(front != NULL && front->time <= 1000000.0)
      {
            struct event *next;
            next = front->ptr;
            hid1=front->hostid;
            psize1=front->size;
            time1=front->time;
            retries1=front->retries;

            if(time1 >= max_arrival && iteration < tot_iteration)
            {
                  iteration++;
                  addPackets(max_arrival);
            }

            clock_time = time1;
            if(next != NULL)
            {
                  //printf("\nT1 = %f H1 = %d S1 = %d", time1,
hid1, psize1);
                  //printf(" T2 = %f H2 = %d S2 = %d", time2, hid2,
psize2);
                  //getch();
```

```
                        if(next->time < (time1 + tprop))
                        {
                              en_collision(next->time + tprop);
                        }
                        else
                        {
                              switch(network_type)
                              {
                                    case 1:
                                    en_successful(&host[hid1], psize1);
                                    break;

                                    case 2:
                                    en_successful_g(&host[hid1], psize1);
                                    break;

                                    case 3:
                                    en_successful_g_pb(&host[hid1],
psize1);
                                    break;
                              }
                        }
                        en_defer();
                  }
                  else
                  {
                        en_successful(&host[hid1], psize1);
                        if(iteration < tot_iteration)
                        {
                              iteration++;
                              addPackets(max_arrival);
                        }
                  }
            }
            //initialization  host_desc
            for(i=1;i<=nhost;i++)
```

```
	{
		while(host_front[i] != NULL)
			deleventq(&(host_front[i]), &(host_rear[i]));
	}
	while(front != NULL)
		deleventq(&front, &rear);
	printf("\n\tSuccesses %ld, Collisions %ld, Discards %ld",
stats.success, stats.collision, stats.discards);
	printf("\n\tThroughput %ld, CollisionsPerPacket %f,
AverageDelay %f", stats.thruput, (stats.collision * 1.0) /
(stats.success * 1.0), stats.delay / stats.success);
	getchar();
}


void init_run(long nhost,long p1,long p2,long p3,long p4,long
ipi_dist_idx1,long len_dist_idx1,long type_of_net)
{
	long bits, pc, jamsize=32, bits_for_100, packets,
offered_load;
	clock_time = 0.0;
	max_arrival = 0.0;
	stats.thruput = 0, stats.success = 0, stats.discards = 0,
iteration = 0;
	front = rear = NULL;
	stats.collision = 0;
	network_type = type_of_net;
	switch(type_of_net)
	{
		case 1:
			tprop=0.8695;
			bitrate=100.0;
			jamtime=jamsize/bitrate;
			interfgap=0.96;
			slottime=5.12;
			tot_iteration = 100;
```

```
                init_random(nhost, (long)(10000.0 / slottime));
                break;
        case 2:
                tprop=0.86956;
                bitrate=1000.0;
                jamtime=jamsize/bitrate;
                interfgap=0.096;
                slottime=4.096;
                tot_iteration = 1000;
                init_random(nhost, (long)(1000.0 / slottime));
                break;
        case 3:
                tprop=0.86956;
                bitrate=1000.0;
                jamtime=jamsize/bitrate;
                slottime=4.096;
                interfgap=0.096;
                tot_iteration = 1000;
                init_random(nhost, (long)(1000.0 / slottime));
                break;
        default:
                break;

}

switch(type_of_net)
{
        case 1:
                bits = 100000000;
                break;
        case 2:
                bits = 100000000;
                break;
        case 3:
                bits = 100000000;
                break;
```

```
            default:
                  break;

      }
      bits_for_100 = (p1 * sz1 + p2 * sz2 + p3 * sz3 + p4 * sz4) *
8;
      printf("\n\tEnter Offered Load : ");
      scanf("%ld", &offered_load);
      packets = ((bits / bits_for_100) * offered_load) / 100;
      p1_packets = (packets * p1) / 100;
      p2_packets = (packets * p2) / 100;
      p3_packets = (packets * p3) / 100;
      p4_packets = (packets * p4) / 100;
      //printf("Packets = %ld", packets);
      en_send(ipi_dist_idx1,len_dist_idx1,nhost);
}

//start of main function
void main(void)
{
      /*INPUT PARAMETERS START  */
      long type_of_net;
      long len_dist_idx1;//Index of packet length distribution
      long ipi_dist_idx1;
      int ch1;//Index of interpacket interval distribution
      long no_of_host;//Number of host

      /*A set of arrival rate of packet.This is used

       to generate packet arrival times*/
      long pktarr_rate[15];

      /*Specifies the no. struct event to be simulated. In general
        successful transmission and collision are considered
       to be events.*/
      long no_of_event;
```

```
      /*If val>1 simulator conduct specifies no of

       repetition of the experiment and present
       average of reported quantities in the output file*/
      long no_of_repitition;

      /*The name of distribution to be used for generating

        either packet lengths or interarrival times*/
      char dist_name[50];

       /*If the val is EXPON the distribution is exponential with
mean parameter 1 .
         If the value is UNIFORM, the distribution is uniform on
the interval from
             the value of parameter1 to value of parameter2.
         If the value is FIXED, the distribution is deterministic
               with the value of parameter 1 as constant value of
the distribution.
       If the value is DESCRETE, the value of the distribution
               is determined from a set of (value, probability)
pairs specified with
               x_val and y_val keywords.*/
      char dist_type[10];

      long i,j,k,sz,t,p1,p2,p3,p4,q1,q2,q3,q4;
      /*INPUT PARAMETER ENDS*/
      int choice;
      //clrscr();
      printf("\n\t\t\tPERFORMANCE
ANALYSIS\n\n\t\t\tOF\n\n\t\t\tGIGABIT
ETHERNET\n\n\t\t\tAGAINST\n\n\t\t\tFAST  ETHERNET");
      getchar();
      //clrscr();
      while(1)
```

```
      {
            //clrscr();
            printf("\n\tAssumptions :");
            printf("\n\t\tThe cable length is the maximum allowed
cable length");
            printf("\n\t\tDistribution of inter-arrival time is
uniform");
            printf("\n\t\tFor Input-I ");
            printf("\n\t\t\tNumber of Stations: 32.");
            printf("\n\t\t\tPacket Length Distribution: ");
            printf("\n\t\t\t\t40%% Are 76 Bytes, 39%% Are 180
Bytes\n\t\t\t\t3%% Are 975 Bytes, 18%% Are 1526 Bytes");
            printf("\n\t\tFor Input-II ");
            printf("\n\t\t\tNumber of Stations: 16.");
            printf("\n\t\t\tPacket Length Distribution: ");
            printf("\n\t\t\t\t40%% are 72 Bytes, 20%% Are 180
Bytes\n\t\t\t\t22%% Are 975 Bytes, 18%% Are 1526 Bytes");
            while(1)
            {
                  printf("\n\n\tChoose The Input Option");
                  printf("\n\t\t1 - Input-I");
                  printf("\n\t\t2 - Input-II");
                  printf("\n\t\t3 - Constant Packet Size of 72
Bytes And 32 Stations");
                  printf("\n\t\t4 - Constant Packet Size of 256
Bytes And 32 Stations");
                  printf("\n\t\t5 - Constant Packet Size of 512
Bytes And 32 Stations");
                  printf("\n\t\t6 - Exit");
                  printf("\n\tYour choice: ");
                  scanf("%d",&choice);
                  switch(choice)
                  {
                        case 1:
                        {
                              ipi_dist_idx1=1;
```

```
        host_count=no_of_host=32;
        no_of_event=1000;
        p1=40;
        p2=39;
        p3=3;
        p4=18;
        sz1=76;sz2=180;sz3=975;sz4=1526;
        break;
    }
    case 2:
    {
        ipi_dist_idx1=1;
        host_count=no_of_host=16;
        no_of_event=1000;
        p1=40;
        p2=20;
        p3=22;
        p4=18;
        sz1=72;sz2=180;sz3=975;sz4=1526;
        break;
    }
    case 3:
    {
        ipi_dist_idx1=1;
        host_count=no_of_host=32;
        no_of_event=1000;
        p1=25;
        p2=25;
        p3=25;
        p4=25;
        sz1=72;sz2=72;sz3=72;sz4=72;
        break;
    }
    case 4:
    {
        ipi_dist_idx1=1;
```

```
                        host_count=no_of_host=32;
                        no_of_event=1000;
                        p1=25;
                        p2=25;
                        p3=25;
                        p4=25;
                        sz1=256;sz2=256;sz3=256;sz4=256;
                        break;
                  }
                  case 5:
                  {
                        ipi_dist_idx1=1;
                        host_count=no_of_host=32;
                        no_of_event=1000;
                        p1=25;
                        p2=25;
                        p3=25;
                        p4=25;
                        sz1=512;sz2=512;sz3=512;sz4=512;
                        break;
                  }
                  case 6:
                        exit(0);
                  default:
                        break;
            }/*end of switch loop*/
            //clrscr();
            printf("\n\n\tChoose The Network Type");
            printf("\n\t\t1 - Fast EthernetT");
            printf("\n\t\t2 - Gigabit Ethernet Without Packet
Bursting");
            printf("\n\t\t3  -  Gigabit  Ethernet  With  Packet
Bursting");
            printf("\n\tYour choice: ");
            scanf("%d",&ch1);
            getchar();
```

```
                  switch(ch1)
                  {
                        case 1:
                              type_of_net=1;
                              break;
                        case 2:
                              type_of_net=2;
                              break;
                        case 3:
                              type_of_net=3;
                              break;
                        default:
                              break;
                  }
                  //clrscr();

      init_run(no_of_host,p1,p2,p3,p4,ipi_dist_idx1,len_dist_idx1,t
ype_of_net);
            }/*end of while loop2*/
      }/*end of while loop1*/
}/*end                of                main                loop*/
```

Bibliography

1. Paul A Farrell, Hong Ong, "Communication Performance over a Gigabit Ethernet Network", 19th IEEE International Performance, Computing and Communication Conference-IPCCC 2000 pp.181-189.
2. J.F. Schoch, Y.K.Dalal, D.D Redell, and R.C.Crane, "Evolution of the Ethernet Local Computer Network", Tutorial Local Network Technology, W.Stalling, Ed, IEEE Computer Society, p.49-66
3. Gerd E. Keiser, "Local Area Networks", Second Edition, Tata McGraw Hill Edition 2002, Singapore.
4. Geoffrey Gordon, "System Simulation", Second Edition, PHI, New Delhi, 1980.
5. W.Stallings, "Data and Computer Communication", Sixth Edition, PHI, New Delhi,2003.
6. A.S. Tanenbaum, "Computer Networks", Fourth Edition, PHI, New Delhi, 2002.
5. J.Banks, J.S.Carson, B.L. Nelson, "Discrete-Event System Simulation", Second Edition, PHI, New Delhi, 1998.
7. IEEE Gigabit Ethernet Working groups papers , presentations http://grouper.iee.org/groups/802/3/z/public/presentations.
8.Building Unlimited Intranetworks / Vol. 1, High Performance LAN Alternatives, http://www.foundrynet.com/infocentre