12-15-2007

Increasing the efficiency of network interface card

Amit Uppal

INCREASING THE EFFICIENCY OF NETWORK INTERFACE CARD

By

Amit Uppal

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

December 2007

INCREASING THE EFFICIENCY OF NETWORK INTERFACE CARD

By

Amit Uppal

APPROVED:

Yul Chu Assistant Professor of Electrical and Computer Engineering (Director of Thesis)	Raymond S. Winton Professor of Electrical and Computer Engineering (Committee Member)
YaroSlav Koshka Assistant Professor of Electrical and Computer Engineering (Committee Member)	Nicholas H. Younan Professor of Electrical and Computer Engineering (Graduate Program Director)
Roger King Associate Dean of College of Engineering	

Name: Amit Uppal

Date of Degree: 15th December 2007

Institution: Mississippi State University

Major Field: Electrical Engineering

Major Professor: Dr. Yul Chu

Title of Study: INCREASING EFFICIENCY OF NETWORK INTERFACE CARD

Pages in Study: 75

Candidate for Degree of Master of Science

A Network Interface Card (NIC) is used for receiving the packets, processing the packets, passing the packets to the host processor. NIC uses the buffer management algorithm to distribute the buffer space among different applications.

This thesis proposes two buffer management algorithms: 1) Fairly Shared Dynamic Algorithm (FSDA) for UDP-based applications; 2) Evenly Based Dynamic Algorithm (EBDA) for both UDP and TCP-based applications

. For the average network traffic load, the FSDA improves the packet loss ratio by 18.5 % over the dynamic algorithm (DA) and by 13.5% over the DADT, while EBDA improves by 16.7 % over the DA and by 11.8% over the DADT. For the heavy network traffic load, the FSDA improves the packet loss ratio by 16.8 % over the DA and

by 12.5% over the DADT while EBDA improves the packet loss ratio by 16.8 % over the DA and by 12.6% over the DADT.

DEDICATION

I would like to dedicate this research to my beloved family.

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my academic advisor, Dr. Yul Chu, who has given direction and support throughout my graduate program and for his constant help and support both technically and emotionally during my studies. I would also like to thank my committee members, Dr. Raymond S. Winton and Dr. Yaroslav Koshka, for their invaluable suggestions and guidance during my research and course work.

Finally, I would like to thank all of my friends at MSU for their constant support away from home.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Data is transmitted from one application to another in the form of packets in a computer network [1]. A packet is the unit of data that is routed between an origin and a destination on the Internet. In a typical end to end communication scenario the client will make a request to the server for some information. The server will respond to the client by sending the requested information. All the information sent to and from the client and the server is in the form of packets. A packet consists of the necessary data for an application program associated with headers, such as TCP header, IP header, etc. The receiver in a network terminal processes and places a packet in a buffer until the application requests the packet. The processing of a packet may involve calculation of the checksums, removal of the headers, and determination of the destination application [2]. After processing, the packets are placed in a packet buffer in a network interface card (NIC), which connects a computer to an Ethernet network. A Network Interface Card is used for receiving the packets, processing the packets, passing the packets to the host processor, and sending the packets to other computers in a network.

A packet buffer is a large shared dual-ported memory [6]. Packets for each application are multiplexed into a single stream. Packet buffer management algorithm

determines whether to accept or reject each packet. The number of total packets accepted divided by the total number of incoming packets is called 'Packet Loss Ratio.' Hence, Packet Loss Ratio is equal to:

$$\text{Packet Loss Ratio} = \frac{\text{Total Number of Packet Accepted}}{\text{Total Number of Incoming Packets}} \quad (1.1)$$

The accepted packet is placed into a logical FIFO (First In, First Out) queue; each application has its own queue in a packet buffer [2-4]. The accepted packet remains in a buffer until the application retrieves it from the buffer. Determining whether to reject the packet or accept the packet is a slow process [5].

In general, incoming packets for different applications at different data rates are placed in a buffer. These accumulated packets in the buffer can reduce the available buffer space for the next incoming packet. Once the buffer is full, further incoming packets will be dropped. Therefore, it is important to reduce packet loss ratio to support any end-to-end applications in a computer network [5] [6]. The Buffer Management algorithm plays a vital role in reducing the packet losses in network terminals. An efficient buffer management algorithm should not only minimize packet losses but also distribute packet losses among different applications evenly. For example, if there are three applications say 'application1,' 'application2' and 'application3' with packet sizes of 128, 128, and 256 Bytes respectively. For packet losses to be evenly distributed, 'application1' should have almost same packet loss ratio as those of 'application2' since their packet sizes are equal (each 128 Bytes). However, for the 'application3', packet loss ratio can be greater than the packet loss ratio for 'application1' or 'application2'. This is

due to the fact that the packet size of ‘application3’ is greater than the packet sizes for ‘application1’ and ‘application2.’

1.1 Problem Statement and Motivation

It is essential to have a buffer management algorithm that can utilize maximum memory of the packet buffer and also be able to distribute packet losses among different applications more evenly. An application may use TCP or UDP, depending upon the type of application. TCP (Transmission Control Protocol) is the most commonly used protocol on the Internet. The reason for this is because TCP offers error correction. For TCP applications the sender first sends small number of packets and then waits for the response from client [22]. The client will respond when it receives those packets. At the time of waiting, no packets would be sent to client. Data critical applications should use TCP [1]. On the other hand, for applications that use UDP, there is no wait period. The sender does not wait for any response for client. There may be loss of some packets but that has no effect on the system. Time-critical applications like multi-media applications use UDP. UDP is faster than TCP because there is no form of flow control or error correction. TCP ensures that the data that the reader gets exactly what was sent, in the right order [22]. UDP offers no such guarantees [22]. So, the buffer management algorithm for UDP applications needs not to care even if there is some loss of packets which the sender is not aware of. Hence, this thesis compares the algorithm designed for UDP in a different section than the algorithm designed for TCP.

There are many algorithms reported in the architecture [16]. Popular algorithms include Completely Shared algorithm, Completely Partitioned algorithm, Dynamic

algorithm and Dynamic Algorithm with Dynamic Threshold [16]. None of the algorithms except for Completely Shared algorithm makes full utilization of the packet buffer memory. That is, packets are rejected even if there is some space left in the packet buffer. However, Completely Shared algorithm is not adaptive to changing traffic conditions. Any active application can fill the entire buffer, thus rejecting the packets of other applications (Section 3.4.2). Hence, Completely Shared algorithm is not fair to all the applications since it does not distribute packet losses evenly among the different applications.

In Complete Partitioned algorithm, each application gets a fixed amount of space in the buffer. Thus, if any application becomes inactive, the space allocated to it is not utilized. Completely Shared algorithm and Completely Partitioned algorithm are called static threshold schemes since they are not adaptive to changing traffic conditions. These algorithms are simple to implement in the hardware.

The third popular algorithm is the Dynamic algorithm [8] [16]. In Dynamic algorithm, packet buffer space allocated to each application is dynamic and it depends on the amount of space left in the packet buffer. Dynamic algorithm is adaptive to changes in traffic conditions (Section 3.4.3). However, Dynamic algorithm does not take packet sizes into consideration. Also, a packet can be rejected even if there is some space left in the buffer.

The fourth algorithm is Dynamic Algorithm with Dynamic Threshold (DADT) [16]. This algorithm is very similar to Dynamic algorithm. Unlike Dynamic algorithm,

Dynamic Algorithm with Dynamic Threshold takes packet size of applications into consideration.

Of all the above algorithms, DADT has minimum packet loss ratio. DADT minimizes packet losses by increasing the packet losses of application with largest packet size and decreasing the packet losses of applications with less packet size. Hence, it is not fair to all the applications since the packet losses are not evenly distributed. So, we need an algorithm which can reduce packet loss ratio by utilizing maximum buffer memory and at the same time distributing the packet losses more evenly.

The main purpose of this research is to address the following issues:

1) Develop and simulate buffer management algorithm specifically for UDP-based applications that can reduce the overall packet losses in network terminals and utilize maximum packet buffer memory.
2) Develop and simulate buffer management algorithm that can be used in both UDP as well as TCP-based applications ands which can reduce the overall packet losses in network terminals and utilize maximum packet buffer memory.

1.2 Summary of Main Contributions

The main contributions of this thesis work are as follows:

1) Proposal of a new buffer management algorithm called Fairly Shared Dynamic Algorithm (FSDA) for protocol processors in an NIC. FSDA utilizes whole packet buffer memory and reduce overall packet losses significantly. This algorithm is primarily designed for multimedia applications and all other applications that use User Datagram Protocol (UDP) (Section 7.5).

2) Proposal of a new buffer management algorithm called Evenly Based Dynamic Algorithm (EBDA) that distributes the packet losses evenly among different applications and also reduces the overall packet losses significantly. This algorithm can be used both in TCP and UDP- based applications.
3) Development of a simulation model for the packet buffer in a protocol processor and performance comparison of the different algorithms.

1.3 Organization

The remainder of the thesis is organized as follows. In Chapter II we briefly describe the existing architectures for network interface card. Chapter III explains the popular buffer management algorithms. After that, Chapter IV introduces new algorithm Fairly Shared Dynamic Algorithm. The working of the algorithm is discussed in detail. Chapter V introduces new algorithm Evenly Based Dynamic Algorithm. Chapter VI explains the simulation environment which we have used to measure and compare the performances of different algorithms. Chapter VII gives the simulation results and analyzes them. Finally, Chapter VIII concludes the thesis and discusses the future work.

CHAPTER II

ARCHITECTURES FOR NETWORK INTERFACE CARD

Network Interface Card is used for receiving the packets and sending the packets to other computers in a network. There are three generations for NIC card reported in the literature [20]:

1) First Generation NIC

2) Second Generation NIC

3) Third Generation NIC

In the first generation NIC, a packet is received by the NIC and the data link layer processing is done by the NIC. After processing, the packet is passed on to the host processor. Hence, first generation NIC handles layer 1 and layer 2 of TCP-IP model. Section 2.1 explains the first generation NIC in detail.

Second generation NIC handles most of the layers 1-3 of TCP-IP model. This results in off-loading the host processor to some extent [20].

In the third generation NIC, additional hardware has been added on the NIC. Third generation systems use an embedded processor to handle layer 4 functionality and exception packets that cannot be forwarded across the fast path. Embedded processor architecture is chosen because ease of implementation and amenability to change are

more important than speed [20]. The algorithms designed and developed in this thesis are for third generation NIC.

2.1 Traditional Architecture for Network Interface Card

Figure 2.1 explains the working of a traditional architecture for an NIC. Traditionally a computer has received packets through an NIC, the Ethernet packet layer is processed on the NIC and the packet is buffered on the NIC, before it is transferred to the main memory [2].

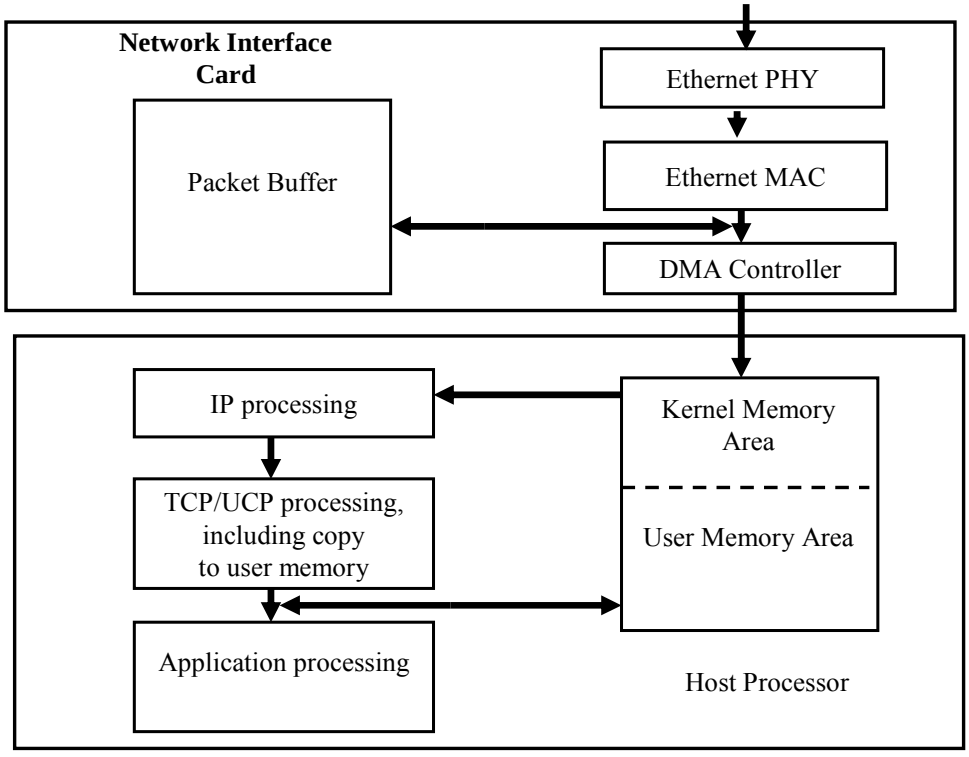


Figure 2.1: Traditional Architecture for Network Interface Card [2].

Then, the IP header and the TCP or UDP header is processed by the OS. Since this implies calculating the checksum over the whole packet, the whole packet has to be read from memory. The OS can also have the memory area divided into one kernel part for the OS and one user part for the applications. This implies a consecutive write operation of the whole packet as well [2].

2.2 Protocol Processor Architecture for Network Interface Card

In the traditional architecture for an NIC, the physical layer and MAC layer processing is done on an NIC [2]. In general, 20%-60% of the processing power of OS is used for protocol handling. Therefore, the traditional packet reception architecture cannot work efficiently for a high-speed network, more than 10 Gb/sec [2].

Tomas Henriksson, et al. [2] proposed protocol processor architecture (third generation NIC) to offload the packet processing from the operating system. The new packet reception shown in Figure 2.2, moves layer 3 and layer 4 processing to an NIC [2]. Packets coming in from the network are received on the NIC and are processed for layer 1-2 protocols. Instead of sending the packet over to the host processor for further processing, the protocol processor on the NIC handles the processing of layer 3 and layer 4 protocols. The main task of a protocol processor is to handle protocol processing at a wire speed [2].

As shown in Figure 2.2, the incoming packets will stream through the protocol processor and the payload (application) data will be stored in the packet buffer until the host application retrieves it. Incoming packets are classified based on the application. Once the packet is classified, it is stored in an output queue for that application in the

buffer. The supporting microcontroller is used for sending acknowledgment to the sender and receiving the next packet.

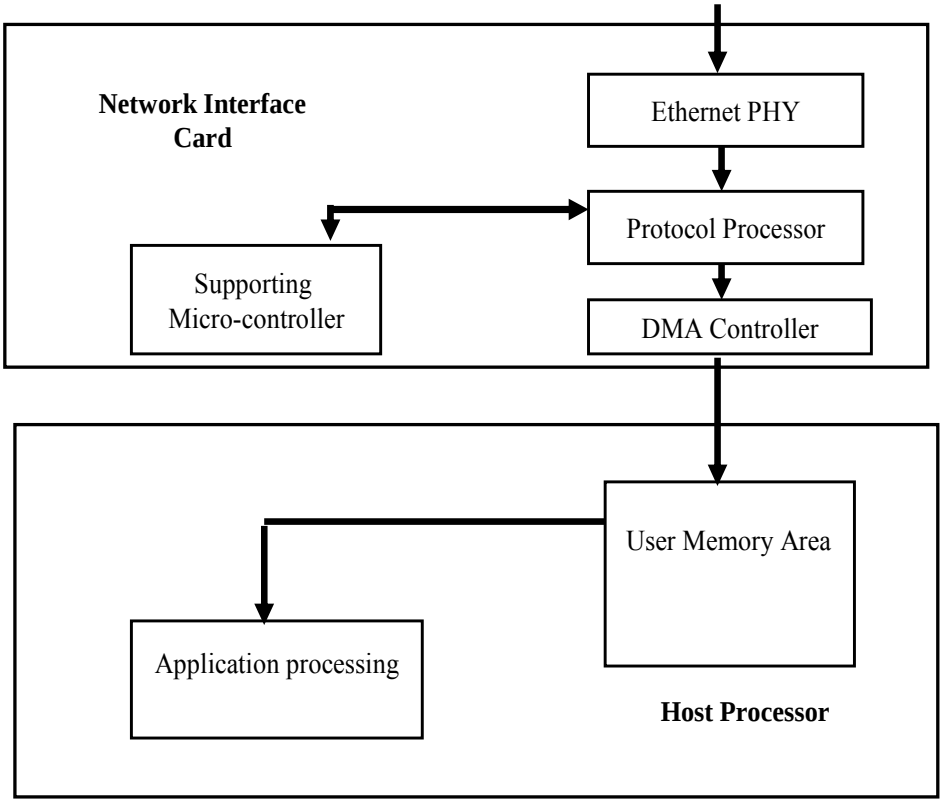


Figure 2.2: Protocol Processor Architecture for Network Interface Card [2].

Figure 2.3 shows the protocol processor core overview. The protocol processor contains accelerators for checksum calculations. The packet is received through the dynamic buffer, which normally holds only one word of data [2]. The dynamic buffer can hold several words of data if that is required. It is controlled from the instruction. Attached to the dynamic buffer is a field extraction unit, which extracts field from the

buffer content. The field is then forwarded to compare units (CU). CU is an array of 'n' comparators. CU gets reference values from parameter code book (PCB) [2]. PCB is a lookup table, with *k* lines of each *n* words. A pointer from the instruction decoder (ID) selects which line to forward to the output.

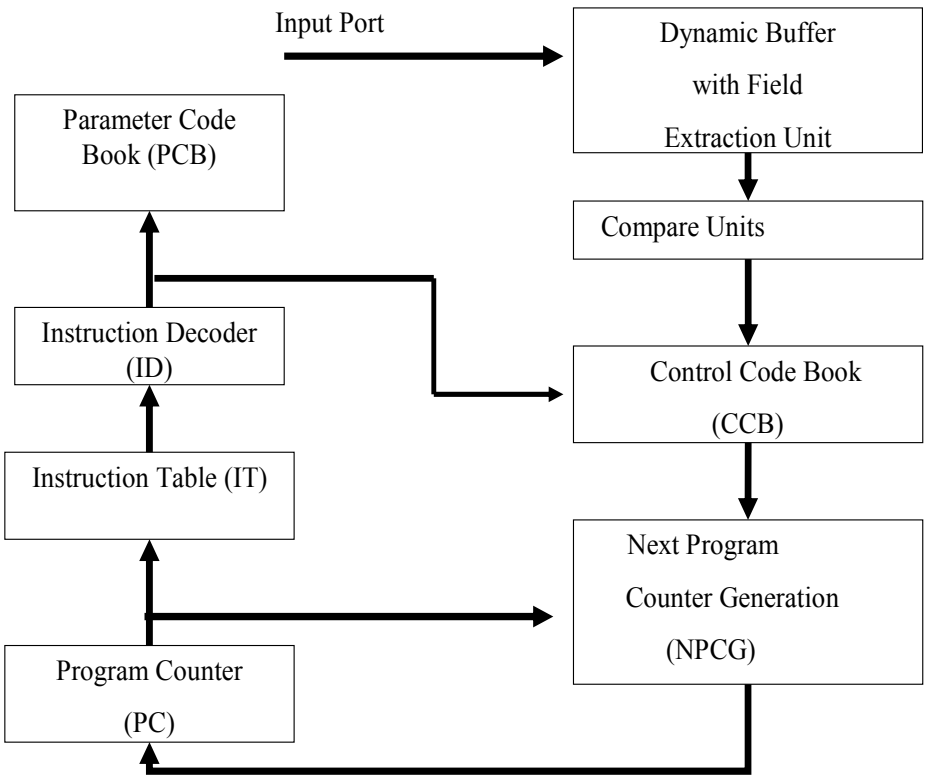


Figure 2.3: Core Overview of Protocol Processor Architecture [2].

The output from the compare units is a vector of *n* bits, in which each bit represents a match or a non-match [2]. These *n* bits are used to select an output from the control code book (CCB). The CCB is another lookup table that contains relative jump

addresses. CCB consists of *k* lines of each *n* addresses [2]. CCB uses the same pointer as PCB in order to select one of the *k* lines. The *n* output bits from the compare units select which address to forward to the next program counter generation (NPCG). The NPCG calculates the next program counter value, which is used by the program counter (PC). The PC is a simple register, which is updated every clock cycle [2]. The output is used to select an instruction from the instruction table (IT). The IT is a lookup table, which contains the instructions for the protocol processor [2].

2.3 Memory Organization Using a Protocol Processor

The simplified packet buffer memory organization is shown in Figure 2.4. As shown in Figure 2.4, incoming packets will stream through the protocol processor and the payload (application) data will be stored in the packet buffer until the host application retrieves it.

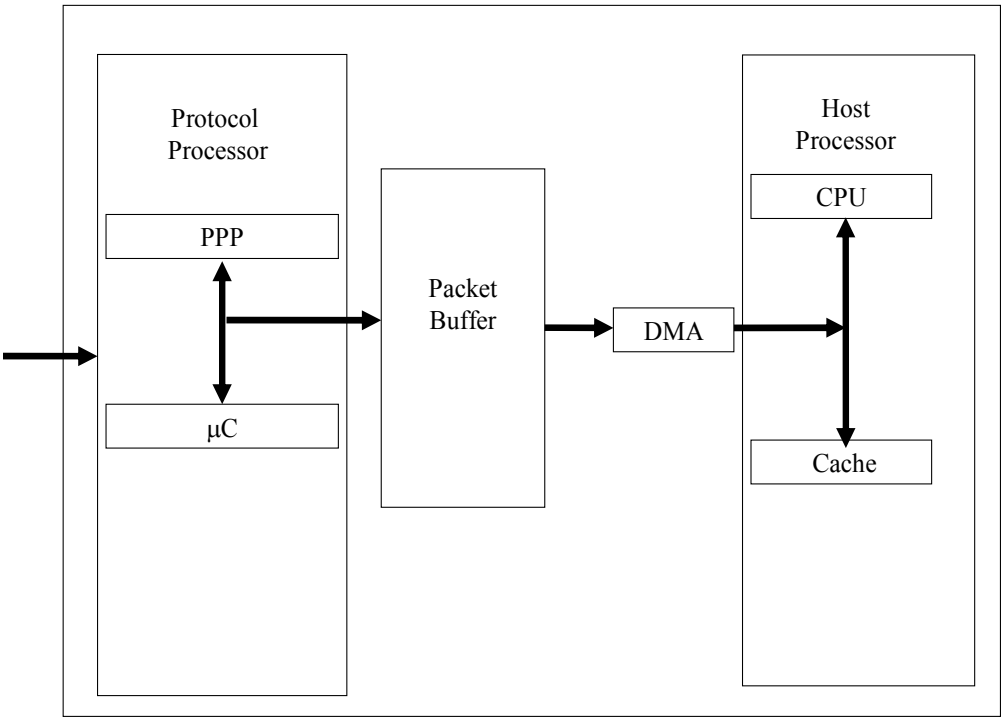


Figure 2.4: Simplified packet buffer memory organization with the protocol processor [2]

First the packets are classified based on the application they are destined for and then they are stored in the output queue for that application in the packet buffer. Each application has an output queue in the buffer. In general, the packet buffer has FIFO based output queues for each application to store its application data [7].

CHAPTER III

BUFFER MANAGEMENT ALGORITHMS

3.1 Role of a Buffer Management Algorithm

After processing of layer 3 and layer 4 protocols, packets are placed in a packet buffer in a network interface card (NIC). Buffer management algorithm determines whether to accept or reject each packet. Figure 3.1 shows the role of a buffer management algorithm in an NIC.

The accepted packet is placed into a logical FIFO queue; each application has its own queue in a packet buffer. In general, incoming packets for different applications at different data rates are placed in a buffer. These accumulated packets in the buffer can reduce the available buffer space for a next incoming packet. Once the buffer is full, further incoming packets will be dropped. Therefore, it is important to reduce packet loss ratio to support any end-to-end application in a computer network [5] [6]. Efficient buffer space management can reduce the packet loss ratio. Buffer management algorithms in an NIC determine how the buffer space is distributed among different applications.

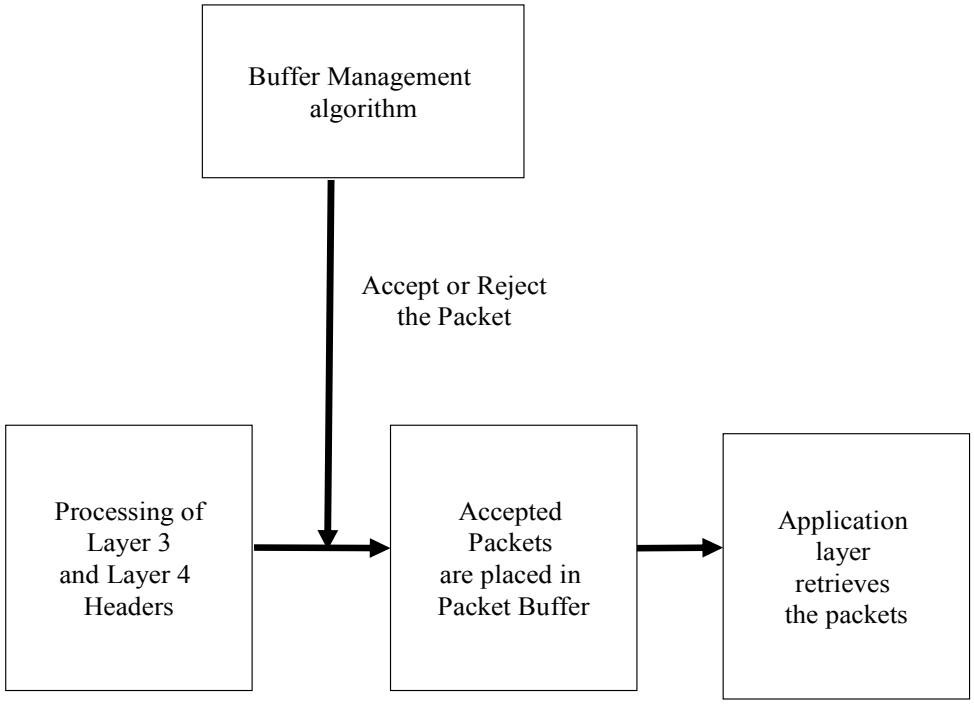


Figure 3.1: Role of Buffer management Algorithm in NIC.

The size of the buffer needed is determined by the packet loss rate. The buffer size must be large enough such that the packet loss ratio does not exceed a certain limit. The required size of the buffer is a function of the incoming traffic rate, the offered load 'ρ', the traffic pattern and also the way the buffer is shared among various output queues [10].

Buffer size = f (traffic rate, offered load ρ *, traffic model, buffer management algorithm)*

3.2 Difference between Buffer Management Algorithms for an NIC and a Switch

Buffer management algorithms in an NIC must be adaptive and intelligent to any changes in traffic conditions. These algorithms are different from what we require in a switch and a hub of the layer 2 **(**MAC Layer). A switch stores all the incoming packets in a common memory buffer that all the switch ports (input/output connections) share. A switch reads the MAC address and sends the packet out to the correct port of the destination node. Hence, the role of a switch is to store and forward a packet to a correct destination [17]. However, in an NIC, a buffer memory must be intelligently shared so that all the applications get fair amount of the buffer space. The aim of the buffer management algorithm should be to minimize the packet loss ratio and simultaneously, be fair to all the applications.

3.3 Design of a Buffer Management Algorithm

The design of a buffer management algorithm needs to consider the following three factors [2]:

1) **Packet loss ratio** - It is defined as the ratio of the number of dropped packets to the total number of received packets [8].

2) **Hardware complexity** - The amount of hardware required to implement a given buffer management algorithm.

3) **Fair to all the applications –** Packet losses should be evenly distributed among different applications.

Buffer space can be managed using either a static threshold scheme or a dynamic threshold scheme among various applications. The static threshold scheme involves establishing the maximum and minimum limits for a buffer space available for each application [16]. In this scheme, a packet is accepted only if the queue length for an application is smaller than the static threshold for the application. The static threshold scheme requires only queue length counters and a comparator [14]. The static threshold scheme is easy to implement in hardware, but it is not adaptive to any changes in traffic conditions. On the other hand, the threshold value of the dynamic scheme is determined by the total amount of unused buffer space at any instant of time. Therefore, the dynamic threshold scheme is adaptive to changes in traffic conditions. In general, the dynamic threshold scheme has less packet loss ratio than the static threshold scheme.

3.4 Popular Buffer Management Algorithms

Four popular buffer management algorithms are reported in literature [16]. They are

- Completely Partitioned Algorithm (CP).
- Completely Shared Algorithm (CS).
- Dynamic Algorithm (DA) and
- Dynamic Algorithm with Dynamic Threshold (DADT).

CP and CS are static threshold schemes, static thresholds; on the other hand, DA and DADT are dynamic threshold schemes, dynamic thresholds.

3.4.1 *Completely Partitioned Algorithm (CP)*

Kamoun and Kleinrock [11] proposed CP. In CP, the total buffer space '*M*' is equally divided among all the applications (N). Hence, CP does not provide any sharing

of a buffer space among different applications. Packet loss for any application occurs when the buffer space allocated to that application becomes full. If '*M*' is the total buffer space, '*n*' is the number of applications and $k_{i,}$ *i= 1....n,* represents the size of queues *i=1....n* then:

$$\sum_{i=1}^{N} k_i = M \qquad (3.1)$$

For example, if the total buffer space is 500 packets, and if the number of applications are 5, then each application gets space for 100 packets. Packet loss for an application occurs when the queue length for an application exceeds 100 packets.

The advantage of this algorithm is that it works well if all the output queues are competing for a buffer space [6]. In addition, it is easy to implement in hardware. However, if all the applications are not competing for the buffer space, then it can reject the incoming packets even though there is some space left in the buffer. Its ability to adapt to the changing traffic conditions is poor because the buffer space allocated to an output queue is not utilized if its corresponding input port becomes inactive.

3.4.2 C*ompletely Shared Algorithm(CS)*

In CS [11], packets are accepted as long as there is some space left in a buffer, independent of the application to which a packet is directed. This algorithm utilizes the whole buffer space. Packet loss occurs only when the buffer is full. If '*M*' is the total buffer space, '*n*' is the number of applications and $k_{i,}$ *i= 1....n,* represents the size of queues *i=1....n* then:

$$k_i = M,\ i = 1, 2, \ldots, N \tag{3.2}$$

For example, if the total buffer space is 500 packets and there are 5 applications, then any one application packets can occupy the entire buffer space, leaving other applications with no buffer space at all. Packets of any application can occupy as much buffer space as possible. The only condition is that accumulative sum of all the queues should not exceed the total buffer space [8].

The algorithm works well under the balanced load conditions. In the balanced load conditions, incoming packets are almost equally distributed among all the applications; hence, this algorithm can provide the fairness to all the applications under the balanced load conditions [16]. In addition, it is easy to implement in hardware. The major drawback of this algorithm is that a single application can occupy the whole buffer space if the load of the application is high. Therefore, it does not guarantee fairness to all the applications.

3.4.3 *Dynamic algorithm(DA)*

When only one application is active, we would like to allocate the maximum buffer space to it. When there are many active applications, we want to divide the memory fairly among them [14]. Dynamic algorithm achieves this by changing the threshold value dynamically, based on the traffic conditions. The threshold value is determined by monitoring the total amount of an unused buffer space

In DA, packets for any application are accepted as long as the queue length for the application is less than the threshold value of that application. Packet loss occurs only when the queue length of an application exceeds its threshold value. If at any instant 't', T (t) be the control threshold and let Q_i (t) be the length of queue 'i.' Q (t) is the sum of all the queue lengths [14], then, if '*M*' is the total buffer space, the controlling threshold will be

$$T(t)=\alpha^{*}(M-Q(t)) \tag{3.3}$$

where 'α' is some constant. The 'α' value is generally taken as a power of two (either positive or negative), so that threshold computation is easy to implement in hardware [14]. This algorithm is robust to changing load conditions in traffic and it is also easy to implement in hardware. However, it has a drawback that it rejects packets when the queue length for an application exceeds the threshold value, though there is some space available in the buffer memory. Also, DA works well for ATM switches since packet size for different application is same in ATM switches. However, in an NIC, different applications may have different packet sizes. Hence, DA does not work that efficiently in an NIC.

3.4.4 *Dynamic Algorithm with Dynamic Threshold (DADT)*

The DADT [16] works like DA. In this algorithm, the alpha 'α' value is different for different applications and is dependent on the packet size of an application. Unlike DA, different applications do not have the same threshold value. By varying the threshold

value, DADT does not allow queues with the largest packet size to fill the buffer at a faster rate. In DADT, we have

$$T(t) = \alpha_i * (M - Q(t)) \tag{3.4}$$

where 'α_i' is the proportionality constant and varies for each queue. This algorithm achieves the least packet loss ratio among all the algorithms described above [16].

However, it has a drawback that it does not use the whole buffer space. Therefore, when the queue length for an application exceeds the threshold value of that application, packets are rejected even if there is some space left in a buffer. Also, it is difficult to determine the optimum alpha 'α' value for each application. The optimum alpha values can come out to be different from power of two. In this case, shift registers cannot be used for implementing it in hardware.

CHAPTER IV

FAIRLY SHARED DYNAMIC ALGORITHM

As we discussed, CS utilizes the buffer memory at full. The algorithm, however, is not fair to all the applications and also not adaptive to changing traffic conditions. On the other hand, DA and DADT are adaptive to changing conditions, but they do not utilize buffer memory at full. Thus, packets can be rejected even if there is space left in the buffer. So, to utilize the full memory space in the buffer, to reduce the overall packet loss ratio and to be fair to all the applications, we need one algorithm that will take care of all the three factors.

Therefore, we propose Fairly Shared Dynamic algorithm (FSDA) that will satisfy three factors:

1) Fairness to all the applications.
2) Full utilization of a buffer space.
3) Reduce overall packet loss ratio.

4.1 Working of FSDA

To achieve fairness and full utilization of buffer space, FSDA maintains a flag for each application. This flag will indicate whether or not the application has taken more

space than its threshold value. The threshold value is determined by monitoring the total amount of an unused buffer space.

Figure 4.1 shows the flowchart of FSDA. The following example explains the working of FSDA in more detail. Let us assume that there are two applications: *application one* and *application two*. Total buffer space '*M*' is 50 bytes. For simplicity, let us take alpha value as 2 for two applications, packet size for *application one* as 4 bytes and for *application two* as 8 bytes.

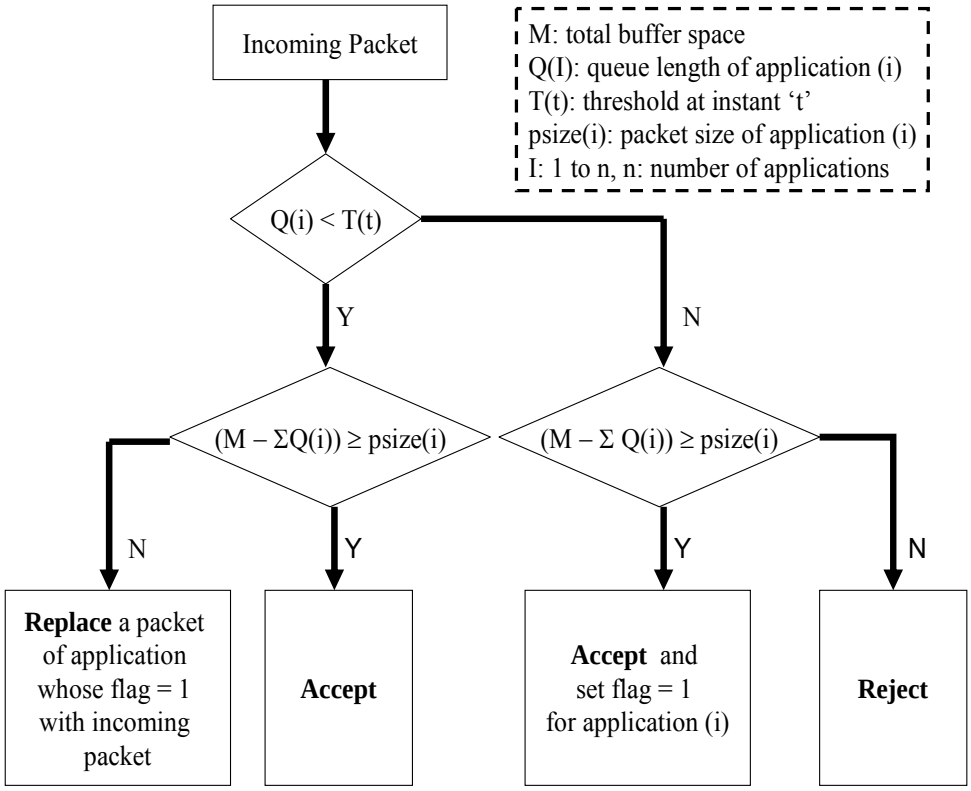


Figure 4.1: Flowchart for FSDA

Say at any instant ‘t’, we have queue lengths (in bytes) as 24 and 16 for *application one* and *application two*, respectively. Figure 4.2 shows the threshold values and buffer state at any instant ‘t.’ Now, if a packet for *application one* comes, then it will be rejected in DA since its queue length (Q(t)=24) exceeds its threshold value (T (t) = 20). On the other hand, FSDA will accept this incoming packet for *application one* and will set the flag for *application one* to ‘1.’

In FSDA, the set flag for *application one* indicates that *application one* has taken more space than its threshold value. Further incoming packets for *application one* will be accepted as long as there is sufficient space in the buffer memory, keeping its flag set to ‘1.’ Figure 4.3 shows the value of flags after the packet for *application one* is accepted. Similarly, for *application two*, packets will be accepted as long as there is some space in the buffer. We will keep the flag of *application two* set to ‘0’ until it takes less space than its threshold value.

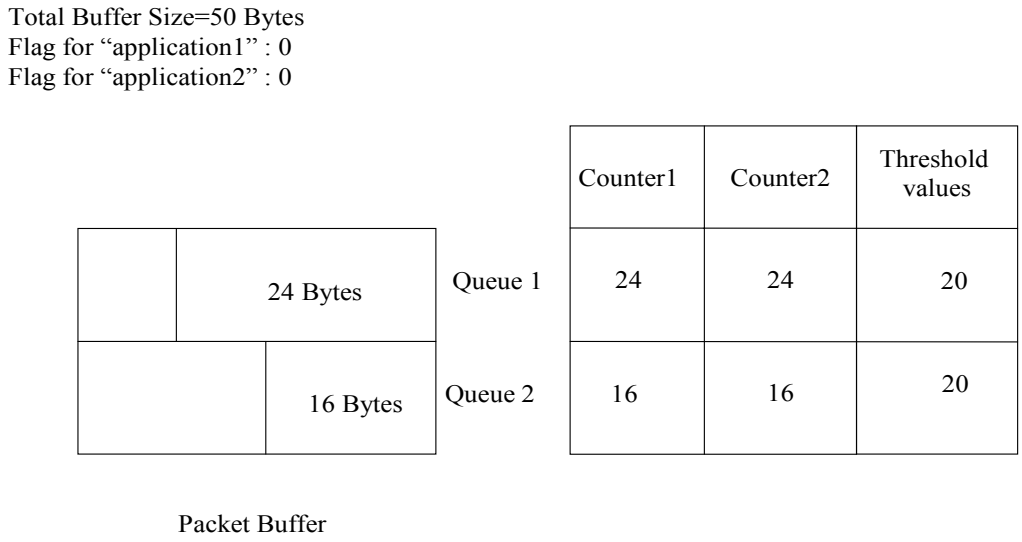


Figure 4.2: Working Example for FSDA.

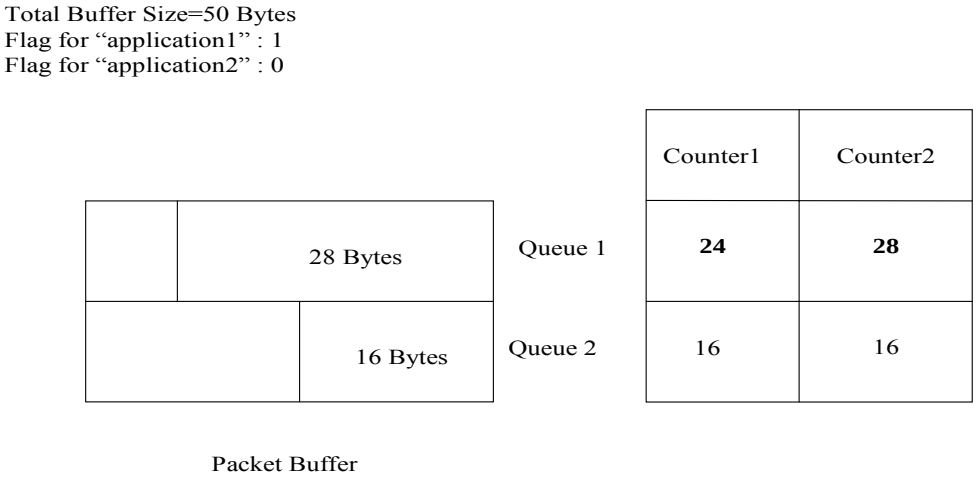


Figure 4.3: Working Example Continued.

Now, there can be two cases when the memory is full:

1) Flag for *application two* is '0' (space occupied by *application two* is less than its threshold value).

2) Flag for *application two* is '1.'

For the case 1, if the current incoming packet is for the *application two*, then we will accept it and replace the packet of the *application one* (since flag for the *application one* is '1') by this incoming packet of the *application two*. This way, we are giving fairness to all the applications and utilizing the whole buffer space simultaneously.

For the case 2, if the incoming packet is for the *application two*, then it will be rejected since there is no space left in the buffer. In FSDA, packets are replaced only when the memory is full and the incoming packet is for an application whose flag is still '0' and there exists an application with its flag as '1'.

As shown in Figure 4.2, in FSDA, we are maintaining two counters, *counter one* and *counter two*, for each application. We will increment *counter one* for an application until the flag for the application is '0.' However, *counter two* for the application will always be incremented whenever the packet for the application is accepted. The value of *counter two* for an application controls the setting and resetting of the flag for the application. The flag for any application will be reset to '0' when the value of *counter two* for that application is less than the threshold value, which is calculated by using *counter one*.

4.2 Advantages of FSDA

The FSDA, DA, and DADT have one major advantage over the static threshold schemes: they are adaptive to changes in traffic conditions [14]. FSDA works similar to DA and DADT. In addition, FSDA utilizes buffer space efficiently. Another advantage of FSDA is that it is more adaptive to changes in traffic conditions. If one application is active, then FSDA will provide the whole buffer space to it, functioning like CS. If many applications are active, then FSDA will work like DA and DADT except for the fact that it will utilize the whole buffer space. Like DA, we keep the 'α' value as a power of 2, which makes its hardware implementation easier. This gives FSDA a distinct advantage over DADT. All the advantages provided by FSDA will come at the cost of more hardware. For example, to implement FSDA more number of counters will be required.

4.3 FSDA for UDP

Though detailed discussion will be done in Section 7.5, FSDA works more efficiently for applications that use UDP. This is due to that fact that information regarding the replaced packets will be lost and the sender will not be aware of the fact that packets have been replaced thus rejected. Hence, FSDA cannot be used for data critical applications or TCP-based applications.

CHAPTER V

EVENLY BASED DYNAMIC ALGORITHM

DADT reduces the overall packet loss ratio by giving less threshold value to the applications with larger packet sizes. This results in an increase in packet losses for applications with larger packet sizes, thus resulting in reducing fairness for applications with large packet sizes.

Table 5.1 compares the packet losses for different queues in DA and DADT. For comparison purposes, we have used six applications, average traffic network load (Chapter 7), buffer size 600 packets, bursty uniform traffic model, and average dequeue time of 14 clock cycles for the burst of 10 packets (Chapter 7).

Table 5.1

Comparison of DA and DADT

Queue	**Packet Size (Packets)**	**Packets rejected in DA**	**Packets rejected in DADT**
0	8	550486	337482
1	2	46901	15841
2	8	547978	335285
3	1	16796	4549
4	4	163126	79563
5	**16**	**1421350**	**1858333**
Total packet loss		*2746637*	*2631053*

As seen from Table 5.1, DADT reduces packet losses for applications 0, 1, 2, 3, 4, by decreasing the amount of threshold value for application 5 [16], which has the largest packet size. Low value of threshold for application 5 results in a reduction of packet losses for applications 0, 1, 2, 3, and 4; though packet losses for application 5 increase significantly. The net result is a decrease in overall packet loss in DADT when compared to DA.

As seen from Table 5.1, by increasing the packet losses for application 5, which already has more packet losses than other applications, degree of fairness has been reduced. Therefore, we proposed Evenly Based Dynamic Algorithm (EBDA) that will take fairness among applications and packet sizes of applications into consideration while allocating buffer space to each application.

5.1 Working of EBDA

Figure 5.1 shows the flowchart of EBDA**.** For any application with packet size 'P', the value of threshold for this application will depend on packet sizes of other applications also. For example, suppose there are six applications with packet size as 256, 256, 512, 128, 64, 128 bytes or 8, 8, 16, 4, 2, 4 packets respectively. The average packet size for these six applications is:

Average Packet size = (8+8+16+4+2+4/6) =7

Now there can be two cases:

1) 'P' is less than Average Packet Size: For such applications the threshold value will be calculated using the equation shown in 5.1.

2) ‘P’ is greater than Average Packet Size: For such applications the threshold value will be calculated using the equation shown in 5.2.

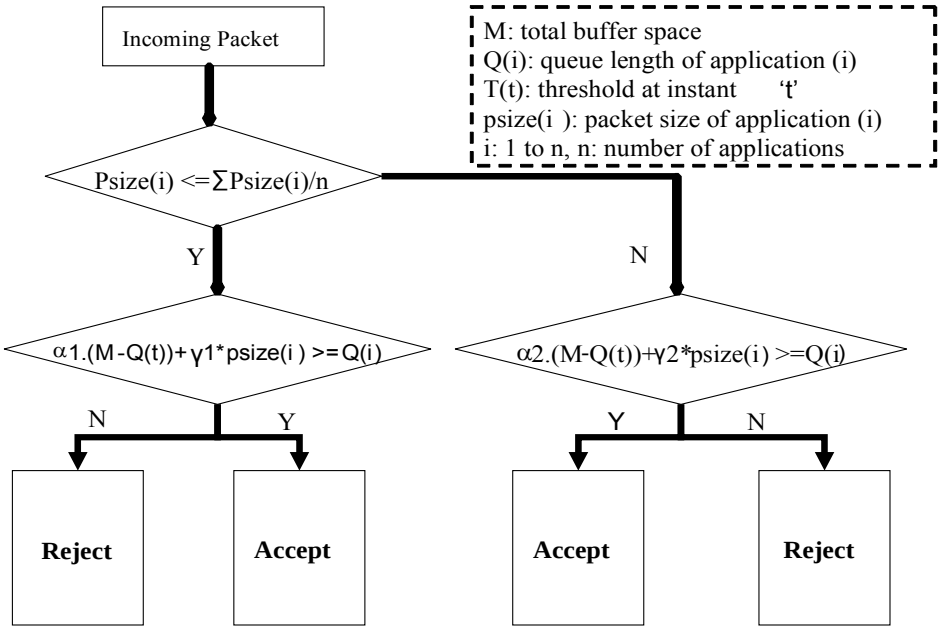


Figure 5.1: Flowchart for EBDA

The idea behind these threshold value computation equations is to distribute the packet losses more evenly among the different applications. Our simulation results have shown that by taking packet size factor in the summation as in equation 5.1 and equation 5.2, instead of multiplication as in DADT for determining the threshold value for the application, we can reduce the overall packet loss ratio as well as distribute the packet losses more evenly among the different applications.

$$T(t)=\alpha_1*(M-Q(t))+\gamma_1*psize(i) \quad (5.1)$$

$$T(t)=\alpha_2*(M-Q(t))+\gamma_2*psize(i) \quad (5.2)$$

The optimum alpha1 (α_1), alpha2 (α_2), gamma1 (γ_1), and gamma2 (γ_2) values as shown in equation 5.1 and 5.2 are determined through simulations.

CHAPTER VI

SIMULATION ENVIRONMENT

We developed our own simulation model instead of using already existing simulators like network simulator (NS) since NS (version 2) [23] is an object-oriented, discrete event driven network simulator developed at UC Berkely written in C++ and OTcl. NS is primarily useful for simulating local and wide area networks. It implements network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, routing algorithms such as Dijkstra, and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations [23].

For our simulations, we have to study the packet loss for different algorithms like DA, DADT, FSDA and DADT. So, we have to write a simulation program for simulating incoming packets for different algorithms. So we have to use a tool in which we could get event driven environment and we could write our own logic. So we used VHDL for our simulations.

The entire simulation model is developed using a Hardware Description Language (HDL) simulator in MODELSIM [1]. VHDL, a Hardware Description Language was chosen to code the entire simulator. We used VHDL since VHDL is a parallel language while C/C++ is a sequential language. Each statement occurring in VHDL is executed

concurrently, that is, all statements run simultaneously. In C/C++ each statement is executed in sequential order.

6.1 Simulation Model for the Packet Buffer

Figure 6.1 shows the diagram of Simulation model for the packet buffer in an NIC.

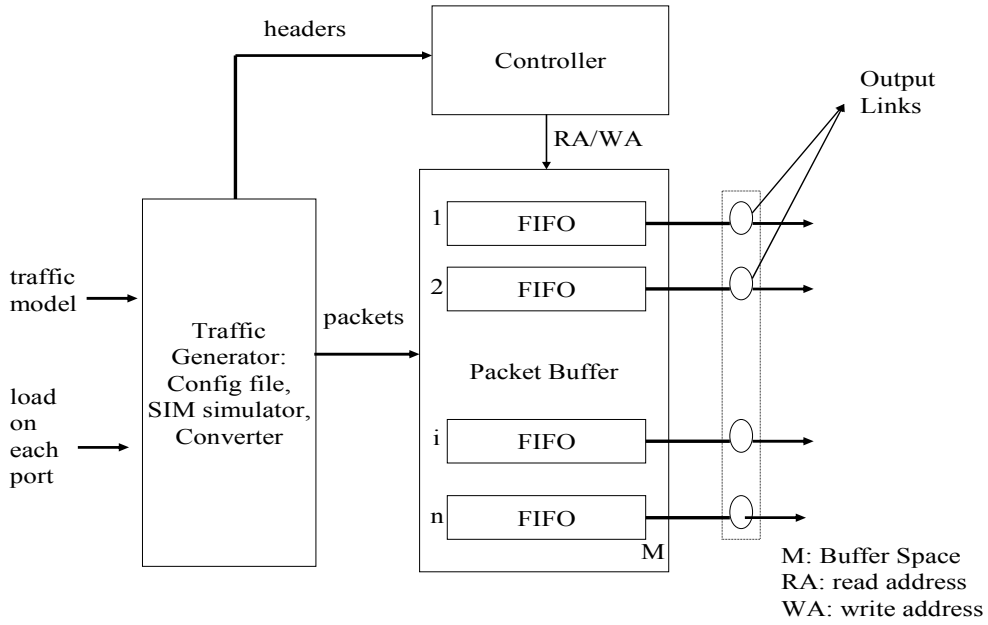


Figure 6.1: Simulation model for the packet buffer.

The *Traffic Generator* block produces output (packets) according to two inputs (Traffic Model and Load on each port).

For the first input, there are three kinds of *Traffic Model* that are available for selection. Those are [6][16]:

- Bursty Uniform Traffic Model: Burst of packets in busy-idle periods with destinations uniformly distributed packet-by-packet or burst-by-burst over all the output ports. The number of packets in the busy and idle periods can be specified; and
- Bursty Non-Uniform Traffic Model: Burst of packets in busy-idle periods with destinations non-uniformly distributed packet-by-packet or burst-by-burst over all the output ports; and
- Bernoulli Uniform Traffic Model: Bernoulli arrivals, destinations uniformly distributed over all the output ports.

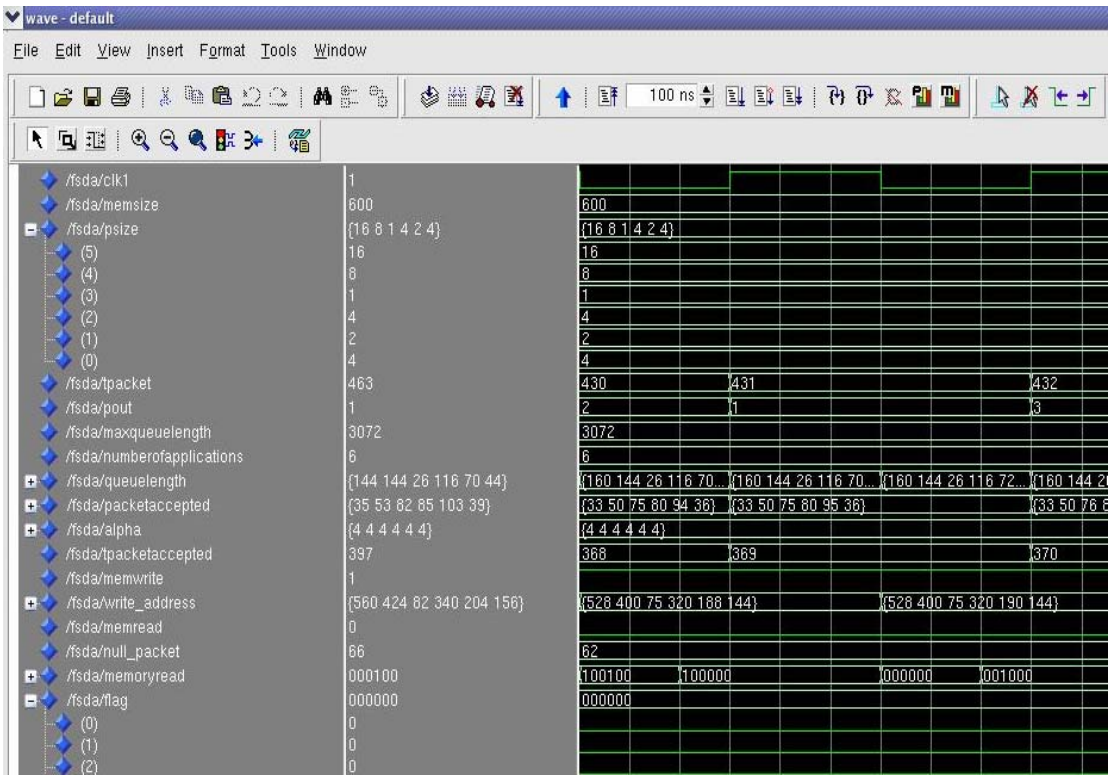
The second input, *Load on each port (ρ),* is determined by the ratio of the number of packets in the busy-idle periods [15] and is given by the equation:

$$\rho = \frac{L_b}{L_b + L_{idle}} \tag{6.1}$$

where L_b = mean burst length and L_{idle}= mean idle length.

For example: For a given load of ρ= 0.7 and a mean burst length of 20 packets, the mean idle length is 10 packets such as $\left(\frac{20}{20+10} = 0.7\right)$.Based on the two inputs, Traffic Generator produces packets (trace file) in a serial fashion with a randomly distributed output destination request. The packets are produced with a mean inter-arrival time and mean burst length [6]. The 'SIM' simulator in [15] is used for producing the trace of packets. In Figure 6.1, once the packet is generated and arrives at the packet buffer, the headers from the Traffic Generator activate the Controller. The Controller then decides to accept or drop the packet based on the buffer management algorithm used. If the packet is

accepted into the buffer, the Controller specifies the write address (WA) based on the output queue to which the packet is destined. Irrespective of whether the packet is accepted or dropped, the Controller updates its state variables (number of packets received, dropped, etc.).



.

Figure 6.2: Sample Waveform for the simulation model

Figure 6.2 shows the sample waveforms and state variables for the simulation model developed for NIC. As seen from the figure 6.2 ‘psize’ represents the size of the

packets of different applications. ‘Memsize’ represent the size of the buffer memory in terms of packets. Variable ‘pout’ represents the destination of current incoming packet.

CHAPTER VII

SIMULATION RESULTS AND ANALYSIS

It has been shown that dynamic threshold schemes are more robust than static threshold schemes for uniform loads [14] [16]. Hence, the dynamic threshold schemes can perform better than the static threshold schemes. Therefore, for our analysis, we will compare our proposed algorithms FSDA and EBDA with the dynamic threshold schemes, DA and DADT.

7.1 Simulation Results for FSDA

Three different network traffic loads are considered for our simulations and comparisons of algorithms: average network traffic load, heavy network traffic load, and actual network traffic load. We have used the "bursty uniform traffic model" for our simulations of all the network traffic loads since it is the most commonly used model.

Figure 7.1 shows the steps performed for performance comparisons of different buffer management algorithms. For each traffic load, first, the optimum alpha 'α' value is determined for DA. After this, the best combination of the alpha values for DADT is determined. This is followed by the performance comparison of DA, DADT, and FSDA when the load and the buffer size are varied. Finally, improvement ratio is determined for each network traffic load. Improvement ratio is defined as the difference of the number of

packet losses in FSDA and the compared algorithm (DA or DADT) divided by the number of packet losses in FSDA. While calculating the improvement ratio and performance analysis, the replaced packets have been taken into consideration for FSDA. For all simulations, we have used six applications, bursty uniform traffic model, and average dequeue time of 14 clock cycles for the burst of 10 packets.

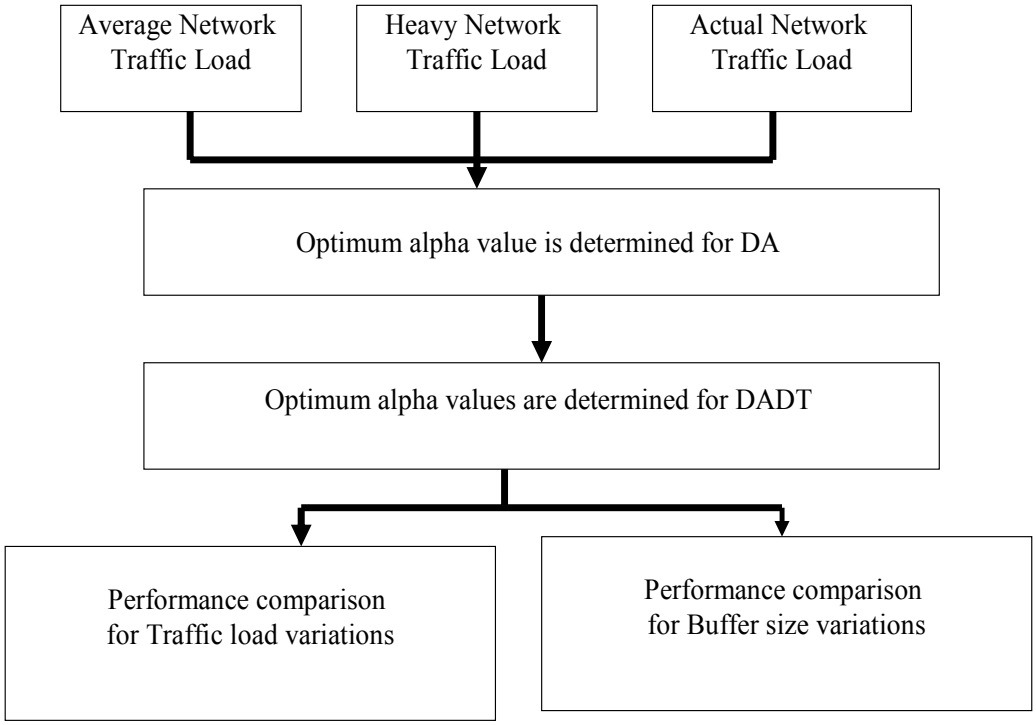


Figure 7.1: Steps performed for comparing DA and DADT with FSDA

7.2 Simulation results for Average Network Traffic load

Table 7.1 shows the packet sizes of different applications in bytes based on the average network traffic load flow in [5]. For our simulation of the average traffic load, we have used these packet sizes for different applications.

Table 7.1

Packet distribution for average traffic flow average network traffic load.

	Queue0	**Queue1**	**Queue2**	**Queue3**	**Queue4**	**Queue5**
Size in bytes	256	64	256	32	128	512
Packet unit # (32 bytes/unit)	8	2	8	1	4	16

7.2.1 *Optimum Value of alpha for DA*

Optimum alpha is considered as the alpha value for which DA gives the minimum packet loss ratio. Figure 7.2 shows the packet loss ratio for DA as the alpha value is varied from 4 to 20. In Figure 7.2, the size of the buffer is 600 packets, and "load on each queue" is 70%. From Figure 7.2, we can see that initially, as the alpha value is increased, packet loss ratio decreases until alpha=14. After then, the packet loss ratio starts increasing because the larger alpha values can increase the control threshold of the queues with large packet sizes. For 'alpha=14' and 'alpha=16', the packet loss ratio is very similar. From a hardware implementation point of view, we will take 'alpha =16

(2^4)' as the optimum value. The reason behind this is that if alpha is a power of 2, shift registers can be used to implement the algorithm in hardware.

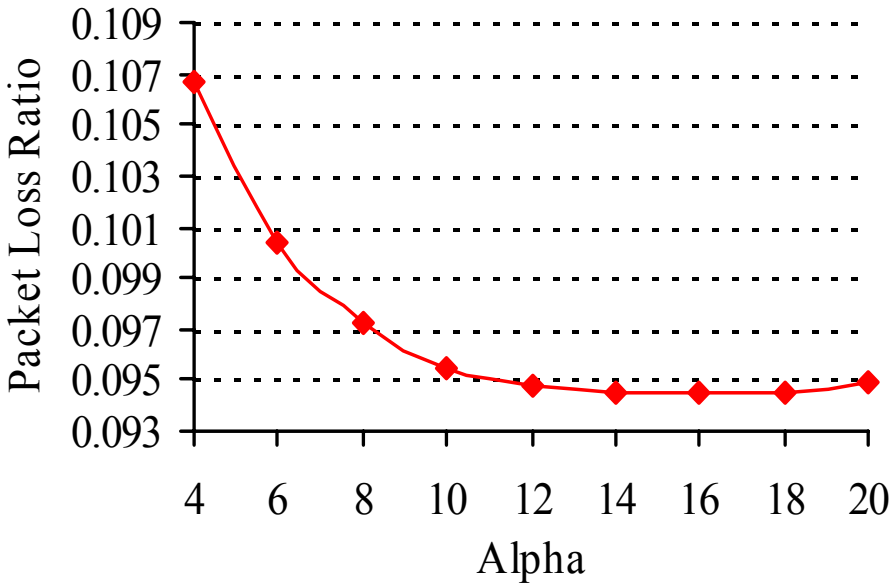


Figure 7.2**:** Packet loss ratio vs. Alpha for DA for the average traffic load

7.2.2 *Optimum Value of alphas for DADT*

For DADT, each queue has a different alpha and different threshold value. For DADT, first we determined the optimum 'α' (alpha) values. Optimum alpha values for DADT is the combination of alpha for different queues for which DADT gives the minimum packet loss ratio for the same load and the same buffer size.

Table 7.2 shows the different combinations of alpha that we have taken and Figure 7.3 shows the packet loss ratio corresponding to them. As seen from figure 7.3, packet losses for variation 5 are less than packet losses for other variations. Note that optimum alpha for application 1 and application 3 for variation 5 comes out to be other than multiple of two. Hence, implementing it in hardware will be more difficult as compared to it would have been in case alpha is multiple of two.

Table7.2

Variation of alpha for DADT for the average traffic load

Variation	**Q0**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**
1	12	10	12	10	10	8
2	14	10	14	10	10	7
3	14	12	14	12	12	8
4	16	14	16	14	14	6
5	16	14	16	14	16	8

So, for our comparison purpose, we will use ‘alpha=16’ for DA and the variation 5 (from table 7.3) as alpha values for DADT. In FSDA, changing the alpha value will have little impact on the performance since FSDA utilizes full memory most of the time. Therefore, ‘alpha =4’ will be used for FSDA. Since 4 is a power of 2; it will make hardware implementation for the FSDA easier.

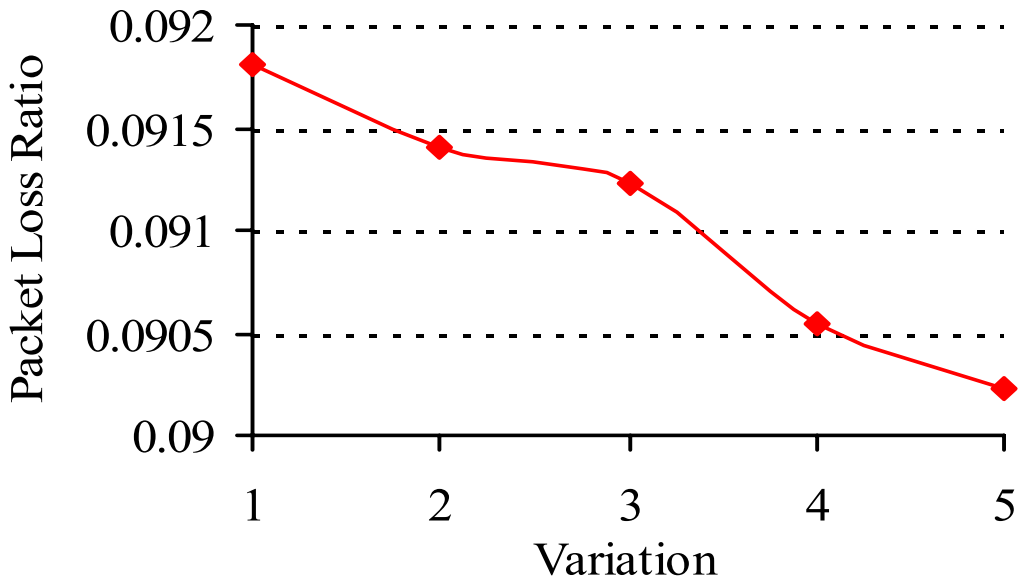


Figure 7.3**:** Packet loss ratio vs. Alpha Variation for DADT for the average traffic load

7.2.3 *Comparison of FSDA, DA and DADT for different loads*

Figure 7.4 shows the performance of the three algorithms (FSDA, DA and DADT) for different load. Load has been varied from 0.5 to 0.9.

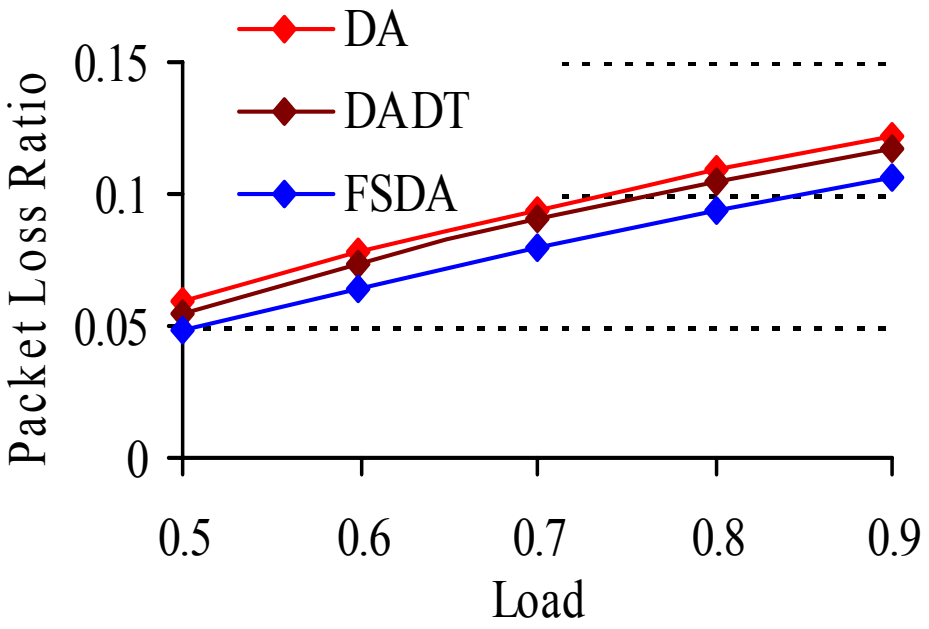


Figure 7.4**:** Packet loss ratio vs. Load for FSDA, DADT, and DA for the average traffic load

Buffer size has been taken as 600 packets. As seen in Figure 7.4, FSDA has the least packet loss ratio for all of loads. The packet loss ratio increases for all the algorithms with increasing "load on the queues". Notice that the performance difference increases more at higher loads. As the load is increased, most applications tend to increase their queue length greater than their threshold values frequently. Since, FSDA utilizes the whole buffer space; FSDA can reduce packet loss ratio efficiently.

7.2.4 *Comparison of FSDA, DA and DADT for different buffer size*

Figure 7.5 shows the performance of the three algorithms FSDA, DA, and DADT as the buffer size is varied from 500 to 800 packets.

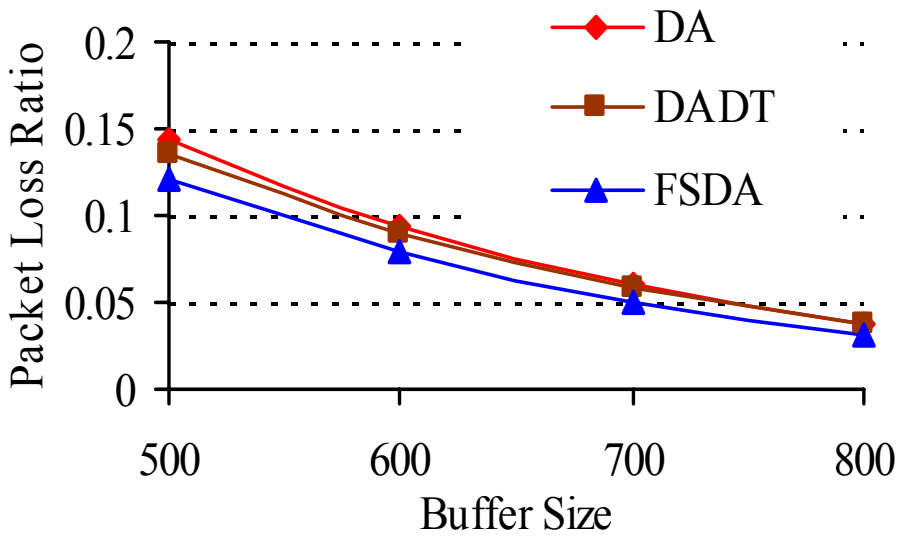


Figure 7.5**:** Packet loss ratio vs. Buffer Size for FSDA, DADT, and DA for the average traffic load

With an increase of buffer size, packet loss ratio decreases for all the three algorithms. This is due to the fact that each queue gets more space to accommodate packets. As seen from figure 7.5, FSDA has least packet loss ratio as compared to other algorithms.

7.2.5 *Improvement ratio of FSDA over DA and DADT*

Table 7.3 shows the improvement in packet loss ratio for 'FSDA over DA' and 'FSDA over DADT' according to different loads, from 0.5 to 0.9. Buffer size has been taken as 600 packets and the traffic model is "bursty uniform". As the load is increased the improvement ratio decreases. This is due to the fact that as the buffer size is increased packet losses are reduced.

Table 7.3

Improvement ratio of FSDA over DA and DADT for the average traffic load.

Load	**Improvement ratio (%) (FSDA/DA)**	**Improvement ratio (%) (FSDA/DADT)**
0.5	23.2	13.8
0.6	21.2	14.0
0.7	18.5	13.5
0.8	15.9	12.2
0.9	13.8	10.2

7.3 Simulation Results for Heavy Network Traffic load

Table 7.4 shows the packet sizes of different applications in bytes based on the heavy network traffic load in [5]. For our simulation of the heavy traffic load, we have used these packet sizes for different applications.

Table 7.4

Packet distribution for heavy traffic flow average network traffic load.

	Queue0	**Queue1**	**Queue2**	**Queue3**	**Queue4**	**Queue5**
Size in bytes	128	64	128	32	256	512
Packet unit # (32 bytes/unit)	4	2	4	1	8	16

7.3.1 *Optimum Value of alpha for DA*

Figure 7.6 shows the packet loss ratio for DA as the alpha value is varied from 4 to 20 for the heavy network traffic load. In Figure 7.6, the size of the buffer is 600 packets, and the "load on each queue" is 70%. The optimum alpha value for DA comes out to be 16**.**

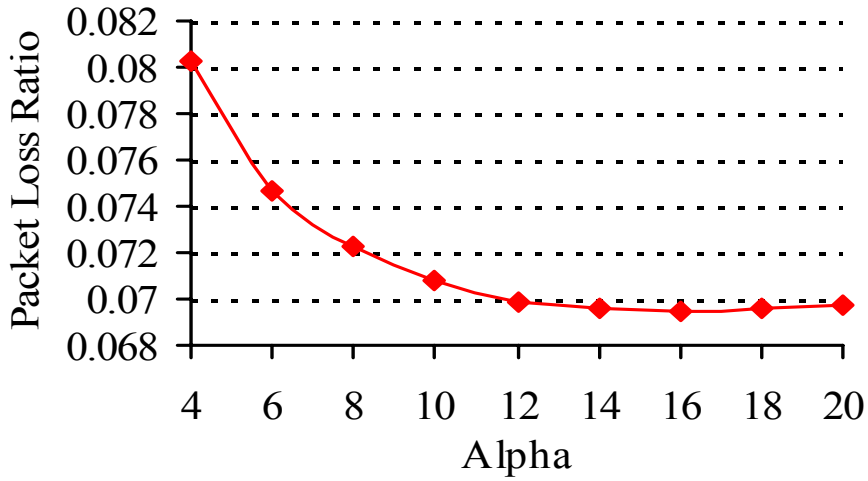


Figure 7.6**:** Packet loss ratio vs. Alpha for DA for the heavy traffic load

7.3.2 *Optimum Value of alphas for DADT*

Now we will determine the optimum values of alpha for DADT. Table 7.5 shows the different combinations of alpha that we have taken.

Table 7.5

Variation of alpha for DADT for the heavy traffic load

Variation	Q0	Q1	Q2	Q3	Q4	Q5
1	18	18	18	18	18	6
2	14	10	14	10	10	7
3	**14**	**12**	**14**	**12**	**12**	**8**
4	16	14	16	14	14	6
5	16	14	16	14	16	8

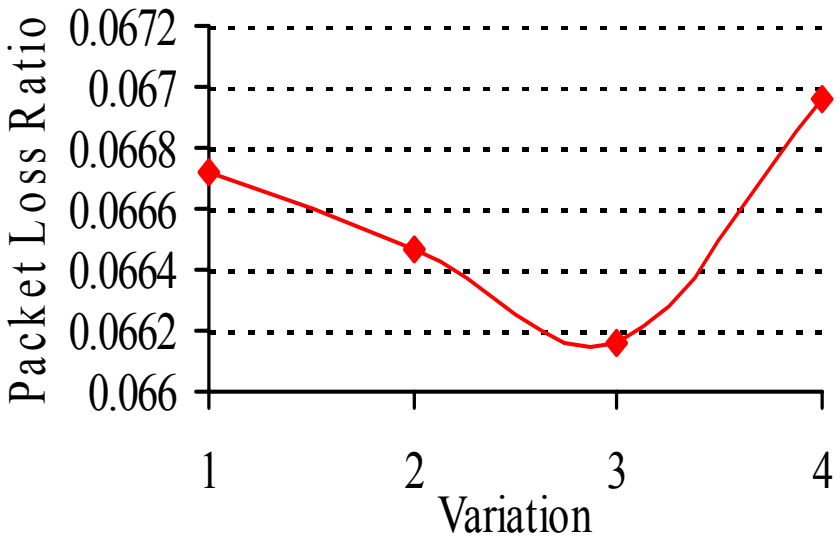


Figure 7.7**:** Packet loss ratio vs. Alpha Variation for DADT for the heavy traffic load

Figure 7.7 shows the packet loss ratio corresponding to them. From Figure 7.7, we can see that optimum combination of alpha comes out of the variation 3.

7.3.3 *Comparison of FSDA, DA and DADT for different loads*

Figure 7.8 shows the performance of the three algorithms (FSDA, DA and DADT) for different loads. Load has been varied from 0.5 to 0.9. Buffer size is taken as 600 packets. As seen from figure 7.8, FSDA outperforms DA and DADT.

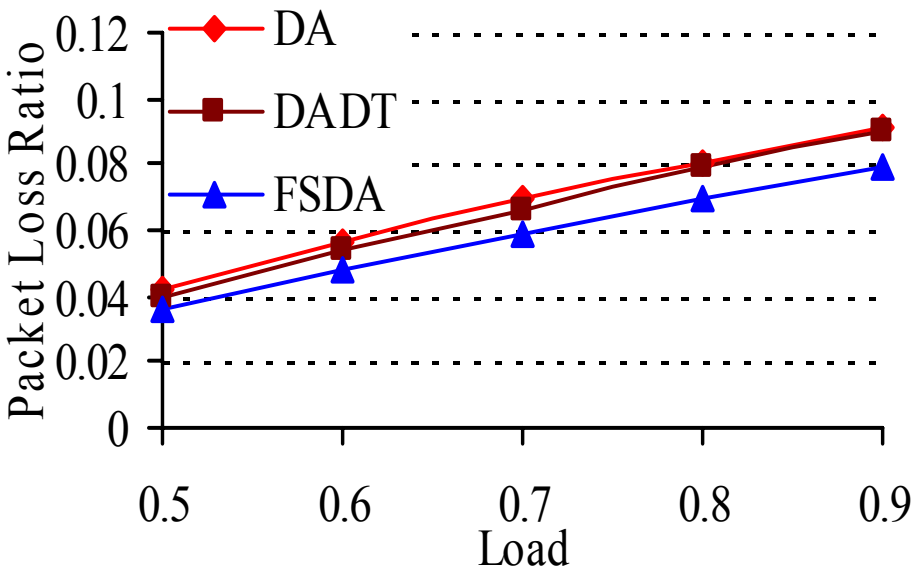


Figure 7.8**:** Packet loss ratio vs. Load for FSDA, DADT, and DA for the heavy traffic load

As the load is increased the performance difference increases. This is due to the fact that as the load is increased, more number of packets is coming and since FSDA

make better utilization of whole memory, the packet losses increase more for DA and DADT as compared to FSDA.

7.3.4 *Comparison of FSDA, DA and DADT for different buffer size*

Figure 7.9 shows the performance of the three algorithms, FSDA, DA, and DADT as the buffer size is varied from 500 to 800 packets. As seen from figure 7.9, FSDA has least packet loss ratio.

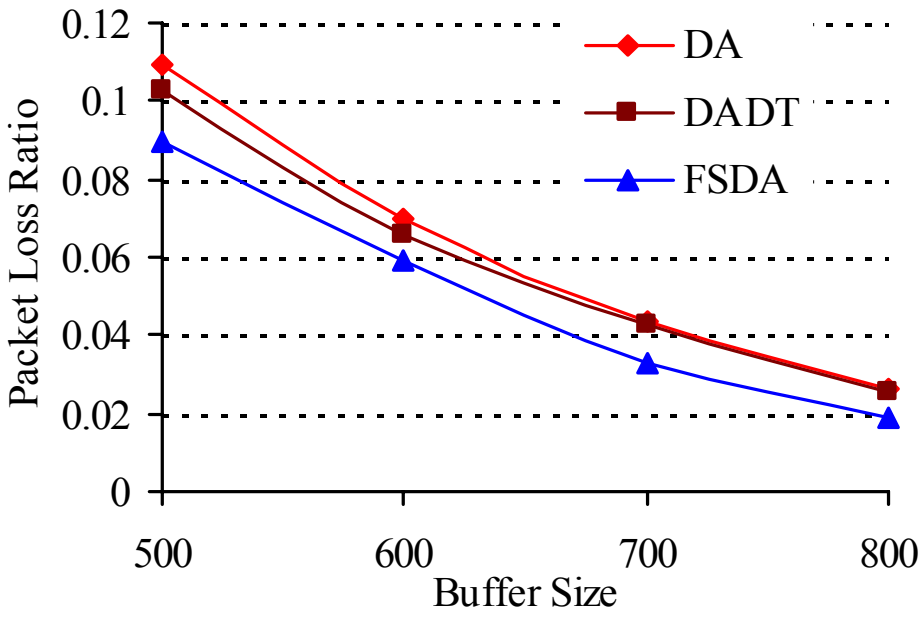


Figure 7.9**:** Packet loss ratio vs. buffer size for FSDA, DADT, and DA for the heavy traffic load

7.3.5 *Improvement ratio of FSDA over DA and DADT*

Table 7.6 shows the improvement in packet loss ratio for 'FSDA over DA' and 'FSDA over DADT' according to different loads, from 0.5 to 0.9. As we can see that for a load of '0.7' improvements ratio is 16.8 when compared with DA and 12.5 when compared with DADT.

Table 7.6

Improvement ratio of FSDA over DA and DADT for the heavy traffic load.

Load	**Improvement ratio (%) (FSDA/DA)**	**Improvement ratio (%) (FSDA/DADT)**
0.5	16.6	10.2
0.6	17.1	11.1
0.7	16.8	12.5
0.8	15.7	13.3
0.9	13.9	12.0

7.4 Simulation Results for Actual Network Traffic load

Table 7.7 shows the packet sizes of the different applications in bytes based on the actual network traffic load flow in [18]. The characteristics of the actual network traffic load presented in this section is based on the data collected by the NAI project from May 1999 to March 2000 at the NASA Ames Internet Exchange [19]. The packet size distribution for the Internet traffic load is as shown in Table 7.7.

Table 7.7

Packet size distribution for an actual network traffic load [19].

Packet Size	**< 44**	**50-500**	**500-600**	**>1500**
Percentage (%)	50	14	18	18

The packet sizes assumed for each queue for simulation purposes are shown in Table 7.8. The packet size distribution resembles the one shown in Table 7.7.

Table 7.8

Packet distribution for Actual flow actual network traffic load.

	Queue0	**Queue1**	**Queue2**	**Queue3**	**Queue4**	**Queue5**
Size in bytes	32	32	32	64	512	1472
Packet unit # (32 bytes/unit)	1	1	1	2	16	46

7.4.1 Optimum Value of alpha for DA

Figure 7.10 shows the packet loss ratio for DA as the alpha value is varied from 4 to 20 for the actual network traffic load. Figure 7.10 shows that the optimum alpha value for DA comes out to be 4. As we can see that as the value of alpha is increased the packet loss ratio is also increased. This is due to the fact that queue 5 has a size of 46 bytes and

any increase in its threshold value will result in great increase in packet losses of other applications.

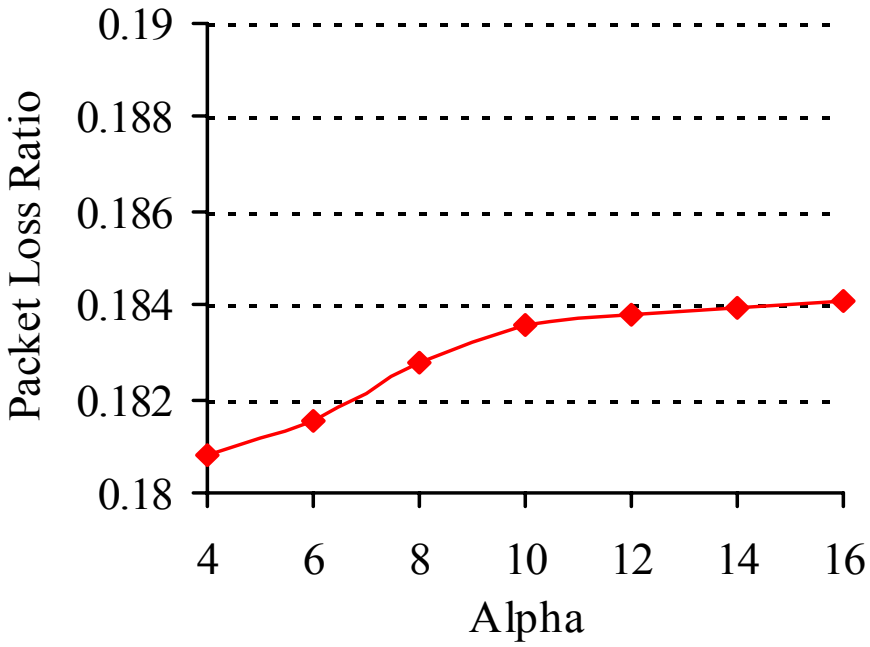


Figure 7.10**:** Packet loss ratio vs. Alpha for DA for the actual traffic load

7.4.2 Optimum Value of alphas for DADT

Table 7.9 shows the different combinations of alpha for DADT and figure 7.11 shows the packet loss ratio corresponding to them. From figure 7.11 we can see that the optimum alpha values come out of the variation 5.

Table 7.9

Variation of alpha for DADT for the actual traffic load

Variation	Q0	Q1	Q2	Q3	Q4	Q5
1	16	16	16	16	6	4
2	16	16	16	16	6	6
3	18	18	18	18	6	4
4	16	16	16	16	16	6
5	**16**	**16**	**16**	**16**	**16**	**4**

.

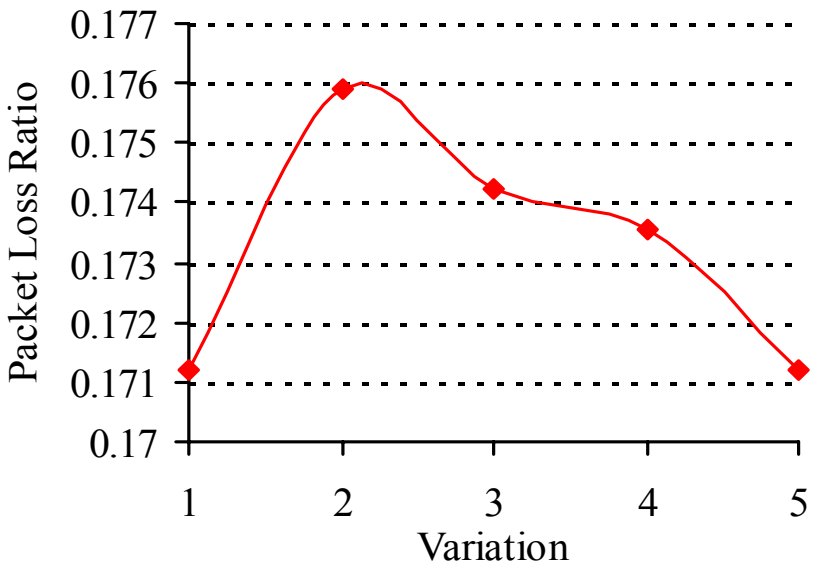


Figure 7.11**:** Packet loss ratio vs. Alpha Variation for DADT for the actual traffic load

For application 5, optimum value of alpha is 4. As we can see for variation two, packet loss ratio is greater than all other combinations. This is due to the fact that in

variation two value of alpha for application 5 is 6 which is high considering its packet size. Also for application 4, value of alpha is 6 which gives it less threshold and thus results in increase packet loss ratio. For variation 4, though value of alpha for application 5 is 6 but high value of alpha for application 4 results in overall less packet loss ratio.

7.4.3 Comparison of FSDA, DA and DADT for different loads

Figure 7.12 shows the performance of the three algorithms (FSDA, DA and DADT) for buffer size of 600 packets. Load has been varied from 0.5 to 0.9.

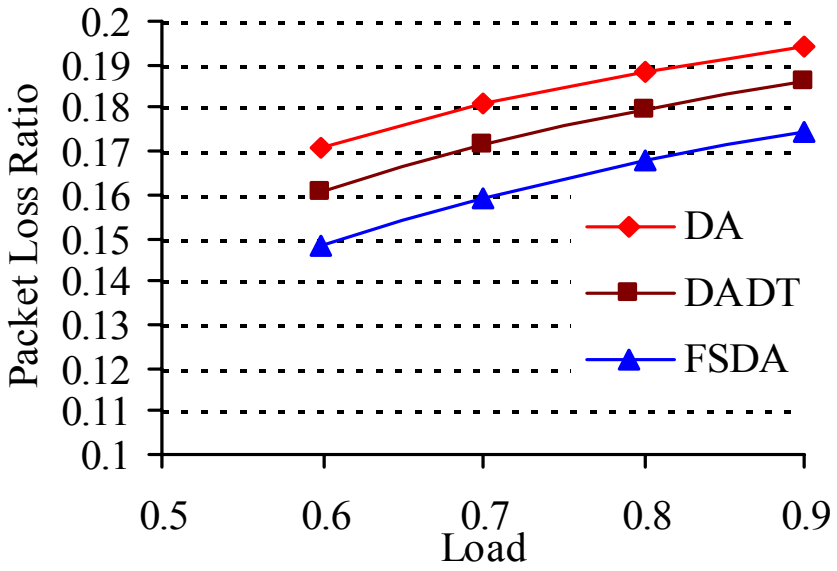


Figure 7.12**:** Packet loss ratio vs. Load for FSDA, DADT, and DA for the actual traffic load

7.4.4 *Comparison of FSDA, DA and DADT for different buffer size*

Figure 7.13 shows performance of three algorithms FSDA, DA, and DADT as the buffer size is varied from 500 to 800 packets. The performance of DA and DADT becomes very similar for buffer size =800 packets while FSDA has better packet loss ratio than DA and DADT.

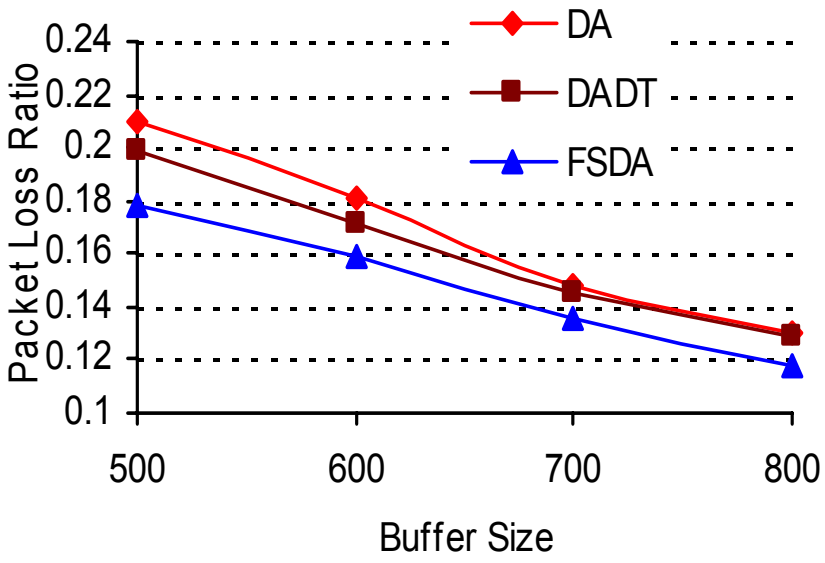


Figure 7.13**:** Packet loss ratio vs. Buffer size for FSDA, DADT, and DA for the actual traffic load.

7.4.5 *Improvement ratio of FSDA over DA and DADT*

Table 7.10 shows the improvement in packet loss ratio for 'FSDA over DA' and 'FSDA over DADT' according to different loads, from 0.5 to 0.9. For a load for '0.7', improvement ratio of FSDA over DA is 13.6% and over DADT is 7.5%.

Table 7.10

Improvement ratio of FSDA over DA and DADT for the actual traffic load.

Load	Improvement ratio (%) (FSDA/DA)	Improvement ratio (%) (FSDA/DADT)
0.5	12.5	5.2
0.6	13.5	6.6
0.7	13.6	7.5
0.8	26	20.1
0.9	12.9	8.1

7.5 FSDA Designed for UDP

In the TCP protocol, a source gets an acknowledgement from a receiver when a packet is accepted by a buffer management algorithm. On the other hand, in the UDP protocol, packets are not acknowledged by a receiver.

As explained in the previous section, in FSDA, packets are replaced when a buffer memory is full and an incoming packet is for an application whose flag is still '0.' For the replaced packet, a source will not get an information that the packet has been replaced, thus rejected, by a receiver. Hence, FSDA works more efficiently for the UDP/IP than the TCP/IP.

Table 7.11 shows the ratio of the number of replaced packets to the total number of incoming packets as load varies from 0.5 to 0.9 for the average traffic load and busty uniform model. Buffer size is taken as 600 packets.

Table 7.11

Ratio of the replaced packets in FSDA

Load	Total number of the replaced packets / Total Incoming packets
0.5	0.009822 (0.98%)
0.6	0.013807 (1.38%)
0.7	0.017489 (1.74%)
0.8	0.018585 (1.85%)
0.9	0.018808 (1.88%)

Though**,** the percentage of replaced packets is very low but in data critical applications even this low percentage of loss in undesirable. Hence, using FSDA for TCP-based applications may result in loss of data.

7.6 Simulation Results for EBDA

Three different network traffic loads are considered for our simulations and comparisons of algorithms: average network traffic load, heavy network traffic load, and actual network traffic load. We have used the "bursty uniform traffic model" for our simulations of all the network traffic loads since it is the most commonly used model.

Figure 7.14 shows the steps performed for performance comparisons of different algorithms

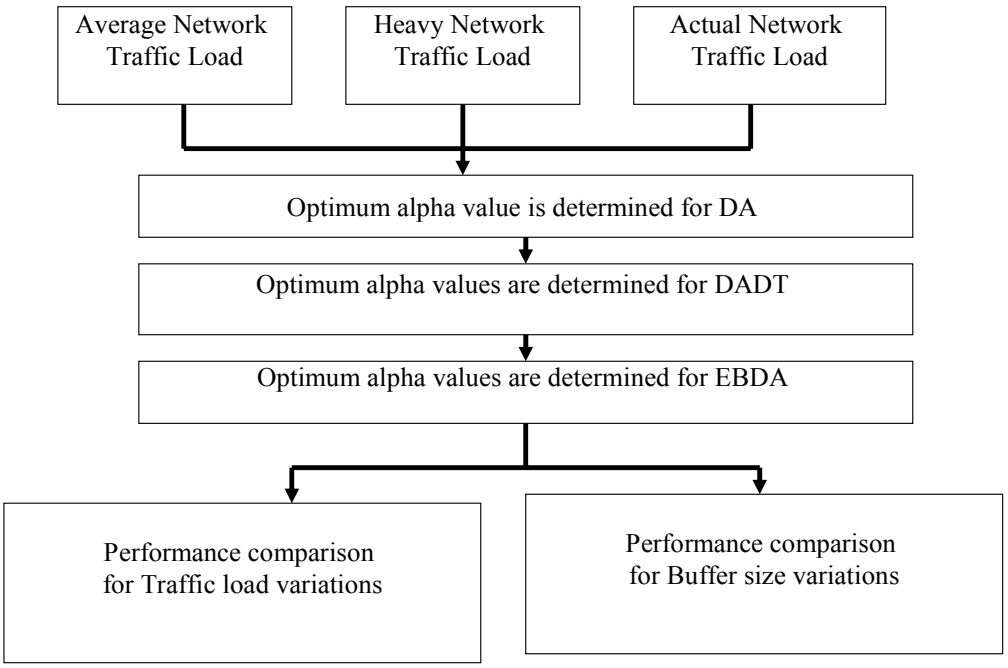


Figure 7.14: Steps performed for comparing DA and DADT with EBDA.

First, the optimum value of alpha is calculated for DA. Then, optimum values of alpha are calculated for DADT. Then, optimum values of alpha1, alpha2, gamma1 and gamma2 are determined for EBDA. This is followed by the performance comparison of DA, DADT, and EBDA when the load and the buffer size are varied. Finally, improvement ratio is determined for each network traffic load.

7.7 Simulation Results for Average Network Traffic load

Packet sizes for different applications based on the average network traffic load flow are given in table 7.1.

7.7.1 *Optimum Value of alpha for DA*

In Section 7.2.1 we have already calculated the optimum value of alpha for DA for average traffic load. The value of alpha comes out to be 16. So, we are going to use optimum value of alpha as 16 for DA.

7.7.2 *Optimum Value of alphas for DADT*

For DADT optimum values of alpha are calculated in section 7.2.2. The optimum values comes out to be 16 for queue0, 14 for queue1, 16 for queue2, 14 for queue3, 16 for queue 4 and 8 for queue5. We are going to use these alpha values for different applications.

7.7.3 *Optimum Value of alpha1, alpha2, gamma1 and gamma2 for EBDA*

Table 7.12 shows the packet losses for different variations of alpha and gamma values for EBDA, for a buffer size of 600 packets and load of 70% on each of queue. All

combinations that we have taken to find out values of alpha1, alpha2, gamma1 and gamma2 are powers of two. From Table 7.12, optimum values of (alpha1, gamma1, alpha2, and gamma2) come out to be for variation 1. For our comparison purpose, we will use values of alpha1, alpha2, gamma1, and gamma2 as 16, 4, 64, and 64 (from **variation 1**) respectively.

Table 7.12

Variation of (alpha1, alpha2, gamma1 and gamma2) vs. total packet loss for EBDA

Variation of alpha1, alpha2, gamma1, gamma2	**Total Packet Loss**
16,4,64,64	2353720
16,2,64,64	2356962
4,16,64,32	2360745
16,8,8,16	2422177
16,8,16,32	2376250
16,8,8,64	2385378
16,8,32,64	2361126
16,8,64,64	2353922

7.7.4 *Comparison of EBDA, DA and DADT for different loads*

Figure 7.15 shows the performance of the three algorithms (EBDA, DA, and DADT) for different loads. Load has been varied from 0.5 to 0.9. Buffer size is taken as 600 packets. As seen in Figure 7.15, EBDA has the least packet loss ratio for all of loads. The packet loss ratio increases for all the algorithms with increasing "load on the queues". Notice that the performance difference increases more at higher loads. As the load is increased, applications with larger packet size tend to increase their queue length to values greater than their threshold values frequently. Since, EBDA utilizes the buffer

space more efficiently, providing fairness to all the applications; EBDA can reduce the packet loss ratio significantly.

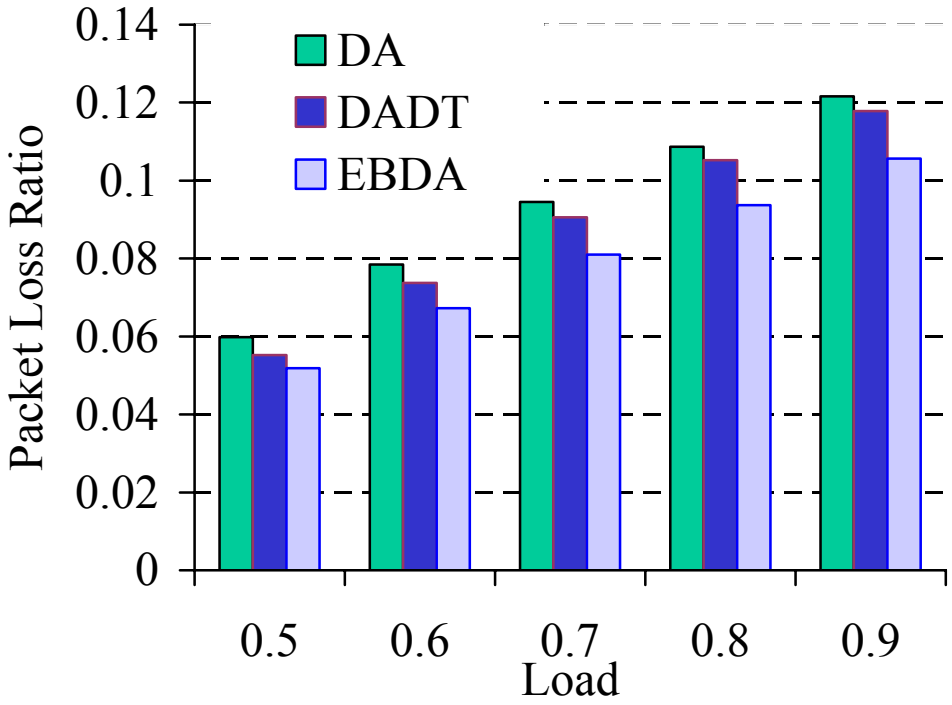


Figure 7.15**:** Packet loss ratio vs. Load for EBDA, DADT and DA for the average traffic load

7.7.5 *Comparison of EBDA, DA and DADT for different buffer size*

Figure 7.16 shows the performance of the three algorithms DA, DADT and EBDA as the buffer size varies from 500 to 800 packets. With an increase of buffer size, packet loss ratio decreases for all three algorithms. This is due to the fact that each queue gets more space to accommodate packets.

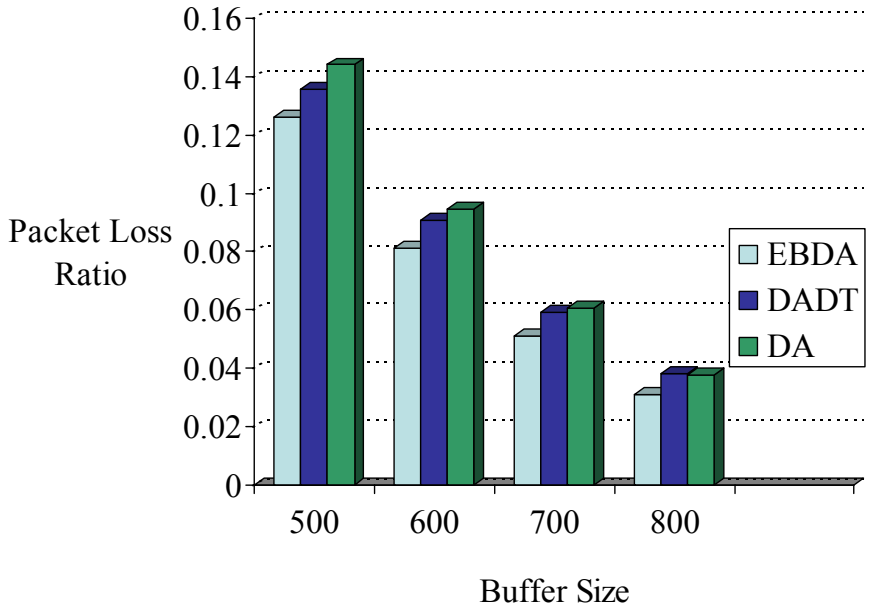


Figure 7.16**:** Packet loss ratio vs. Buffer Size for EBDA, DADT, and DA for the average traffic load

7.7.6 *Improvement ratio of EBDA over DA and DADT*

Table 7.13 shows the improvement in packet loss ratio for 'EBDA over DA' and 'EBDA over DADT' as the load varies from 0.5 to 0.9. For a load of 0.7 improvement ratio is 16.7 over DA and 11.8 over DADT. When compared with FSDA, EBDA has less improvement ratio. This is due to the fact that FSDA utilizes whole buffer memory. In FSDA, packets are accepted as long as there is enough space for them in the buffer memory while in EBDA, there is a controlling threshold. The advantage that EBDA has over FSDA is that there is no need to replace packets in EBDA. This means that EBDA can work more efficiently for applications that use TCP.

Table 7.13

Improvement ratio of EBDA over DA and DADT for the average traffic load

Load	**Improvement ratio (%) (EBDA /DA)**	**Improvement ratio (%) (EBDA /DADT)**
0.5	15.3	6.57
0.6	16.6	9.59
0.7	16.7	11.8
0.8	15.9	12.2
0.9	15.1	11.6

7.8 Simulation Results for Heavy Network Traffic load

Packet sizes for different applications based on the average network traffic load flow are given in table 7.4.

7.8.1 *Optimum Value of alpha for DA*

Optimum value of alpha for DA for heavy traffic load comes out to be 16. This is shown in section 7.3.1.

7.8.2 *Optimum Value of alphas for DADT*

From section 7.3.2, we can see that Optimum Value of alphas for DADT comes out be for variation 3 of table 7.5. The optimum alpha values are 14, 12, 14, 12, 12, and 8 for queue0 to queue5 respectively.

7.8.3 *Optimum Value of alpha1, alpha2, Gamma1 and Gamma2 for EBDA*

Table 7.14 shows the packet losses for different variations of alpha and gamma values for EBDA, for a buffer size of 600 packets, and a load of 70% on each of the

queue. From Table 7.14 optimum values of (alpha1, gamma1, alpha2, and gamma2) come out to be for variation 1.

Table 7.14

Variation of (alpha1, alpha2, gamma1 and gamma2) vs. total packet loss for EBDA

Variation of alpha1, alpha2, gamma1, gamma2	**Total Packet Loss**
16,4,64,64	1727801
16,2,64,64	1747305
4,16,64,32	1789091
4,16,32,32	1824368

7.8.4 *Comparison of EBDA, DA and DADT for different load*

Figure 7.17 shows the performance of the three algorithms (EBDA, DA, and DADT) for buffer size of 600 packets. Load has been varied from 0.5 to 0.9.

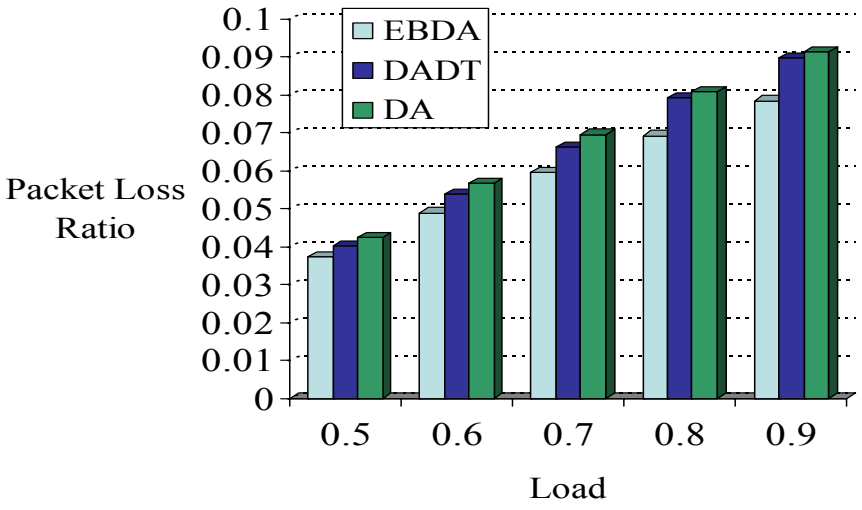


Figure 7.17: Packet loss ratio vs. Load for EBDA, DADT, and DA for the heavy traffic load

7.8.5 *Comparison of EBDA, DA and DADT for different buffer size*

Figure 7.18 shows the performance of the three algorithms, EBDA, DA, and DADT as the buffer size varies from 500 packets to 800 packets.

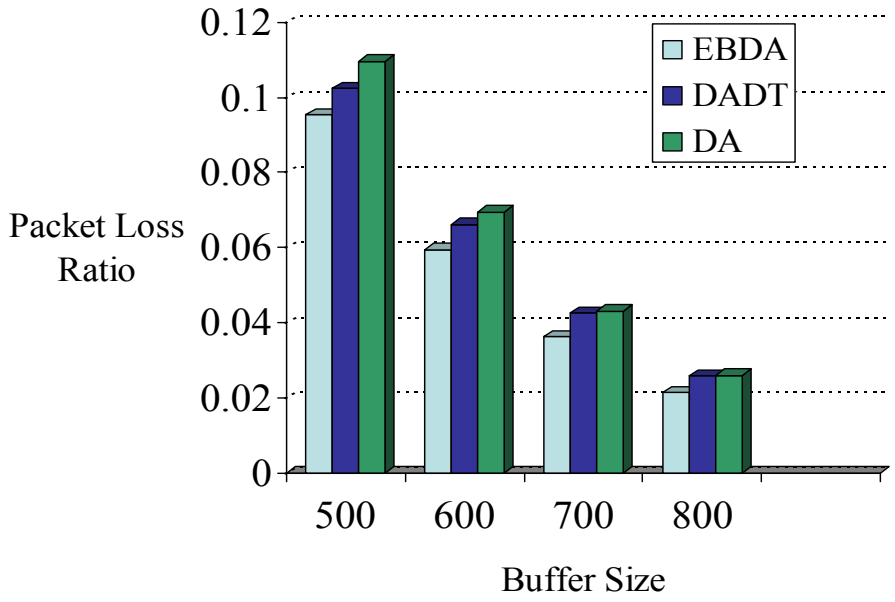


Figure 7.18**:** Packet loss ratio vs. Buffer Size for EBDA, DADT, and DA for the heavy traffic load

7.8.6 *Improvement ratio of EBDA over DA and DADT*

Table 7.15 shows the improvement in packet loss ratio for 'EBDA over DA' and 'EBDA over DADT' as the load varies from 0.5 to 0.9. For a load of 0.7 the improvement ratio is 16.8% over DA and 12.6% over DADT.

Table 7.15

Improvement ratio of EBDA over DA and DADT for the heavy traffic load

Load	Improvement ratio (%) (EBDA /DA)	Improvement ratio (%) (EBDA /DADT)
0.5	13.5	7.39
0.6	16.0	10.0
0.7	16.8	12.6
0.8	16.8	14.4
0.9	16.2	14.3

7.9 Simulation Results for Actual Network Traffic load

Packet sizes for different applications based on the actual network traffic load flow are given in table 7.8.

7.9.1 *Optimum Value of alpha for DA*

Optimum value of alpha for DA for actual traffic load comes out to be 4. This is shown in section 7.4.1.

7.9.2 *Optimum Value of alphas for DADT*

In section 7.4.2 we have already determined the optimum values of alpha for DADT for actual traffic load. The optimum values come out to be for variation 5 in table 7.8. The values are 16, 16, 16, 16, 16, and 4 for queue0 to queue5 respectively.

7.9.3 *Optimum Value of alpha1, alpha2, gamma1 and gamma2 for EBDA*

Table 7.16 shows the packet losses for different variations of alpha and gamma values for EBDA, for a buffer size of 600 packets, and a load of 70% on each of the

queue. From Table 7.16 optimum values of (alpha1, gamma1, alpha2, and gamma2) come out to be for variation 1.

Table 7.16

Variation of (alpha1, alpha2, gamma1, and gamma2) vs. total packet loss for EBDA

Variation of alpha1, alpha2, gamma1, gamma2	**Total Packet Loss**
16,4,64,64	4880774
8,4,16,16	4957822
8,4,32,32	4890188
8,4,32,64	4890423
8,4,64,64	4886391

7.9.4 *Comparison of EBDA, DA and DADT for different load*

Figure 7.19 shows the performance of the three algorithms (EBDA, DA and DADT) for different loads. Load has been varied from 0.5 to 0.9. Buffer size has been taken 600 packets.

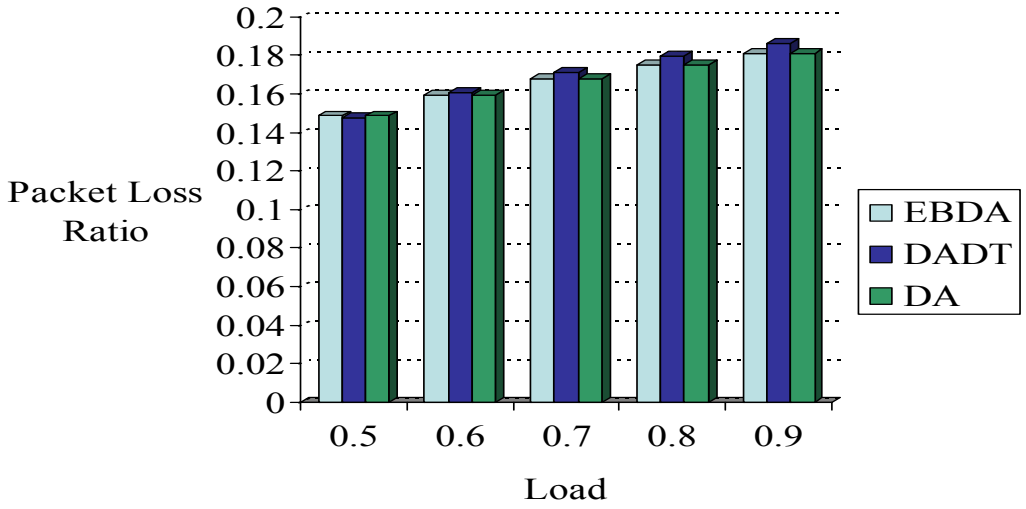


Figure 7.19**:** Packet loss ratio vs. Load for EBDA, DADT, and DA for the actual traffic load

7.9.5 *Comparison of EBDA, DA and DADT for different buffer size*

Figure 7.20 shows performance of three algorithms EBDA, DA, and DADT as the buffer size varies from 500 to 800 packets. As the size of the buffer goes up, the packet loss ratio decreases at a fast rate.

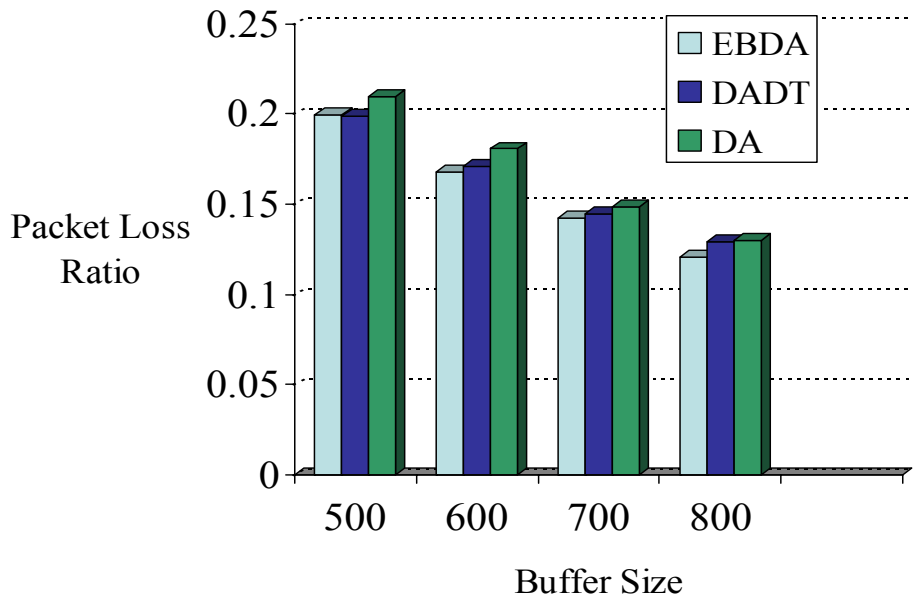


Figure 7.20**:** Packet loss ratio vs. Buffer Size for EBDA, DADT, and DA for the actual traffic load

7.9.6 *Improvement ratio of EBDA over DA and DADT*

Table 7.17 shows the improvement in packet loss ratio for 'EBDA over DA' and 'EBDA over DADT' according to different loads, from 0.5 to 0.9.

Table 7.17

Improvement ratio of EBDA over DA and DADT for the actual traffic load

Load	**Improvement ratio (%) (EBDA /DA)**	**Improvement ratio (%) (EBDA /DADT)**
0.5	0.072406	0.002982
0.6	0.072734	0.007699
0.7	0.076505	0.019389
0.8	0.076193	0.02627
0.9	0.074594	0.029234

CHAPTER VIII

CONCLUSION AND FUTURE WORK

8.1 Summary of the Achievements

In this thesis, we developed two buffer management algorithms: Fairly Shared Dynamic Algorithm and Evenly Based Dynamic Algorithm to reduce packet losses in Network terminals and also to distribute packet losses among different applications more evenly. We also implemented four buffer management algorithms:

1) Dynamic Algorithm (DA): In DA, packets for any application are accepted as long as the queue length for the application is less than its threshold value. The threshold value depends on amount of the unused buffer space. It does not take packets sizes into consideration.
2) Dynamic Algorithm with Dynamic Threshold (DADT): It works like DA except for the fact that it takes packet sizes into consideration.
3) Fairly Shared Dynamic Algorithm (FSDA): FSDA utilizes whole buffer memory. If single application is active it works like CS. If multiple applications are there then it works like DA except for the fact that packets are not rejected unless the buffer is completely filled. Flags are used to maintain fairness among applications. Packets can be replaced in FSDA.

4) Evenly Based Dynamic Algorithm (EBDA): EBDA reduces packet losses by taking packet size into summation instead of multiplication. This helps in maintaining fairness among all applications.

The performances of Fairly Shared Dynamic Algorithm and Evenly Based Dynamic Algorithm are compared with all other algorithms.

8.1.1 FSDA

Fairly Shared Dynamic Algorithm utilizes full memory. Packets are accepted as long as there is enough space for them in the packet buffer. Flags are maintained to keep track of applications which have taken more space than what they would have taken in Dynamic algorithm. Flags help in maintaining fairness among different applications. FSDA works very efficiently for applications that use UDP, since information about replaced packets is lost in FSDA. For the replaced packet, a source will not get an information that the packet has been replaced, thus rejected, by a receiver. Hence, FSDA works more efficiently for the UDP/IP than the TCP/IP. If used for TCP/IP, FSDA may result in loss of some data.

The simulations considered a buffer of size as 600 cells (packets), 6 output queues (0-5), bursty uniform traffic model, dequeue time (packet processing time) of 14 clock cycles for a burst of 10 cells and uniform load for all output queues. For our simulation model, three traffic mixes were considered.

8.1.2 Improvement in efficiency for FSDA

For the average network traffic load, the FSDA improves the packet loss ratio by 18.5% over the dynamic algorithm and by 13.5% over the DADT. For the heavy network traffic load, the FSDA improves the packet loss ratio by 16.8% over the dynamic algorithm and by 12.5% over the DADT. While for the actual traffic load the improvement is 13.6% over DA and 7.5% over DADT.

8.1.3 EBDA

EBDA takes fairness among applications and packet sizes of applications into consideration while allocating buffer space to each application. The idea behind EBDA is to eliminate the need to calculate optimum alpha value for each application as in DADT and also distribute the packet losses more evenly among the different applications.

8.1.4 Improvement in efficiency for EBDA

For the average network traffic load, the EBDA improves the packet loss ratio by 16.7% over the dynamic algorithm and by 11.8% over the DADT. For the heavy network traffic load, the EBDA improves the packet loss ratio by 16.8% over the dynamic algorithm and by 12.6% over the DADT. While for the actual traffic load the improvement is 7.6% over DA and 1.9% over DADT.

8.2 Future Work

One of the areas of future research could be to incorporate priority applications in the buffer management algorithms. Packets with higher priority should be give preference over packets with lower priority.

REFERENCES

[1] A. Tanenbaum, *Computer Networks*, 4th ed., Prentice Hall, 2002.

[2] T. Henriksson, U. Nordqvist, D. Liu, "Embedded Protocol Processor for fast and efficient packet reception", *IEEE Proceedings on Computer Design: VLSI in Computers and Processors*, vol. 2, pp. 414-419, September 2002.

[3] V. Paxson, "End-to-End internet packet dynamics", *Proceedings of ACM SIG-COM*, vol. 27, pp. 13-52, October 1997.

[4] Tomas Henriksson, "Intra-Packet Data-Flow Protocol Processor", *PhD Dissertation*, Linkopings universitet, 2003.

[5] U. Nordqvist, D. Liu, "Power optimized packet buffering in a protocol processor", *Proceedings of the 2003 10th IEEE International Conference on Electronics, Circuits and Systems*, vol. 3, pp. 1026-1029, December 2003.

[6] M. Arpaci, J.A. Copeland, "Buffer Management for Shared Memory ATM Switches", *IEEE Communication Surveys*, First Quarter 2000.

[7] T. Henriksson, U. Nordqvist, D. Liu, "Specification of a configurable general-purpose protocol processor", *IEEE Proceedings on Circuits, Devices and Systems*, vol. 149, issue: 3, pp. 198-202, June 2002.

[8] A. Tobagi, "Fast Packet Switch Architectures for Broadband Integrated Services Digial Networks", *Proceedings of IEEE*, vol. 78, pp. 133-167, January 1990.

[9] M. Irland, "Buffer Management in a Packet Switch", *IEEE Transactions on Communications*, COM-26, no. 3, pp. 328-337, March 1978.

[10] G. J. Foschini, B. Gopinath, "Sharing Memory Optimally", *IEEE Transactions on Communications*, vol. COM-31, no. 3, pp. 352-360, March 1983.

[11] F. Kamoun, L. Kleinrock, "Analysis of Shared Finite Storage in a Computer Network Node Environment under General Traffic Conditions", *IEEE Transactions on Communications*, vol., COM-28, pp. 992-1003, July 1980.

[12] S. X. Wei, E.J. Coyle, M.T. Hsiao, "An Optimal Buffer Management Policy for High-Performance Packet Switching", *Proceedings of IEEE GLOBECOM'91*, vol. 2, pp. 924-928, December 1991.

[13] A. K. Thareja, A.K. Agarwal, "On the Design of Optimal Policy for Sharing Finite Buffers", *IEEE Transactions on Communications*, vol. COM—32, no. 6, pp 737-780, June 1984.

[14] A. K. Choudhury, E.L. Hahne, "Dynamic Queue Length Thresholds for Shared-Memory Packet Switches", *IEEE/ACM Transactions on Communications*, vol. 6, no. 2, pp. 130-140, April 1998.

[15] Sundar Iyer, " *SIM: A Fixed Length Packet Simulator*", http://klamath.stanford.edu/tools/SIM

[16] Yul Chu, Vinod Rajan "An Enhanced Dynamic Packet Buffer Management", In the proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC'05), Cartagena, Spain, June 2005

[17]Cisco Systems: http://www.cisco.com/warp/public/473/lan-switch-cisco.shtml.*(Accessed : 2nd March ,2006)*

[18] S. McCreary and K. Claffy, "*Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange,"* In ITC Specialist Seminar on IP Traffic Measurement, Modeling, and Management, Manterey, California, September 2000.

[19] Sam Manthorpe: http://lrcwww.epfl.ch/people/sam/research_protlevels.html. *(Accessed : 2nd October ,2005).*

[20] Intel Corporation : http://www.intel.com/education/highered/Networking/lectures/lesson11.ppt#282,27,Summary *(Accessed : 2nd December ,2006).*

[21] http://en.wikipedia.org/wiki/Network_card. *(Accessed : 17th March 2007).*

[22] http://www.cs.utah.edu/~swalton/Documents/Articles/Multicasting-1.pdf *(Accessed: 10th Sptember 2007).*

[23] http://nile.wpi.edu/NS/ *(Accessed: 10th Sptember 2007).*