IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.7A, July 2006

An Efficient Buffer Management in a Network Interface Card

Amit Uppal,[†] *and Yul Chu*[††]*,*

Mississippi State University, Mississippi State, MS, USA

Summary
This paper proposes a dynamic packet buffer management algorithm for a protocol processor in a network terminal. The protocol processor is to handle high-speed data streams, more than 10 Gb/s, in a network interface card (NIC). There are two types of packet buffer management algorithms, static and dynamic. In general, the dynamic buffer management algorithms work better than the static ones for reducing the packet loss ratio. However, conventional dynamic buffer management algorithms do not utilize packet buffer memory efficiently. Therefore, we propose an algorithm, which contributes to fairness and full utilization of the packet buffer memory. Our experimental results show that the proposed algorithm improves the packet loss ratio by 13.5 % to 18.5% compared to conventional dynamic algorithms.

Key words:
Packet buffer management, network interface card, protocol processor, high-speed computer network

1. Introduction

Data is transmitted from one application to another in the form of packets in a computer network [1]. A packet consists of 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. After processing of layer 3 and layer 4 protocols, packets are placed in a packet buffer in a network interface card (NIC), which connects a computer to an Ethernet network.

A packet buffer is a large shared dual-ported memory [6]. Packets for each application are multiplexed into a single stream. In Figure 1, packet buffer management algorithm determines whether to accept or reject each packet. The accepted packet is placed into a logical FIFO 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.

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 a NIC determine how the buffer space is distributed among different applications.

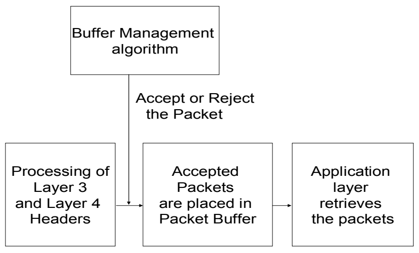
Buffer Management algorithm
Accept or Reject the Packet
Processing of Layer 3 and Layer 4 Headers
Accepted Packets are placed in Packet Buffer
Application layer retrieves the packets

Fig. 1 Role of buffer management algorithm in a NIC.

Buffer space can be managed using either a static threshold scheme or a dynamic threshold scheme for 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.

The design of a buffer management algorithm needs to consider the following two 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]; and 2)

Manuscript received July 7, 2006.
Manuscript revised . July 25, 2006

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

In this paper, we will present four popular buffer management algorithms for a NIC: Completely Partitioned algorithm (CP), Completely Shared algorithm (CS) [11], Dynamic algorithm (DA) [14], and Dynamic Algorithm with Dynamic Threshold (DADT) [16].

None of these algorithms except CS makes full utilization of a buffer space; in other words, packets can be rejected even if there is some unused buffer space for those algorithms. This happens because of a controlling threshold value of each queue. However, CS has a disadvantage that any application can fill the whole buffer space. This will increase packet losses for the other applications. That means CS cannot provide fairness to applications even though CS utilizes full buffer space. Therefore, it is required to develop an algorithm for fairness and efficient memory utilization to reduce packet losses.

In this paper, we propose an efficient buffer management algorithm called Fairly Shared Dynamic algorithm (FSDA); FSDA makes full utilization of memory and provide fairness to all the applications.

The remainder of the paper is divided into eight sections. Section 2 describes the traditional and new architecture for packet reception; Section 3 discusses four popular buffer management algorithms; Section 4 introduces the proposed algorithm, FSDA; Section 5 explains our simulation model to measure the performance of buffer management algorithms; Section 6 compares the performance of FSDA with DA and DADT for different traffic mix; finally, Section 8 gives the conclusions.

2. Brief Description of Protocol Processor

Traditionally, the physical layer and MAC layer processing, layer 1 and 2, has been done in a NIC [2]. After processing in a NIC, the packet is transferred to the main memory of a host processor, general-purpose processor of a network terminal. Then, the TCP/IP header or UDP/IP header is processed by a host operating system (OS). In general, 20%-60% of the processing power of the OS has been used for the protocol handling [2]. Therefore, the traditional NIC cannot handle packets efficiently for a high-speed network, more than 10 Gb/sec [2].

2.1 New Architecture for Packet Reception

Tomas Henriksson, et al. [2][7] proposed protocol processor architecture to offload the host processor for a high-speed network. The new packet reception shown in Figure 2, the supporting micro-controller and protocol processor, moves the layer 3 and layer 4 processing to a NIC [1][5][7]; as usual, packets coming in from the network are received in a NIC and are processed for the layer 1-2 protocols; instead of sending the packet to a host processor for further processing, the protocol processor in a NIC handles the processing of the layer 3 and layer 4 protocols. The main goal of the protocol processor is to handle the TCP/IP or the UDP/IP processing at a wire speed [2][7].

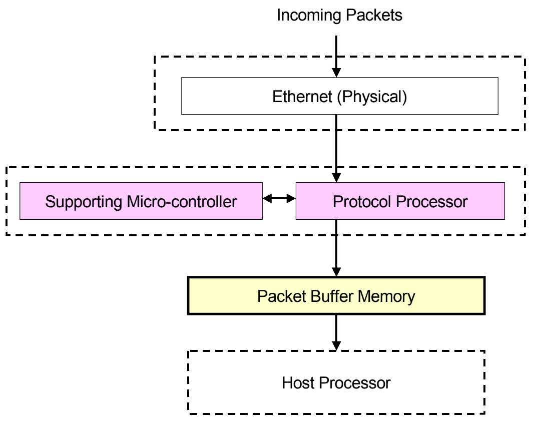

Fig. 2 Protocol processor architecture [2].

As shown in Figure 2, incoming packets will stream through the protocol processor and the payload (application) data will be stored in a packet buffer until a host application retrieves it [5]. Packets are classified based on an application. Once a packet is classified, it is stored in an output queue of a buffer**.** The processor is estimated to be able to handle 10 Gb/s data streams [2]. The packet loss ratio at this speed comes out to be around 9-10 percent for Dynamic algorithms [14,16]. So, it is necessary to have a new algorithm, which can reduce the overall packet loss ratio and provide fairness to all the applications.

Buffer management algorithms in a NIC must be adaptive and intelligent to any changes in traffic conditions. These algorithms are different from what we require in a switches 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 for the destination node. Hence, the role of a switch is to store and forward a packet to a correct destination [17]. However, in a 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. Buffer Management Algorithms for a NIC

Various buffer management algorithms have been proposed so far [8-10] [12-13] to reduce packet loss ratio. This section describes four popular algorithms: Completely Partitioned (CP), Completely Shared (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.

3.1. Static Thresholds

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:

$$k_1 + k_2 + \dots + k_n = M \qquad (1)$$

$$\sum_{i=1}^{N} k_i = M \qquad (2)$$

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 a buffer space, then it can reject the incoming packets even though there is some space left in the buffer.

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:

$$k_i = M,\ i = 1, 2, \dots, N \qquad (3)$$

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. 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.

3.2. Dynamic Thresholds

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 algorithms achieve 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 its threshold value. 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

$$T(t) = \alpha.\ (M - Q(t)) \qquad (4)$$

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.

The DADT [16] works like DA. In this algorithm, the alpha (α) value is different from 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

$$T(t) = \alpha_i.\ (M - Q(t)) \qquad (5)$$

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, packets are rejected even if there is some space left in a buffer. Also, it is difficult to determine the optimum alpha value for each application.

3.3. Performance Comparison

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 scheme with the dynamic threshold schemes, DA and DADT.

4. Proposed Dynamic Algorithm: FSDA

As we discussed, CS utilizes a buffer memory at full. The algorithm, however, is not fair to all the applications. On the other hand, DA and DADT are fair to all the applications, but they do not utilize a buffer space at full. Therefore, we proposed FSDA (Fairly Shared Dynamic algorithm) that will satisfy two factors: 1) fairness to all the applications; and 2) full utilization of a buffer space. To achieve this, FSDA will maintain 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.

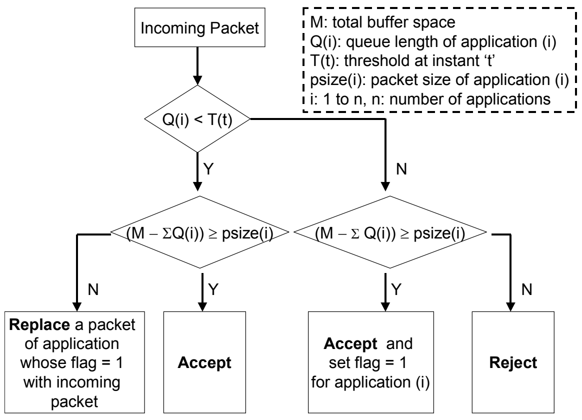
Incoming Packet
M: total buffer space
Q(i): queue length of application (i)
T(t): threshold at instant 't'
psize(i): packet size of application (i)
i: 1 to n, n: number of applications
Q(i) < T(t)
Y
N
(M – ΣQ(i)) ≥ psize(i)
(M – Σ Q(i)) ≥ psize(i)
N
Y
Y
N
Replace a packet of application whose flag = 1 with incoming packet
Accept
Accept and set flag = 1 for application (i)
Reject

Fig. 3 Flowchart of FSDA

Figure 3 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.

Say at any instant 't', we have queue lengths (in bytes) as 24 and 16 for *application one* and *application two*, respectively. 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 as '1'.

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. 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); and 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*. Like 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'.

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 the 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 the application is less than the threshold value, which is calculated by using *counter one*.

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, giving fairness to all the applications. In addition, FSDA utilizes a 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 alpha value as a power of 2, which makes its hardware implementation easier. This gives it a distinct advantage over DADT. According to our simulation results, by utilizing the whole buffer space, FSDA can decrease the total packet losses efficiently.

4.1 FSDA for UDP

In the TCP protocol, a source used to get 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, rejected, by a receiver. Hence, FSDA works more efficiently for the UDP/IP than the TCP/IP.
Table 1 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 (refer to Section 5).

Table 1: 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%)

Table 1 shows that the number of replaced packets is much less than the total incoming packets; therefore, it might be possible to ignore the number of the replaced packets. However, for doing the performance analysis in Section 6, the replaced packets have been taken into consideration for FSDA.

5. Simulation Model

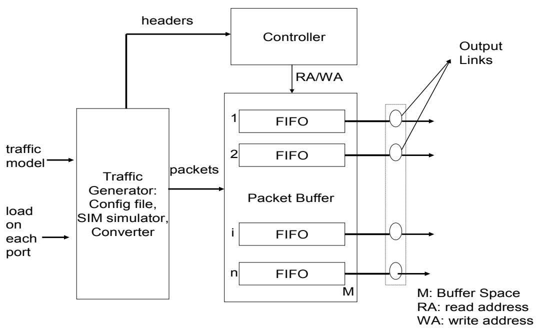

Fig. 4 Simulation model for the packet buffer

We developed a simulation model for a packet buffer by using VHDL as shown in Figure 4. In Figure 4, the *Traffic Generator* block produces packets according to the specifications provided in the configuration file (config file). The config file specifies the traffic model and “load on each port” [6].

There are three kinds of traffic model that are available [16]. These are:

- 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: Incoming packets are in the form of Bernoulli arrivals and distributions on all output ports

“load on each port (ρ)” is determined by the ratio of the number of packets in the busy-idle periods [14] and is given by the equation:

$$\rho = \frac{L_b}{L_b + L_{idle}} \qquad (6)$$

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

The Traffic Generator produces packets with a mean inter-arrival time and a mean burst length [6]. The “SIM” simulator in [15] is used for producing a trace of packets. The trace of packets from the “SIM” simulator is written to some output file through converter.

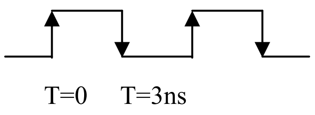

Fig. 5 Clock used for the Simulation Model

Figure 5 shows how the simulation model works as a clock-base. In Figure 5, the packet from output file is read at every positive edge of the clock. Then, the buffer management algorithm in the controller determines whether to accept or reject a packet. If the packet is accepted, the controller specifies the write address (WA) based on the output queue for which the packet is destined. The packet is written into the memory at the negative edge of the clock. In addition, the queue length for that application is incremented. This also initiates the dequeue process for the packet. Dequeue time has been taken as a Poisson random variable with a fixed mean [5].

6. Simulation Results and Analysis

Three different network traffic loads are considered for our simulations: 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 [19, 16].

For each traffic load, first, the optimum alpha value is determined for DA. After then, 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. Section 6.1, 6.2 and 6.3 discusses our simulation results for the average network traffic loads, heavy network traffic load, and actual network traffic load, respectively.

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.

6.1 Average Traffic

We implemented a traffic mix with the average network traffic loads according to [5]. First, we determined the optimum 'α' (alpha) value for DA. Optimum alpha is considered as the alpha value for which DA gives the minimum packet loss ratio.

Table 2 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 2: Queue properties for average traffic load

	Q0	Q1	Q2	Q3	Q4	Q5
Size in bytes	256	64	256	32	128	512
packet unit # (32 bytes/unit)	8	2	8	1	4	16

Figure 6 shows the packet loss ratio for DA as the alpha value is varied from 4 to 20. In Figure 6, the size of the buffer is 600 packets, and "load on each queue" is 70%.

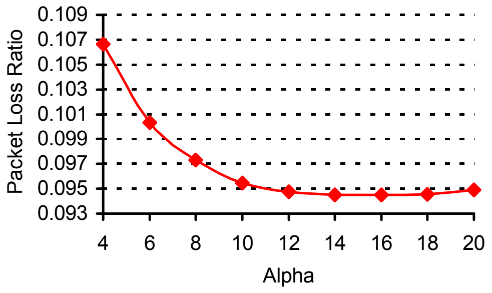

Fig. 6 Packet loss ratio vs. Alpha for DA for the average traffic load

From Figure 6, 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.

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 3 shows the different combinations of alpha and Figure 7 shows the packet loss ratio corresponding to them.

In Figure 7, the buffer size is 600 packets, and "load on each queue" is 70%. Figure 7 shows that the optimum combination of alpha comes out of the variation 5.

For our comparison purpose, we will use 'alpha=16' for DA and the variation 5 (from table 3) as alpha values for DADT.

Table3: 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

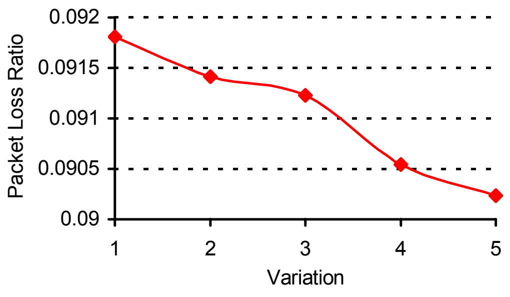

Fig. 7 Packet loss ratio vs. variations of alpha for DADT for average traffic load

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 proposed algorithm easier.

Figure 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. As seen in Figure 8, 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.

Figure 9 shows the performance of the three algorithms FSDA, DA, and DADT as the buffer size is varied from 500 to 800 packets. 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.

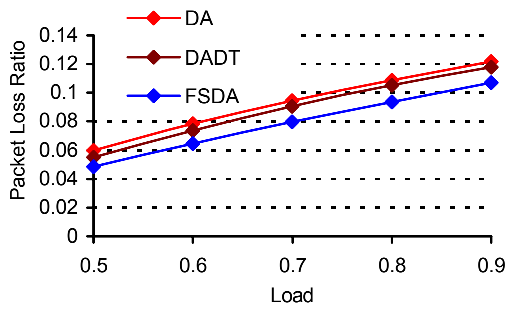

Fig. 8 Packet loss ratio vs. Load for FSDA, DADT, DA for the average traffic load

Table 4 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.

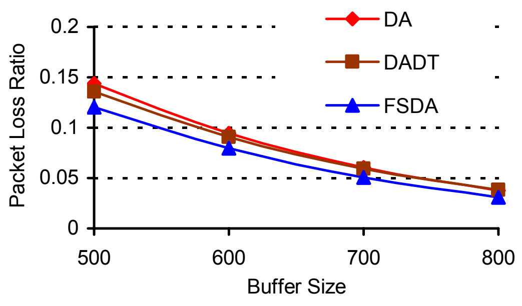

Fig. 9 Packet loss ratio vs. Buffer Size for FSDA, DADT and DA for the average traffic load

Table 4: 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

6.2 Heavy Network Traffic

Table 5 shows the packet sizes of different applications in bytes based on the heavy network traffic load in [5].

Table 5: Queue properties for the heavy traffic load

	Q0	Q1	Q2	Q3	Q4	Q5
Size in bytes	128	64	128	32	256	512
packet unit # (32 bytes/unit)	4	2	4	1	8	16

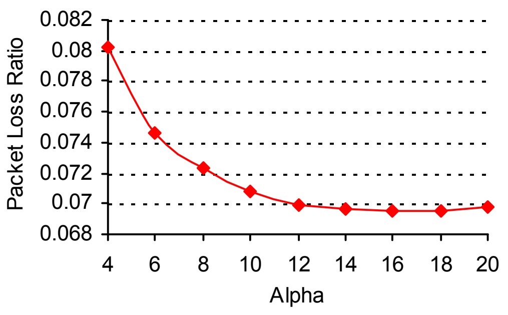

Fig. 10 Packet loss ratio vs. Alpha for DA for the heavy traffic load

Figure 10 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 10, 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.

Now we will determine the optimum values of alpha for DADT. Table 6 shows the different combinations of alpha and Figure 11 shows the packet loss ratio corresponding to them. Figure 11 shows that the optimum combination of alpha comes out of the variation 3.

Table6: 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

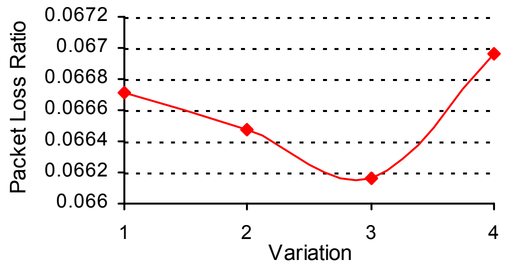

Fig. 11 Packet loss ratio vs. Variation for DADT for the heavy traffic load

Figure 12 shows the performance of the three algorithms (FSDA, DA and DADT) for different loads. Load has been varied from 0.5 to 0.9.

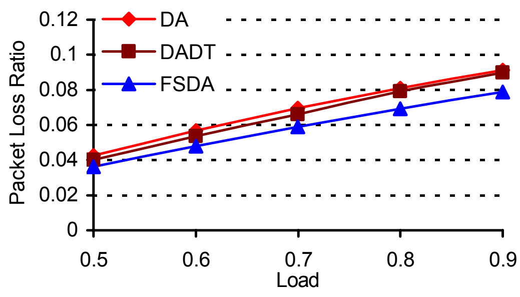

Fig. 12 Packet loss ratio vs. Load for FSDA, DADT, DA for the heavy traffic load

Figure 13 shows the performance of the three algorithms, FSDA, DA, and DADT as the buffer size is varied.

Table 7 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.

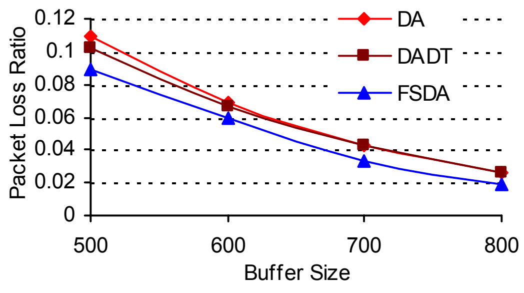

Fig. 13 Packet loss ratio vs. Buffer Size for FSDA, DADT and DA for the heavy traffic load

Table 7: Improvement ratio of FSDA over DA and DADT for the heavy network 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

6.3 Actual Network Traffic

Table 8 shows the packet sizes of the different applications in bytes based on the actual network traffic load flow in [18].

Table 8: Queue properties for the actual traffic load

	Q0	Q1	Q2	Q3	Q4	Q5
Size in bytes	32	32	32	64	512	1472
packet unit # (32 bytes/unit)	1	1	1	2	16	46

Figure 14 shows the packet loss ratio for DA as the alpha value is varied from 4 to 20 for the actual network traffic load. Figure 14 shows that the optimum alpha value for DA comes out to be 4.

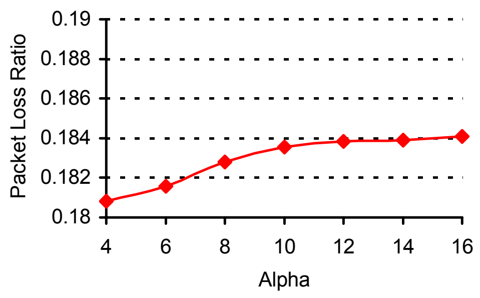

Fig. 14 Packet loss ratio vs. Alpha for DA for the actual traffic load

For DADT, first we determined the optimum 'α' (alpha) values. Table 9 shows the different combinations of alpha and Figure 15 shows the packet loss ratio corresponding to them. Figure 15 shows the optimum alpha values come out of the variation 5.

Table9: 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

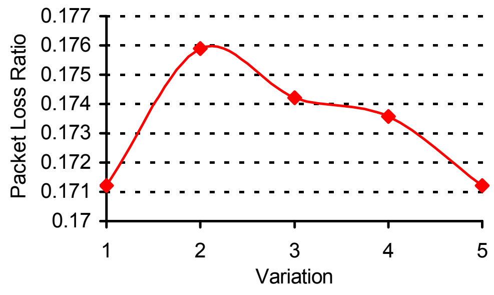

Fig. 15 Packet loss ratio vs. Variation for DADT for the actual traffic load

Figure 16 shows the performance of the three algorithms (FSDA, DA and DADT) for different loads. Load has been varied from 0.5 to 0.9.

Figure 17 shows performance of three algorithms FSDA, DA, and DADT as the buffer size is varied.

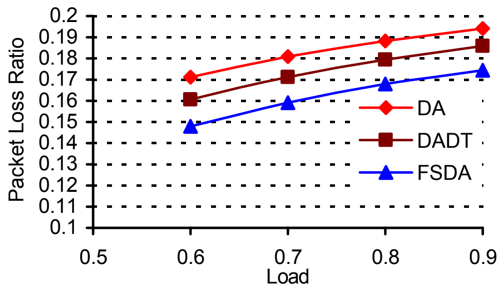

Fig. 16 Packet loss ratio vs. Load for FSDA, DADT, DA for actual traffic load

Table 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.

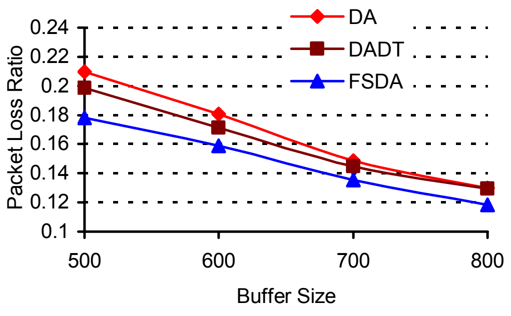

Fig. 17 Packet loss ratio vs. Buffer Size for FSDA, DADT and DA for the actual traffic load

Table10: Improvement ratio of FSDA over DA and DADT for the actual network 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. Conclusions

This paper proposes the Fairly Shared Dynamic Algorithm (FSDA) to reduce the number of packets being dropped at the packet buffer.

A buffer management algorithm will decide the amount of space for each output queue in the packet buffer. Three buffer management algorithms are implemented for our simulations: 1) Dynamic algorithm (DA); 2) Dynamic Algorithm with Dynamic Threshold (DADT); and 3) Fairly Shared Dynamic algorithm (FSDA). FSDA utilizes the whole buffer space, which makes it different from DA and DADT. FSDA also keeps track of those applications, which take more space in the buffer than their threshold values. Flags help in maintaining fairness to all the applications. With the utilization of the entire buffer space, FSDA can reduce the number of packets dropped efficiently.

The simulations considered 6 output queues (0-5), bursty uniform traffic model, dequeue time of 14 clock cycles for a burst of 10 packets, and uniform load for all the output queues. For the traffic mix with the average network traffic loads [5], the FSDA improves the packet loss ratio by 18.5% as compared with DA, and by 13.5% as compared with DADT. For the heavy network traffic [5], improvement in packet loss ratio is 16.8% over DA and

12.5% over DADT, and for the actual traffic [18] improvement in packet loss ratio is 13.6% over DA and 7.5% over DADT.

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] Vinod Rajan “An enhanced dynamic algorithm for packet buffer”, Master thesis, Mississippi State University, 2004.
[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)*

Amit Uppal received the B.E. degree in Electronics and Engineering from Thapar Institute of Engineering and Technology India in 2003 and presently pursuing M.S. degrees in Electrical Engineering from Mississippi State University. During 2003-2004, he worked as Assistant System Engineer in Tata Consultancy Services (TCS), India.

Yul Chu is an Assistant Professor in the Department of Electrical and Computer Engineering at Mississippi State University. He received his Ph.D. in Electrical and Computer Engineering from University of British Columbia, Canada in 2001, MS in Electrical engineering from Washington State University in 1995, and Bachelor in applied electronics from KwangWoon University, Seoul, Korea in 1984. His current research interests lie in the area of high performance computer architecture, low-power embedded systems, wireless-terminal architecture, Telematics, parallel processing, cluster and high-available architectures, etc.