2007

Packet Buffer Management for a High-Speed Network Interface Card

Shalini Batra

Yul Chu
The University of Texas Rio Grande Valley

Yan Bai

Recommended Citation

S. Batra, Y. Chu and Y. Bai, "Packet Buffer Management for a High-Speed Network Interface Card," 2007 16th International Conference on Computer Communications and Networks, Honolulu, HI, 2007, pp. 191-196, doi: 10.1109/ICCCN.2007.4317818.

Packet Buffer Management for a High-Speed Network Interface Card

Shalini Batra and Yul Chu
Dept. of Electrical and Computer Engineering
Mississippi State University
Mississippi State, MS, USA
{sb621, chu}@ece.msstate.edu

Yan Bai
TSYS Dept. of Computer Science
Columbus State University
Columbus, GA, USA
bai_yan@colstate.edu

***Abstract*— Packet buffers in a smart network interface card are managed in a way to reduce any packet losses from high-speed burst incoming data. Currently, two types of packet buffer management techniques are used. They are static buffer management and dynamic buffer management techniques. Dynamic buffer management techniques are more efficient than the static ones because they change the threshold value according to network traffic conditions. However, current dynamic techniques cannot adjust threshold instantaneously. Thus, packet losses with dynamic techniques are still high. We, therefore, propose a history- based buffer management scheme to address the issue. Our experiment results show that the history-based scheme reduces packet loss by 11% to 15.9% as compared to other conventional dynamic algorithms.**

Keywords - packet buffer; network interface card; layer 3 and 4 protocols; VHDL; static and dynamic buffer management.

I. INTRODUCTION

In computer networks, data from one application is wrapped into a packet and transmitted to another application. A packet consists of application data and packet headers. In a traditional communication model, a client sends a request for data to a server. The server responds the request and sends the data to the client. The client and the server acknowledge each other. The client processes the data and places it in the packets, and the packets are stored in the buffer of Network Interface Card (NIC), until the host application retrieves the packet, after a time popularly known as 'dequeue time' [1-3].

Packet buffer is a large shared dual-ported memory [4]. Packets for each application are multiplexed into a single stream. Packet buffer management algorithm determines whether to accept or reject a packet. The accepted packets are then placed into logical FIFO queues; each application has its own queue in a packet buffer [2][5]. The accepted packet remains in the buffer until the application retrieves it from the buffer. Once the buffer gets full, newly arrived packets will be rejected.

Figure 1 shows a packet buffer. In Figure 1, each application uses an output port (e.g., Application 1 uses port 1, Application 2 uses port 2, and so on). Port 1 has a space for buffering four packets and two packets are already buffered in the space; so, port 1 can only accept two additional packets for application 1. For the port 4, it has a space for 5 packets and all the packets are buffered; therefore, if a packet for the application 4 comes, it will be dropped since no buffer space is available at the moment. Therefore, it is important to allocate buffer space efficiently to reduce packet loss [4][6]. A buffer management algorithm determines how a buffer space is distributed among different applications. There have been many proposed buffer management algorithms since a packet buffer without an algorithm could not perform well under overload conditions [9 -16].

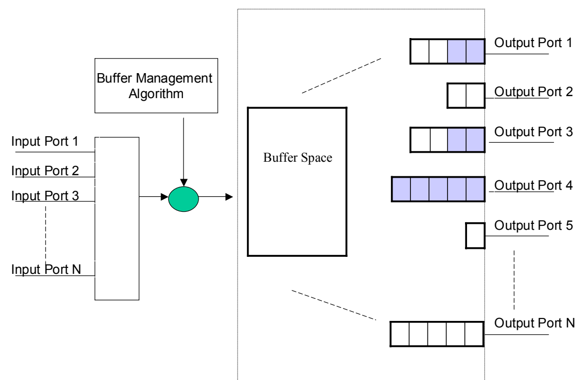

Figure 1. Packet Buffer

Therefore, it is important to allocate buffer space efficiently to reduce packet loss [4][6]. A buffer management algorithm determines how a buffer space is distributed among different applications. There have been many proposed buffer management algorithms since a packet buffer without an algorithm could not perform well under overload conditions [9 -16].

Traditionally, physical layer and data link layer processing is done on a NIC [1][2]. After the processing is done, the packet will be transferred to the main memory of a host processor. Then, the Operating System (OS) processes the IP header and the TCP/UDP header of the packet. In general, 20%-60% of the processing power of OS is used for protocol handling. Therefore, traditional packet reception architecture cannot work efficiently for a high-speed network, with a link speed more than 10 Gigabit per second [2].

Tomas Henriksson, et al. [2][7] proposed a protocol processor architecture. As shown in Figure 2, in this new architecture, packet reception scheme moves layer 3 and layer 4 processing to a NIC [1][6][7]. Packets received at NIC are processed for physical layer and link layer protocols as well as network layer and transport layer protocols. Protocol processor can handle protocol processing at a wire speed [1][7].

In particular, incoming packets will pass through the protocol processor and the payload (application) data will be stored in the packet buffer until the host application retrieves it [6]. 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.

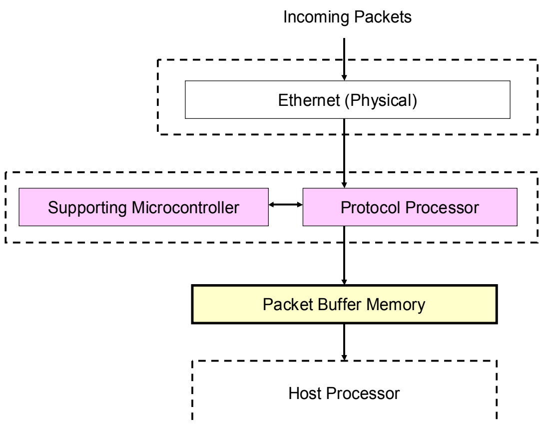

Figure 2. Protocol Processor Architecture [2]

II. BUFFER MANAGEMENT FOR A NIC

Buffer space can be distributed among different applications using either a static threshold scheme or a dynamic threshold scheme. This section discusses two static threshold schemes, Completely Partitioned Algorithm (CP) and Completely Shared Algorithm (CS); and two dynamic threshold schemes, Dynamic Algorithm (DA) and Dynamic Algorithm with Dynamic Threshold (DADT).

A. *Completed Partitioned Algorithm (CP)*

Kamoun and Kleinrock [11] proposed Completely Partitioned algorithm (CP). In CP, total buffer space is equally divided among all the applications. Packet loss for any application occurs when the allocated buffer space for that application is full. Let 'M' be the total buffer space, 'n' be number of applications and k_i, be the size of queues (i=1….n), we then have:

$$k_1 + k_2 + \ldots + k_n = M \quad (1)$$

CP is easy to implement in hardware works effectively when all the applications are active [4]. However, it is not adaptive to changes in traffic conditions. When one of the applications is inactive, it will not use the allocated buffer space, but CP does not reallocate buffer space between active applications.

B. *Completed Shared Algorithm (CS)*

Unlike the Completely Partitioned algorithm, the individual queues do not have any static thresholds placed on them in Completely Shared Algorithm (CS). Incoming packet will be accepted as long as there is space in the memory to accommodate it. Let 'M' be the total buffer space, 'n' be number of applications and k_i, be the size of queues (i=1….n), we then have:

$$0 \le k_i \le M, \; i = 1, 2, \ldots, N \quad (2)$$

CS is easy to implement in hardware works effectively under balanced load conditions. Generally, incoming packets are equally distributed between different applications, under balanced load conditions; hence, CS can provide fairness to all the applications. However, when one application is active at any moment, it is possible for this application to occupy the buffer space.

C. *Dynamic Algorithm (DA)*

Dynamic Algorithm (DA) is more adaptive to changes in traffic conditions than CS and CP. In DA, the threshold value for a particular application at any moment is a function of unused buffer space. Packets are accepted if queue length for the particular application is less than the corresponding threshold value. Otherwise, packets will be dropped. Let T (t) be the controlling threshold at time 't' and let Q_i (t) be the length of queue 'i' and Q(t) be the sum of all the queue lengths. Then if 'M' is the total buffer space

$$T(t) = \alpha \times (M - Q(t)) \quad (3)$$

Where 'α' is a constant. 'α' is chosen to be power of two, so shift registers can be used to implement DA in hardware. This algorithm is easy to implement in the hardware and is robust to changes in traffic conditions. However, DA does not consider that different applications often have different packet sizes. When an application with large packet size tends to occupy more buffer space, other applications will experience packet losses. So, DA is suitable for ATM switches because in ATM switch, packet size is same for all applications.

D. *Dynamic Algorithm with Dynamic Threshold (DADT)*

Dynamic Algorithm with Dynamic Threshold (DADT) [16] is similar to DA. But, different from DA, it also considers the packet sizes for different applications when the threshold values are calculated. The threshold value is calculated as follows:

$$T_i(t) = \alpha_i \times (M - Q(t)) \quad (4)$$

Where 'α_i' is proportionality constant and varies for each application. Optimum 'α' value for each queue is obtained through simulations. By varying the threshold value, DADT does not allow queues with large packet size to fill up the buffer quickly.

It has been shown that dynamic threshold scheme (DT) is more efficient than static threshold scheme (ST) [16]. Among the dynamic algorithms, DADT achieves the smallest packet loss ratio in network terminals. However, it is difficult for DADT to determine the optimum 'α' value for each application [16]. Moreover, the optimum 'α' turns to be different from a power of 2. So, it is difficult to implement DADT in hardware [16].

III. PROPOSED DYNAMIC ALGORITHM

DA and DADT consider unused buffer space when they calculate the threshold values, but not application state (i.e., active or inactive). Different from DA and DADT, we proposed a History-Based Dynamic Algorithm (HBDA), which considers all the three factors: unused buffer space, packet size and application state. The algorithm of HBDA is shown in Figure 3.

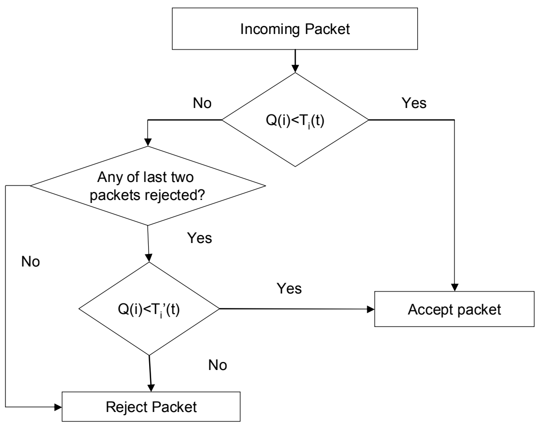

Figure 3. History-Based Dynamic Algorithm (HBDA)

In this figure, $T_i(t)$ and $T_i'(t)$ are the two controlling thresholds and $Q_i(t)$ be the length of queue 'i', at time't', and 'M' is the total buffer space. $T_i(t)$ and $T_i'(t)$ can be calculated as follows:

$$T_i(t) = (\alpha / psize_i) \times (M\text{-}Q(t)) \quad (5)$$

$$T_i'(t) = (\alpha / psize_i) \times (M\text{-}Q(t)) + History_i(1) \times M/a + History_i(2) \times M/b \quad (6)$$

Where, psize represents the packet size of an application, 'a' and 'b' are constants which are determined through simulations; and $History_i(1)$ is '1' if the last packet of the i^{th} application is rejected and '0' if accepted; and $History_i(2)$ is '1' if the second last packet of the i^{th} application is rejected and '0' if accepted.

The idea of HBDA is to consider application state and optimize the use of buffer while threshold value for each application is calculated. The following example explains how HBDA works in more detail. Let us consider three applications. Assume that at time 't', *application one* has filled up its allocated buffer space so the queue length of *application one* is greater than the threshold value for it. *Application two* is inactive at time 't', so queue length of *application two* is less than the threshold value for it. *Application three* at time 't' has filled nearly half of its allocated buffer space so queue length of *application three* is nearly half of the threshold value for it. If an incoming packet is for *application three*, it will be accepted since queue length for *application three* is less than the threshold value for it; while incoming packet for *application one* will be, rejected since queue length for *application one* is greater that its threshold value, although there is still free space available in the buffer.

Our simulation studies have shown that when an application fills its buffer space, then the dropping probability, for further few incoming packets for that application is high. In other words, by the time the packets for that application are dequeued [4] and the threshold value for that application is increased above the queue length, a few packets for that application have already been rejected.

Based on this observation, we keep track of last two packets for each application. If an application has queue length greater than the threshold value (calculated by equation 6) and any of the last two packets for that application has been rejected, we increase the threshold value for such an application (calculated by equation 7). In such a way, packet loss for that application can be minimized. Though, this increase in efficiency will come at some additional cost of some hardware (registers). In fact, tracking the last three packets for an application can further increase the efficiency of buffer management algorithms, but the hardware cost increases a lot as compared to increase in efficiency.

A. *Threshold Value Computation in HBDA*

In DADT, the threshold value is calculated as shown in equation 7.

$$T_i(t) = \alpha_i \times (M\text{-}Q(t)) \quad (7)$$

Different applications have different alpha value in DADT. In general, the optimum alpha value comes out to be different than the power of two in DADT [16]. Also, in DADT, determining the optimum alpha value for each application is difficult. Therefore, equation for calculating the threshold value for an application has been modified as shown in equation 5.

In equation 5, alpha value is same for all the applications and since different applications will have different packet size, factor of 'α / psize' in equation 5, achieves the same effect as different alpha values for different applications, in DADT and this eliminates the need to determine the optimum alpha value for each application.

When an application has a queue length greater than the threshold value, then the threshold value for such an application is determined using equation 6.

The 'α' value is generally taken as a power of two (either positive or negative), so that threshold computation is easy to implement in the hardware.

TABLE I. OPTIMUM VALUE OF FACTORS

(a,b)	Packets Rejected / Total Packets Arrived
2,2	0.093
2,4	0.081
2,8	0.085
4,4	0.088
4,8	0.090

In our simulations, we used six applications, bursty uniform traffic model , alpha value as 128 (from table 4) , average traffic mix, an average dequeue time of 14 clock cycles for the burst of 10 packets, a buffer size as 600 packets and a load of 70% on each of the queue. Our simulation studies shows that optimum value of 'a' and 'b' comes out to be 2 and 4 respectively, as shown in Table 1. Unlike alpha value in DADT, the value of parameters 'a' and 'b' has to be calculated only once and the value is same for all the applications.

IV. VHDL BASED SIMULATION MODEL

We have developed the simulation model for a packet buffer by using VHDL as shown in Figure 4. In Figure 4, there are three important modules, Traffic Generator, Packet Buffer and Output Link.

The *Traffic Generator* block produces (packets) according to inputs provided in the Configuration file (Config file). The Config file specifies traffic model and load on each port [4].

There are three kinds of traffic model that are available. These are described as follows:

- 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;
- 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;
- Bernoulli Uniform Traffic Model: Bernoulli arrivals, destinations uniformly distributed over all the output ports.

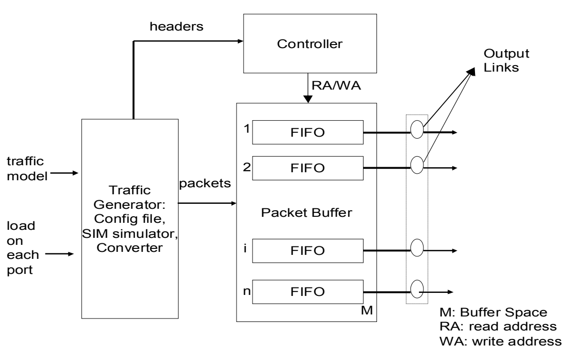

Figure 4. Simulation model for the packet buffer

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 10:

$$\rho = L_b/(L_b+L_{idle}) \qquad (10)$$

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

Traffic Generator produces packets with a mean inter-arrival time and mean burst length [4]. The 'SIM' simulator [15] is used for producing the packets traces. The packet traces from the 'SIM' Simulator is written to input file.

The Packet Buffer (i.e. shared memory) is First In-First Out (FIFO) queues [4]. Depending on the memory (buffer) size and the number of output queues, the memory can be partitioned among the different output queues.

The Output Link is to remove the packets from the memory after a certain dequeue time [4]. Dequeue time, is one unit of time, which matches the inter-packet time [4]. We modeled dequeue time as a Poisson random variable with a fixed mean [6].

V. SIMULATION RESULTS AND ANALYSIS

In our simulations average network traffic is considered. We used Bursty Uniform Traffic Model for our simulations since this is the most commonly used traffic model [16, 19]. For each traffic load, the following steps have been followed:

a) Optimum alpha value is determined for DA.

b) Optimum combination of alpha values for different queues is determined for DADT. Optimum alpha values are the combination of alpha for different queues for which DADT gives minimum packet loss ratio.

c) Optimum alpha value is determined for HBDA.

d) Packet loss ratio is plotted for DA, DADT and HBDA as the load is varied, keeping the buffer size constant.

e) Packet loss ratio is plotted for DA, DADT and HBDA as the buffer size is varied, keeping the load constant.

f) Improvement ratio is calculated for different values of load for the average traffic mix. Improvement ratio is defined as the difference of packet loss in HBDA and the compared algorithm (DA or DADT) divided by packet loss in HBDA.

For simulations purposes, we have used the number of the applications as six, bursty uniform traffic model and average dequeue time of 14 clock cycles for the burst of 10 packets. For a confidence interval of around 95% [14], we require a minimum of 10^6 packets.

We implemented a traffic mix with average network traffic load according to [6]. We first determined the optimum 'α' (alpha) value for DA.

TABLE II. QUEUE PROPERTIES FOR AVERAGE TRAFFIC LOAD

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

Table 2 shows the packet sizes for different applications based on the average network traffic load flow in [6]. For our simulation purpose, we have used these packet sizes for different applications.

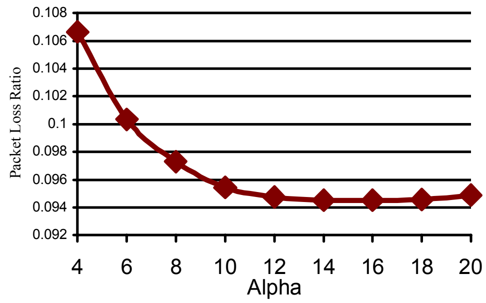

Figure 5. Packet Loss Ratio Vs Alpha for DA for average traffic load

Figure 5 shows the packet loss ratio (number of dropped packets/ number of received packets [8]) in DA. In Figure 5, size of buffer is 600 packets, and load on each queue is 70%. As shown in Figure 5, packet loss ratio decreases when 'α' varies from 4 to 14, but increases when 'α' value is greater than 14. Large 'α' value increases the control threshold of the queues of large packet, thus preventing large packets from being dropped even though they have occupied a large number of buffer space. Based on the result, we determine the optimum 'α' value to be 14 for DA.

For DADT, we have also done the simulations to determine optimum 'α' values corresponding to different queues. Table 3 shows the different combinations of alpha and Figure 6 shows the packet loss ratio corresponding to different combinations.

TABLE III. COMBINATION OF ALPHA VALUE FOR DADT

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

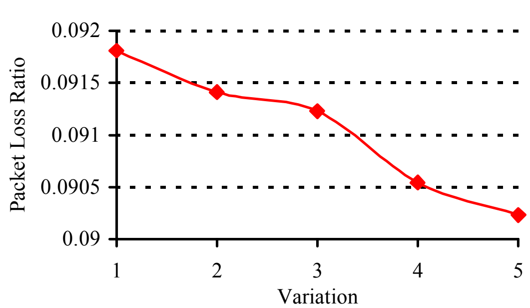

Figure 6. Packet loss ratio vs. variations of alpha for DADT

The alpha values in combination 5 (from table 3) are chosen for DADT.

Table 4 shows the packet loss ratio for HBDA as 'α' value is changes from 16 to 256. As shown from table 4, optimum 'α' value comes out to be 128. So, we will use 128 as 'α' value for HBDA.

TABLE IV. VARIATION OF ALPHA VALUE FOR HBDA

'α' value	Packet Loss Ratio
16	0.095
32	0.093
64	0.087
128	0.081
256	0.083

Figure 7 compares the performance of HBDA, DA and DADT under different load conditions. Here, buffer size is set to 600 packets. Load varies from 0.5 to 0.9. As seen in Figure 7, for all the loads, HBDA has lowest packet loss ratio.

Figure 8 shows the performance HBDA, DA and DADT with different buffer size, Here, each queue is with 70 percent load. The buffer size varies from 500 packet size to 800 packet size. As seen from the Figure 8, as the buffer size increases, packet loss ratio decreases for all the three algorithms. This is due to the fact that all applications get more buffer space.

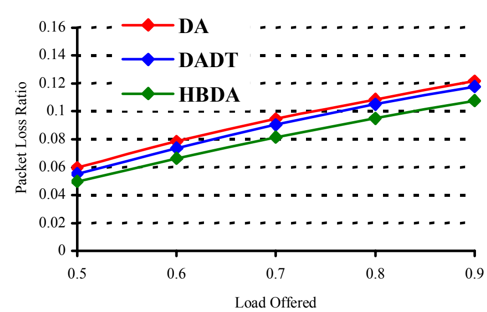

Figure 7. Packet Loss Ratio Vs Load for HBDA, DA, DADT

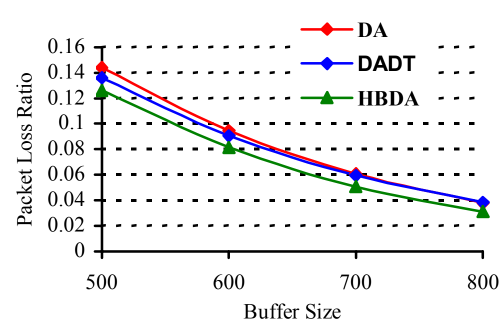

Figure 8. Packet Loss Ratio Vs Buffer Size for HBDA, DA, DADT

Table 5 shows the improvement in packet loss ratio for HBDA, for different loads when compared with DA and DADT. The improvement ratio is defined as the difference of packet loss in HBDA and the compared algorithm (DA and DADT) divided by packet loss in HBDA.

TABLE V. IMPROVEMENT RATIO OF HBDA OVER DA AND DADT

Load	Improvement ratio (%) (HBDA / DA)	Improvement ratio (%) (HBDA / DADT)
0.5	20	10.89
0.6	18.4	11.29
0.7	15.9	11.02
0.8	14.4	10.77
0.9	13	9.52

VI. CONCLUSION

This paper proposes a History-Based Dynamic algorithm (HBDA) to reduce the packet loss in a Network Interface Card. HBDA considers packet size and application state. HBDA can adjust threshold values in a timely manner and adapt to network conditions quickly. As a result, HBDA effectively controls packet losses at Network Interface Card.

The Dynamic algorithm (DA) works well for ATM switches where packet size is same for all the applications. However in network terminals, different applications have different packet sizes. Therefore, if we use DA, application with large packet size tends to occupy more buffer space resulting in packet loss of other applications. The Dynamic Algorithm with Dynamic Threshold (DADT) takes only the packet size into consideration and not the application state while calculating the threshold. Also, it is difficult to determine the optimum alpha value for each application in DADT.

The simulations considered a buffer size of 600 packets, 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 average network traffic loads [6], the HBDA improves the packet loss ratio by 15.9% and 11% (for load = 0.7) compared to DA and DADT, respectively.

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] M. Arpaci, J.A. Copeland, "Buffer Management for Shared Memory ATM Switches", *IEEE Communication Surveys*, First Quarter 2000, Vol. 3, No. 1, pp. 2-10.

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

[6] 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.

[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 and Yul Chu, "An Enhanced Dynamic Packet Buffer management," in the *proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC 2005),* pp. 869-874, La Manga del Mar Menor, Cartagena, Spain, June 27-30, 2005.