2005

An Enhanced Dynamic Packet Buffer Management

Vinod Rajan

Yul Chu
The University of Texas Rio Grande Valley

Recommended Citation

Rajan, Vinod, and Yul Chu. 2005. "An Enhanced Dynamic Packet Buffer Management." Proceedings of the 10th IEEE Symposium on Computers and Communications, ISCC '05, , 869–874. https://doi.org/10.1109/ISCC.2005.27.

An Enhanced Dynamic Packet Buffer Management

Vinod Rajan
Cypress Southeast Design Center
Cypress Semiconductor Cooperation
vur@cypress.com

Yul Chu
Dept. of Electrical and Computer Engineering
Mississippi State University
chu@ece.msstate.edu

Abstract

A packet buffer for a protocol processor is a large shared memory space that holds incoming data packets in a computer network. This paper investigates four packet buffer management algorithms for a protocol processor including Dynamic Algorithm with Different Thresholds (DADT), which is proposed to reduce the packet loss ratio efficiently. The proposed algorithm takes the advantage of different packet sizes for each application by allocating buffer space for each queue proportionally. According to our simulation results, the DADT algorithm works well in reducing packet loss ratio compared to other three algorithms.

1. Introduction

Data is transmitted over the computer network as the form of data packets [1]. Each packet consists of necessary data for an application associated with headers (control information). Packets coming into the computer network terminal are processed and classified based on its destination application by the protocol processor [2]. The processed application (payload) data is stored in a packet buffer until the host application retrieves it [3].

The packet buffer is a large dual ported memory shared by all output queues. Packets for each application are multiplexed into a single stream and fed into the packet buffer for storage. In a packet buffer, the packets are organized into logical FIFO queues (output queues), one for each application [2][4].

Each application has a logical FIFO queue inside the packet buffer associated with it [3]. Packets coming in for different applications at different data rates can fill up the large FIFO-based packet buffer. Once the packet buffer is full, further incoming packets will be dropped. This packet dropping (or loss) happens whenever the buffer is full. Therefore, it is important to reduce packet loss ratio to support any end-to-end application over the computer network [5][6]. Packet loss ratio can be improved by using buffer management algorithms. Buffer management algorithms determine how the total buffer space is distributed among the various output queues. Therefore, 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 number of packets dropped to the total number of packets received [8]; and 2) Hardware complexity: The amount of hardware that is required to implement a given buffer management algorithm.

There have been many proposed buffer management algorithms since a packet buffer without an algorithm could not perform well under overload conditions [9 -16]. In this paper, we introduced three popular buffer algorithms; Completely Partitioned algorithm, Completely Shared algorithm, and Dynamic algorithm [11][14].

The purpose of these algorithms is to provide an even packet loss ratio for all output queues in a buffer. However, when these algorithms are applied to a packet buffer, they have an uneven packet loss ratio for the various output queues. This is due to the different packet sizes for each output queue (refer to section 3). In addition, queues with large packet sizes tend to fill up the buffer space at a much faster rate, resulting in excessive packet losses for other output queues. Therefore, we propose an efficient buffer management algorithm called DADT to develop: 1) a dynamic buffer management algorithm intended for protocol processors, which reduces the packet loss ratio by using different threshold values for the queues; and 2) a simulation model of the packet buffer intended for a protocol processor.

This paper is set out as follows: Section 2 discusses the background information for a protocol processor and related works regarding buffer management algorithms; section 3 introduces the proposed DADT algorithm; section 4 discusses our simulation model to measure the performance metrics for buffer

management algorithms; section 5 provides and analyzes the simulation results; and section 6 gives the conclusions.

2. Background and Related Works

To reduce the bottleneck, Henrikkson et. al [2][4][7] proposed off-loading the host processor for handling the protocol tasks. The idea is to move a processor (protocol processor for handling layers 3-4) to Network Interface Card (NIC). They showed significant reduction processing time spent by the host processor in 3-4 layers.

The memory organization using a protocol processor is shown in Figure 1. The protocol processor consists of three major components: 1) General Purpose Micro-controller (μC) to control the slow path operations; 2) Programmable Protocol Processor (PPP) to process the data at a wire speed; and 3) Packet Buffer Memory (PBMEM) to hold incoming packets [5].

As shown in Figure 1, 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 [5]. Packets are classified based on the application (per-flow). Once the packet is classified, it is stored in an output queue in the buffer [5].

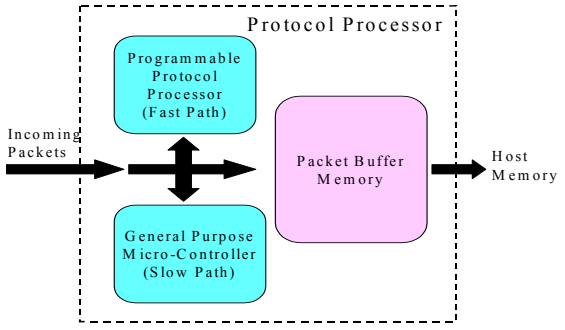

Figure 1. Major components of a protocol processor

Different applications have different sizes of packet associated with it [5]. The packets for different applications at different data rates can fill up the large FIFO-based packet buffer. To reduce the packet losses, a sophisticated algorithm is required to manage the buffer efficiently.

There are various buffer management algorithms that exist for managing a packet buffer [8-10][12-13]. Some of the popular algorithms are: 1) Completely Partitioned: The entire buffer is completely partitioned with a constant buffer size for each queue; 2) Completely Shared: the entire buffer is shared among the various queues; and 3) Dynamic: This algorithm varies the size of each queue dynamically based on the remaining space in the buffer.

Kamoun and Kleinrock [11] proposed Completely Partitioned algorithm; the total buffer space 'M' is divided equally among the queues. Since the number of output queues are known, the buffer space allocated to each and every queue can be pre-determined. Since the buffer space for every queue is decided statically, this is known as static threshold scheme. Packet loss for a queue occurs when the buffer space allocated to the particular queue becomes full. If it cannot accommodate any more packets, the incoming packet is dropped. If k_i, i= 1…n, represents the size of queues i=1…n, and M is the total buffer space, then:

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

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

The advantage of this algorithm is that it works well when all the output queues are competing for the buffer space [6]. It can be implemented easily in hardware [6]. The drawback is that it sometimes rejects incoming packets (or cells) even though there is a space left in the buffer [6].

Completely Shared algorithm allows the output queues to completely share all the available space in the buffer. If there is a memory space available in the buffer, the corresponding incoming packet will be accommodated in the buffer [11]. In other words, individual buffer allocations may run up to the total buffer memory.

Packet loss occurs only when the entire buffer is full. Unlike the Completely Partitioned case, the individual queues do not have any static thresholds placed on them. If k_i, i= 1…n, represents the size of queues i=1…n, and M is the total buffer space, then:

$$k_i = M,\ i = 1,2,\ldots,N$$

The advantage of this algorithm is that it is well suited for balanced traffic since it allows complete sharing of the buffer space [6]. It is easy to implement in hardware [6]. The drawback is that a single output queue can monopolize most of the buffer space if load on its corresponding input port is very high.

For the Dynamic algorithm, the amount of buffer space each queue is decided by a threshold placed on each queue [14]. This threshold is called as the control threshold. The main idea is that the control threshold of each queue is proportional to the remaining space in the buffer.

$$T(t) = f(B - Q(t)) = f\left(B - \sum_i Q^i(t)\right) \quad (3)$$

Where T(t) is the control threshold, B is the total buffer space, Q(t) is the sum of all the output queue lengths and f is a proportionality constant. When an incoming packet to the output queue 'i' finds that the current queue length ($Q^i(t)$) is greater than the current threshold value (T(t)), then the incoming packet is dropped. Since then, no further packets are accepted into the queue unless the existing packets in queue i are drained out or else T(t) increases beyond $Q^i(t)$ [14]. The simplest scheme based on the above theory is the Dynamic algorithm [14]. Dynamic algorithm chooses the control threshold to be a multiple 'α (alpha)' of the remaining buffer space.

$$T(t) = \alpha \cdot (B - Q(t)) \tag{4}$$

3. Proposed Dynamic Algorithm

The DADT algorithm is similar to the dynamic algorithm except that it has multiple threshold values as opposed to a single control threshold value in the Dynamic algorithm.

$$T_i(t) = \alpha_i \cdot (B - Q(t)) \tag{5}$$

where α_i is the proportionality constant and varies for each and every queue.

The Dynamic and DADT algorithms have two major advantages over the static threshold schemes: 1) It is adaptive to change according to traffic conditions [14]. Once an output queue becomes active, it starts receiving packets and its queue length increases. This increases the total buffer occupancy, and the control threshold decreases correspondingly. If the current queue length is greater than the control threshold value, the incoming packets will be dropped until the output queues naturally drain out [14]; and 2) It is easy to implement in hardware. The only requirements are queue length counters and comparators with a shift register [6][14].

DADT algorithm has similar properties like the Dynamic algorithm (i.e. the control threshold value of the queues is directly proportional to the remaining space in the buffer) except different thresholds for each queue. The DADT algorithm is specially developed for the packet buffer, while the Dynamic algorithm was initially developed for an ATM switch [6]. The ATM switch has a fixed size of incoming ATM packets [6]. That is why the behavior of all the output queues is similar when experiencing uniform traffic. Hence, one queue cannot monopolize the buffer space when experiencing uniform traffic. However, for the case of the packet buffer on a NIC, there are various incoming packet sizes for output queues [5]. With different packet sizes for each queue, having a single threshold value for all the queues may let one queue monopolize the entire buffer space. Queues with large incoming packet sizes tend to fill the buffer space at a much faster rate. Having a single threshold value for all the queues results in excessive packet loss for other output queues. According to our simulation results, by controlling the threshold values of each queue separately, the proposed DADT algorithm can decrease the packet loss at other queues efficiently.

4. Simulation Model

We developed the simulation model for a packet buffer by using VHDL as shown in Figure 2. Most of the blocks in Figure 2 resemble the blocks in a shared memory switch model [6]. In Figure 2, the *Traffic Generator* block produces output (packets) according to two inputs (Traffic Model and Load on each port) [6].

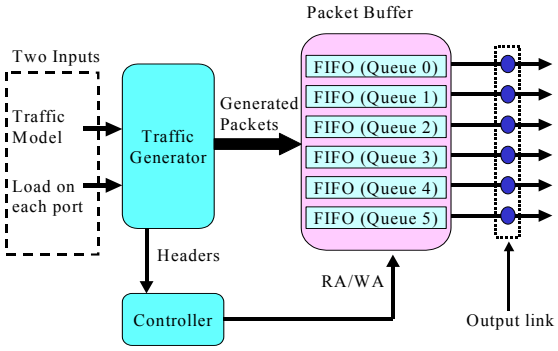

Figure 2. Simulation model for the packet buffer

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 [14] and is given by the equation:

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

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 2, 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.).

The Packet Buffer (shared memory) in Figure 2 is arranged in terms of First In-First Out (FIFO) queues [6]. Depending on the memory (buffer) size and the number of output queues, the memory can be partitioned among the different output queues.

The primary task of the Output Link in Figure 2 is to remove the packets from the memory after a certain dequeue time [6]. Dequeue time, for the simulations shown in [6], is one unit of time, which matches the inter-packet time [6]. For simulations of the packet buffer memories on the NIC (where the packets stay for a specified amount of time), we model dequeue time as a Poisson random variable with a fixed mean [5].

5. Simulation Results and Analysis

We implemented a traffic mix with average network traffic loads according to [5]. With this traffic mix, we determined the optimum value of α (alpha) and then compared the performance between the DADT and Dynamic algorithm.

Figure 3 explains the properties of six output queues of our simulation model, which are based on the average network traffic load flow in [5].

Figure 3 shows that Queue 5 has the largest buffer size (512 bytes), and Queues 0, 2, and 5 can manage more incoming packets than other Queues.

	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

** Queue i = Queue i (e.g., Queue 1 = Q1)

Figure 3. Queue properties for average traffic load

Figure 4 shows the variation of packet loss ratio (# of dropped packets / # of received packets) with alpha for the Dynamic algorithm through the traffic mix. In Figure 4, we implemented the buffer with 600 incoming packets, six output queues, bursty uniform traffic model with the load of 70% on each of the queues and the average dequeue time of 14 clock cycles for the burst of 10 packets. In Figure 4, the loss ratio first decreases till *alpha* = 10.0 and then increases afterwards because larger *alpha* values can increase the control threshold of the queues with large packet sizes.

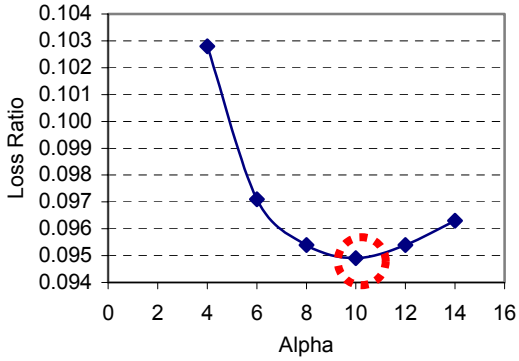

Figure 4. Packet loss ratio vs. Alpha for Dynamic

The increase in control threshold prevents them from being dropped even though they have a large queue length. Therefore, we determined the optimum value of *alpha* for the Dynamic algorithm as 10. Our results show that this is the best packet loss ratio that Dynamic algorithms can achieve.

As discussed in section 3, for the DADT algorithm, each queue has different threshold values since the alpha can be varied for each queue. Therefore, the optimum value of alpha for a given queue depends upon the packet size it receives. As different queues receive different sizes of packet, the optimum value of alpha is dependent on the traffic mix. Our simulation results show that the optimum value of *alpha* is determined between 6 and 16. Table 1 shows the best

optimum *alpha* values for each Queue based on our simulation results and also shows the improvement ratio (%) for *DADT over Dynamic algorithm*. According to Table 1, we found that *DADT* would work better (> 6%) than *Dynamic algorithm*.

Table 1. Optimum alpha value for DADT

	Q0	Q1	Q2	Q3	Q4	Q5	Improvement ratio (%) (DADT/Dynamic)
1st	16	14	16	14	14	6	6.7%
2nd	15	12	15	12	12	6	6.1%
3rd	16	14	16	14	16	8	5.5%

** Qi = Queue i (ex: Queue 1)

The performance comparison for other buffer management algorithms for the traffic mix is shown in Figure 5. Figure 5 shows the performance of the four algorithms (Completely Shared, Completely Partitioned, Dynamic, and DADT) for varying loads. As seen in Figure 5, the DADT algorithm has the most efficient packet loss ratio for all the loads. The packet loss ratio increases for all the algorithms with increasing load on the queues. Notice that the performance difference decreases somewhat at higher loads. The Completely Shared and Completely Partitioned algorithms have higher packet loss ratio than Dynamic and DADT algorithm. Figure 5 shows that DADT algorithm works better than Dynamic just like Dynamic works well than Completely Shared or Completely Partitioned [14].

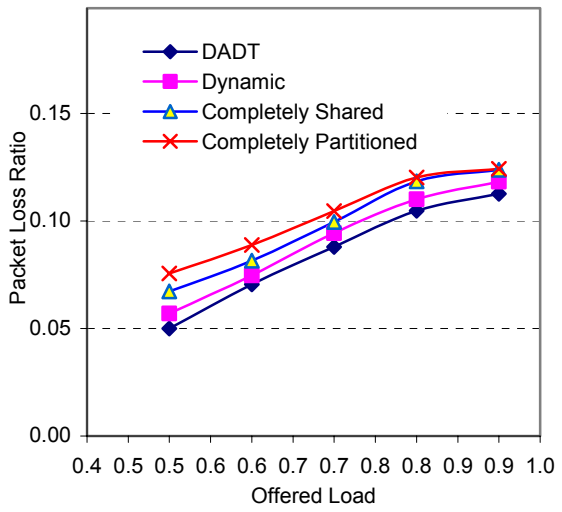

Figure 5. Packet loss ratio vs. Load for algorithms

The performance of DADT and Dynamic algorithm are compared when the buffer size is increased. Figure 6 shows the variation of packet loss ratio for increase in buffer sizes. Increasing the number of packets in the buffer increases the buffer size. In Figure 6, the packet loss ratio decreases by an average of 18.5% and 18 % for an increase of 100 packets for both the Dynamic and DADT algorithm respectively. With an increase in buffer size, since each queue has a much bigger space, it is reasonable for a queue to accommodate more number of packets. This results in an improvement of packet loss ratio as well.

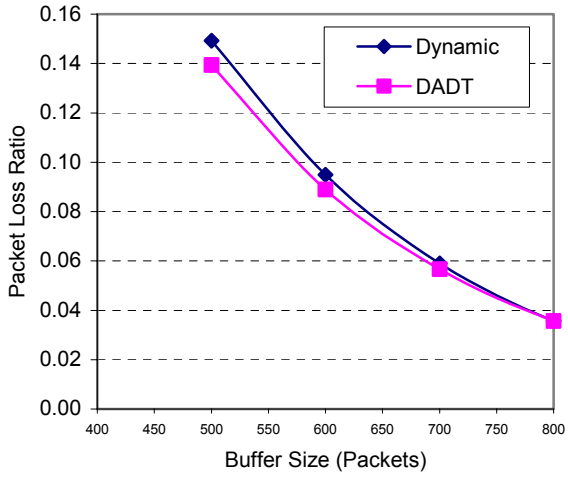

Figure 6. Packet loss ratio vs. Buffer Size for Dynamic algorithm and DADT

6. Conclusions

This paper proposes the Dynamic Algorithm with Dynamic Thresholds (DADT) to reduce the number of packets being dropped at the packet buffer. Packets coming into the packet buffer can be dropped if the output queue in a buffer is full and each packet size can be different.

Packets with the same size will be stored in the same queue of the buffer. A buffer management algorithm will decide the amount of space for each output queue in the packet buffer. Four buffer management algorithms are implemented for our simulations: 1) Completely Partitioned algorithm; 2) Completely Shared algorithm; 3) Dynamic algorithm; and 4) DADT algorithm.

The DADT algorithm has different threshold values for each queue, which is different from other algorithms such as the Dynamic algorithm. The DADT algorithm takes advantage of the different size of packets coming into each output queue. By observing the packet size for a particular queue, we can determine an optimum value of threshold for the queue. With the optimum threshold value for a queue, the DADT algorithm can reduce the number of packets dropped significantly.

The optimum threshold value for a queue depends upon the size of a packet it receives. Different-sized packets were used to determine the optimum value of threshold during simulations. The simulations considered a buffer of size as 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 output queues. For the traffic mix with average network traffic loads [5], the DADT algorithm improves the packet loss ratio by 6.7% compared to the Dynamic algorithm and more than 10% for other algorithms.

7. 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", *IEE 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] Sundar. I, McKeown. N, "Techniques for Fast Shared Memory Switches", *Stanford University HPNG Technical Report*, TR01-HNPG-081501, Stanford, March 2001.