Packet Buffer Management for a High-Speed Network Interface Card

Shalini Batra

Yul Chu
The University of Texas Rio Grande Valley

Yan Bai

Follow this and additional works at: https://scholarworks.utrgv.edu/ece_fac

Part of the Electrical and Computer Engineering Commons

Recommended Citation

This Conference Proceeding is brought to you for free and open access by the College of Engineering and Computer Science at ScholarWorks @ UTRGV. It has been accepted for inclusion in Electrical and Computer Engineering Faculty Publications and Presentations by an authorized administrator of ScholarWorks @ UTRGV. For more information, please contact justin.white@utrgv.edu, william.flores01@utrgv.edu.
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].

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.

\[ k_1 + k_2 + \ldots + k_n = M \]  

(1)

\[ 0 \leq k_i \leq M, \quad i=1, 2, \ldots, N \]  

(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 \( Q(t) \) be the sum of all the queue lengths. Then if \( M \) is the total buffer space

\[ T(t) = \alpha \times (M-Q(t)) \]  

(3)

Where \( \alpha \) is a constant. \( \alpha \) 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:

\[ T(t) = \alpha \times (M-Q(t)) \]  

(4)

Where \( \alpha \) is proportionality constant and varies for each application. Optimum \( \alpha \) 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 \( \alpha \) value for each application [16]. Moreover, the optimum \( \alpha \) turns to be different from a power of 2. So, it is difficult to implement DADT in hardware [16].
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)](image)

In this figure, \( T_i(t) \) and \( T'_i(t) \) are the two controlling thresholds and \( Q(t) \) 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:

\[
T_i(t) = \frac{\alpha}{\text{psize}} \times (M-Q(t)) \tag{5}
\]

\[
T'_i(t) = \frac{\alpha}{\text{psize}} \times (M-Q(t)) + \text{History}(1) \times \frac{M}{a} + \text{History}(2) \times \frac{M}{b} \tag{6}
\]

Where, \( \text{psize} \) represents the packet size of an application, ‘\( a \)’ and ‘\( b \)’ are constants which are determined through simulations; and \( \text{History}(1) \) is ‘1’ if the last packet of the \( i^{th} \) application is rejected and ‘0’ if accepted; and \( \text{History}(2) \) 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 than 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.

\[
T_i(t) = \alpha_p \times (M-Q(t)) \tag{7}
\]

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 ‘\( \alpha_p / \text{psize} \)’ 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 ‘\( \alpha \)’ 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>
<thead>
<tr>
<th>((a,b))</th>
<th>Packets Rejected / Total Packets Arrived</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,2</td>
<td>0.093</td>
</tr>
<tr>
<td>2,4</td>
<td>0.081</td>
</tr>
<tr>
<td>2,8</td>
<td>0.085</td>
</tr>
<tr>
<td>4,4</td>
<td>0.088</td>
</tr>
<tr>
<td>4,8</td>
<td>0.090</td>
</tr>
</tbody>
</table>
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 show 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.

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 106 packets.

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

\[
\rho = \frac{L_b}{L_b + L_{idle}} \quad \text{(10)}
\]

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

![Figure 4. Simulation model for the packet buffer](image)

**TABLE II. QUEUE PROPERTIES FOR AVERAGE TRAFFIC LOAD**

<table>
<thead>
<tr>
<th></th>
<th>Q0</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet Size (in Bytes)</td>
<td>256</td>
<td>64</td>
<td>256</td>
<td>32</td>
<td>128</td>
<td>512</td>
</tr>
<tr>
<td>Packet Unit # (32 bytes/unit)</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>
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.

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 3 shows the different combinations of alpha and Figure 6 shows the packet loss ratio corresponding to different combinations.

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**

<table>
<thead>
<tr>
<th>Variation</th>
<th>Q0</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>10</td>
<td>14</td>
<td>10</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>12</td>
<td>14</td>
<td>12</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>14</td>
<td>16</td>
<td>14</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>14</td>
<td>16</td>
<td>14</td>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>

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.

**TABLE IV. VARIATION OF ALPHA VALUE FOR HBDA**

<table>
<thead>
<tr>
<th>‘α’ value</th>
<th>Packet Loss Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.095</td>
</tr>
<tr>
<td>32</td>
<td>0.093</td>
</tr>
<tr>
<td>64</td>
<td>0.087</td>
</tr>
<tr>
<td>128</td>
<td>0.081</td>
</tr>
<tr>
<td>256</td>
<td>0.083</td>
</tr>
</tbody>
</table>

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 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>
<thead>
<tr>
<th>Load</th>
<th>Improvement ratio (%) (HBDA / DA)</th>
<th>Improvement ratio (%) (HBDA / DADT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>20</td>
<td>10.89</td>
</tr>
<tr>
<td>0.6</td>
<td>18.4</td>
<td>11.29</td>
</tr>
<tr>
<td>0.7</td>
<td>15.9</td>
<td>11.02</td>
</tr>
<tr>
<td>0.8</td>
<td>14.4</td>
<td>10.77</td>
</tr>
<tr>
<td>0.9</td>
<td>13</td>
<td>9.52</td>
</tr>
</tbody>
</table>

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