ISSN (Print) : 0974-6846
**Indian Journal of Science and Technology,** *Vol 10(12), DOI: 10.17485/ijst/2017/v10i12/113008, March 2017* ISSN (Online) : 0974-5645

# Mathematical Models for Network Card Packet Buffering Simulation

**Okta Nurika, Mohd Fadzil Hassan, Nordin Zakaria and Low Tan Jung**

Universiti Teknologi PETRONAS, Seri Iskandar, Tronoh - 32610, Perak, Malaysia;
okta.rider@gmail.com, mfadzil_hassan@utp.edu.my,
nordinzakaria@utp.edu.my, lowtanjung@utp.edu.my

## Abstract

**Objectives:** Current network card buffer optimizations are focusing on optimal configuration of transmission's sender with singular buffer. However, there exists receiver-oriented network card module with multiple buffers. In our research, we mirrored PF_RING, which alters the traditional packet processing that requires copying the packets to application space. **Methods/Statistical Analysis:** PF_RING provides buffers for higher layer applications to access packets directly. **Findings:** Each buffer's capacity, transmission size, packet size, number of packets, and number of applications give raise to the question on the number of buffers required, because it affects the duration of packets read by the applications. Optimal number of buffers is therefore needed to generate minimum packet reading duration of the applications. **Application/Improvements:** We propose mathematical models to represent the simulation of packet buffering in a multi-buffer receiver's network card. Our simulation models are used to determine the optimal number of buffers for specific network transmission with distinct properties, which include transmission size, packet size, and number of packets.

**Keywords:** Mathematical Model, Multi-Buffer, Network Card, Simulation

## 1. Introduction

Packet queuing or buffering takes place when the network card receives more packets than it can process. The received packets can then be accessed by applications, by copying the packets to their workspace.

If there are multiple applications that need to copy the packets, then the copying of packets must be done alternately among the applications. A way to speed up packets copying among the applications is to have multiple buffers. However, more buffers do not mean faster copying of packets among the applications. There is time consumed to create buffers. More buffers mean more time is needed to create them. Then there is time constraint to distribute packets to the multiple buffers. More buffers mean more time is needed to distribute the packets. Another time constraint is the duration needed for applications to move from a buffer to another. More buffers mean more movements. A correct number of buffers are therefore needed to ensure the minimum time required for software applications to access the packets.

Based on our literature review, simulation of multiple buffers for packet distribution has not been done yet. The current works on buffer optimizations focus only on the packet transmission, either on the sender side or on the receiver side.

We would thus to propose mathematical model for network card multiple buffers optimization. Our model is to complement the current works that do not consider application level constraint.

## 2. Network Buffering Simulations Review

A comprehensive study of buffer bloat phenomena was conducted[1]. This was motivated by the effects of having different buffer sizes in router ports. Buffer size optimization will affect throughput, packet queue length, and packet drop rate. Optimal buffer size depends on several factors, which are packet size, transmission size, number of transmissions, duration of transmission, transmission frequency, bandwidth, size of TCP receive window[1].

* *Author for correspondence*

Different arbitrary buffer sizes and their effects on throughput and latency were simulated and investigated[1]. Bigger buffer size does not necessarily increase the throughput. The TCP behavior in certain characteristics may even decrease the throughput and the packet drop rate could be higher or lower (latency). The optimal buffer size needs to be determined carefully to achieve good throughput with low latency.

Mixed Integer Linear Programming (MILP) framework was built[2] to optimize the allocation of logical buffers to the physical memories (Dynamic Random Access Memory), where there are multiple core of processors. The number of logical buffer links allocated to each physical memory is not allowed to over-capacitate the physical memory. This method was proven to be able to distribute the load of logical buffers, such that the physical memories had less load compared to the static allocation method. Their buffer allocation optimization context is different from ours, because our works are on network card instead of on processors.

Sender oriented buffer allocation optimization in TCP connection was proposed[3]. Different TCP connections need their own size of buffer to fully utilize their specific bandwidth. Their solution of local optimization determines the optimal buffer size for an active TCP connection, while the global optimization works on multiple TCP connections requiring different size of buffers. These methods were able to divide the 'sender window' size optimally among single and multiple connections. Hence, a connection with lower bandwidth receiver will not be over-allocated, while another connection with higher bandwidth receiver will not be under-allocated. Our research work in buffer allocation optimization will complement their work, because our work focuses on the receiver side's buffer with number of applications constraint, whether it should be one buffer only (per-flow type) or multiple buffers (round-robin type) with 6 clusters/buffers at maximum.

Packet categorizer/classifier to prioritize packets, based on the packet type[4] was simulated. In this case, the management packets over the multicast and unicast packets are prioritized. This categorization and prioritization were included in link layer. However, the structure was independent from the link layer itself, which makes it modular. Besides to have queuing based on strict

priorities, they also implemented weighted round robin queuing to prioritize specific traffic types. The queuing methods are generic, since the packets of same type can be from different data transmissions.

Hence, their queuing approach[4] is packet-oriented, and not transmission oriented. The existence of applications that need to access those packets have not been taken into account. Our research approaches the optimization in different ways. Instead of single buffer, we are using multiple buffers and we also consider the number of applications that need to access the packets.

Furthermore, we include transmission size as a constraint.

Dynamic programming to optimize the allocation of buffer size among groups of links connected to the server was implemented[5]. The objective was to distribute the total bandwidth to each link in every group, so that the throughput can be maximized. Their approach was applicable for industry production line and computer data flow. For example the distribution of bandwidth from a network segment to another network segment. However, this is different from our buffer allocations optimization. Our optimization is for buffer inside the network card and it does not involve multiple sequential links. Instead it involves multiple buffers with considerations for higher layer applications, data transmission and packet size. Moreover, our throughput is determined by the minimum time required for all applications to finish their access to the packets inside the buffer buckets.

Dynamic ATM (Asynchronous Transfer Mode) switch buffer allocation among multiple ports has been developed[6]. They hybridized Complete Sharing (CS) technique to share the whole buffer, so when the traffic load is between low to medium, it will activate pure CS, but when the traffic increases, it will behave like either Partial Sharing (PS) or Complete Partitioning (CP). This dynamic behavior was optimized by Algorithm for Pattern Extraction (ALOPEX) based on artificial neural network. However, they did not specify the actual sizes of low, medium, and high traffic.

Optimization simulation assumes finite buffer size but infinite number of buffers[6]. This makes it suitable for aggregating traffic, which is a standard focus of a switch. However this method was only suitable for intermediate or core switch, but unsuitable for edge switch, because it focuses on delivering the packets with minimum packet loss and minimum queue delay. Furthermore, this method did not take into consideration of the distribution of packets from the same transmission, and did not look at the end applications that need to access

specific transmission's packets. Moreover, the infinite number of buffers assumption may not be applicable to all types of switches. This is different with our research, where our research also optimizes the number of buffers. Another difference is that their work focuses on buffer size allocation, while ours is on allocation of packets to multiple buffers.

Dynamic Round-Robin (DRR) packet scheduling algorithm was modified[7]. Similar to the previous work[6], it was only applicable for intermediate or core network devices (switch, router, etc), because its objectives lie on queuing delay and output burst before forwarding the packets to the next link. This algorithm managed to reduce the packet queuing delay and output burst. It works by optimizing the number of allowed packets to be sent from each queue. This work[7] complements the previous DRR which was only suitable for long duration transmission.

## 3. Network Card Buffering Simulation

Our network card buffering simulation works according to the following rules:

- It represents PF_RING[8] packet buffering that provides either single buffer (per-flow mode) or round-robin (2 up to 6 buffers).
- Transmission size is taken into account, as it will be divided by the time constraints when the packets are finished being copied by the applications.
- Packets are distributed evenly to the available buffers, where every buffer has the same limited capacity e.g. 4096 bytes.
- Time constraints include: buffers creation time, duration for packets distribution, duration for applications to move from buffer to buffer, and total duration for all applications to finish accessing packets.

We show a simple case of two buffers (B1 and B2) filled with evenly distributed packets, which then be accessed by three applications (App1, App2, and App3) as depicted in Figure 1.

$$t = t_p \text{ x } n_p \quad (1)$$

From Figure 1, we present our buffering design, where every buffer access is considered as one round. Every buffer access is included in the same round if it occurs to different buffers at the same time. Therefore, we only have to calculate the access duration of one buffer to get the total access duration of all buffers.

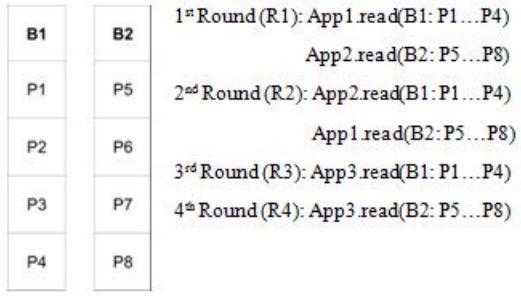


**Figure 1.** Buffering Design.

We can calculate the total access time ($t$) for R1 (Fig.1) by multiplying the access time per packet ($t_p$) which is 10ms with the number of packets ($n_p$) inside B1 which is 4 packets. In this case, the total access time for R1 is 10ms x 4 = 40ms.

Since there are four rounds and the total access time for each round is the same, because there are same number of packets in each buffer, thus the total access time for all rounds ($T$) can be generated by multiplying total access time per round ($t$) by the number of rounds ($r$) as depicted below.

$$T = t \times r \quad (2)$$

therefore in the example, the $T = 40ms \text{ x } 4 = 160ms$.

The next time constraints to be added are the buffers creation duration ($d_b$), packets distribution duration ($d_p$), and duration to switch applications from buffer to buffer ($d_a$). The equations are mentioned next.

$$d_b = t_b \text{ x } n_b \quad (3)$$

$$d_p = (t_p \text{ x } n_p) \text{ x } n_b \quad (4)$$

$$d_a = \sum_{1}^{Napp} t_s \text{ x } n_s \quad (5)$$

The symbols in Eq. 3, 4, and 5 are explained as follows:
$d_b$: total duration to create buffers
$t_b$: duration to create one buffer
$n_b$: number of buffers
$d_p$: total duration to distribute packets
$t_p$: duration to distribute one packet
$n_p$: number of packets
$d_a$: duration to switch applications from buffer to buffer

$N_{app}$: number of applications

$t_s$: duration to move application from one buffer to another

$n_s$: number of buffer switching of an application. The value of $n_s$ is always $n_b - 1$, because it represents the movements an application makes from its existing buffer to the rest of the buffers. For example, if there are 6 buffers, then an application has to make 5 movements, which are from its existing buffer to the rest of buffers (5 buffers).

Specifically, number of packets in each buffer ($n_p$) is calculated by the below equation:

$$n_p = \left(\frac{s}{p}\right) / n_b \tag{6}$$

The descriptions of symbols in Eq. 6 are below:

$s$: transmission size

$p$: packet size

Since the capacity of each buffer is limited, thus all buffers will be cleared and refilled until the transmission completes. The frequency of refilling buffers ($f$) is calculated so that we can get the ($T$) value for the whole transmission ($W_t$). The equation for ($f$) is below:

$$f = s / (b \times n_b) \tag{7}$$

$b$: buffer capacity

Hence, the ($W_t$) value can be generated as follows:

$$W_t = T \times f \tag{8}$$

Finally, the total transmission duration ($D$) is derived by Eq. 9:

$$D = d_b + (d_p \times f) + (d_a \times f) + W_t \tag{9}$$

In the next section, we present a mathematical model to represent simulation of applications' total access times to read the packets in buffers. We create our mathematical model based on the number of buffers in the network card. In our case, we take the architecture from [8] that can provide 1 up to 6 buffers. Our model's objective is to find the number of buffering rounds ($r$) by Eq. 2.

## 3.1 Buffering Model for Network Card with 1 Buffer

A network card with 1 buffer only allows 1 application to access the respective buffer. With multiple buffers, therefore they will access the buffer alternately one after another. It means that the number of rounds ($r$) equals the number of applications ($N_{app}$). The total access time ($T$) for all applications to finish reading the buffer is calculated by Eq. 10.

$$T = (t_P \times n_p) \times N_{app} \tag{10}$$

## 3.2 Buffering Model for Network Card with 2 Buffers

In the buffering model with 2 buffers, the mechanism of reading packets is done in round robin procedure, where every application accesses every buffer alternately. Within 1 round, there can only be 2 applications that access the packets in the 2 buffers. Therefore, if there are more than 2 applications, the applications will access different buffers alternately in every round until all the applications have read all buffers. The mechanism of buffering with 2 buffers is shown in the Table 1 below.

**Table 1.** Process of Reading Packets in 2 Buffers System

| |
|---|
| **2 Buffers and 1 Application** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| r = 2. |
| **2 Buffers and 2 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B1: P1…P4) |
| r = 2. |
| **2 Buffers and 3 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B1: P1…P4) |
| 3rd Round (R3): App3.read(B1: P1…P4) |
| 4th Round (R4): App3.read(B2: P5…P8) |
| r = 4. |
| **2 Buffers and 4 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B1: P1…P4) |
| 3rd Round (R3): App3.read(B1: P1…P4) |
| App4.read(B2: P5…P8) |
| 4th Round (R4): App3.read(B2: P5…P8) |
| App4.read(B1: P1…P4) |
| r = 4. |
| **2 Buffers and 5 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B1: P1…P4) |
| 3rd Round (R3): App3.read(B1: P1…P4) |
| App4.read(B2: P5…P8) |
| 4th Round (R4): App3.read(B2: P5…P8) |
| App4.read(B1: P1…P4) |
| 5th Round (R5): App5.read(B1: P1…P4) |
| 6th Round (R6): App5.read(B2: P5…P8) |
| r = 6. |

### 3.3 Buffering Model for Network Card with 3 Buffers

In the case of buffering model having 3 buffers, the process of reading packets is also accomplished in round robin mechanism, with every single round consisting of 3 buffers, accommodating maximum 3 applications that access different buffer alternately. The round will reoccur until all applications have read the 3 buffers. The procedure of buffering with 3 buffers is depicted in Table 2.

**Table 2.** Process of Reading Packets in 3 Buffers System

| **3 Buffers and 1 Application** |
|---|
| 1st Round (R1): App1.read(B1: P1…P4) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| r = 3. |
| **3 Buffers and 2 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| 4th Round (R4): App2.read(B3: P9…P12) |
| r = 4 |
| **3 Buffers and 3 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| r = 3 |
| **3 Buffers and 4 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| 4th Round (R4): App4.read(B1: P1…P4) |
| 5th Round (R5): App4.read(B2: P5…P8) |
| 6th Round (R6): App4.read(B3: P9…P12) |
| r = 6. |
| **3 Buffers and 5 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| 4th Round (R4): App4.read(B1: P1…P4) |
| App5.read(B2: P5…P8) |
| 5th Round (R5): App4.read(B2: P5…P8) |
| App5.read(B1: P1…P4) |
| 6th Round (R6): App4.read(B3: P9…P12) |
| 7th Round (R7): App5.read(B3: P9…P12) |
| r = 7. |

**Table 3.** Process of Reading Packets in 4 Buffers System

| **4 Buffers and 1 Application** |
|---|
| 1st Round (R1): App1.read(B1: P1…P4) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| r = 4 |
| **4 Buffers and 2 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B4: P13…P16) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B3: P9…P12) |
| r = 4. |
| **4 Buffers and 3 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| 5th Round (R5): App2.read(B4: P13…P16) |
| 6th Round (R6): App3.read(B4: P13…P16) |
| r = 6. |
| **4 Buffers and 4 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| App4.read(B4: P13…P16) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B4: P13…P16) |
| App4.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B4: P13…P16) |
| App3.read(B1: P1…P4) |
| App4.read(B2: P5…P8) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| App4.read(B3: P9…P12) |
| r = 4. |
| **4 Buffers and 5 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| App4.read(B4: P13…P16) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B4: P13…P16) |
| App4.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B4: P13…P16) |
| App3.read(B1: P1…P4) |
| App4.read(B2: P5…P8) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| App4.read(B3: P9…P12) |
| 5th Round (R5): App5.read(B1: P1…P4) |
| 6th Round (R6): App5.read(B2: P5…P8) |
| 7th Round (R7): App5.read(B3: P9…P12) |
| 8th Round (R8): App5.read(B4: P13…P16) |
| r = 8. |

## 3.4 Buffering Model for Network Card with 4 Buffers

The round robin process with 4 buffers serve maximum 4 applications in one round of packet reading. In the case of more than 4 applications, therefore the round repeats until all applications have read the entire buffers. The mechanism of this process is in Table 3.

## 3.5 Buffering Model for Network Card with 5 Buffers

In round robin with 5 buffers, maximum 5 applications can be accommodated in one round of packet reading/access, while in the case with more than 5 applications participating, the round reruns until all applications have read all buffers alternately. The processes of this round robin are listed in Table 4.

## 3.6 Buffering Model for Network Card with 6 Buffers

In the round robin packet buffering with 6 buffers, 6 applications are allowed to read/access every different buffer within one single round. Extra round(s) will be required if there are more than 6 applications needing to read/access the 6 buffers; in every round, the applications read/access different buffer alternately until they have finished reading all buffers. The procedures are listed in Table 5.

**Table 4.** Process of Reading Packets in 5 Buffers System

| |
|---|
| **5 Buffers and 1 Application** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| r = 5. |
| **5 Buffers and 2 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B4: P13…P16) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B3: P9…P12) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| 6th Round (R6): App2.read(B5: P17…P20) |
| r = 6. |
| **5 Buffers and 3 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B5: P17…P20) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| App2.read(B4: P13…P16) |
| 6th Round (R6): App3.read(B4: P13…P16) |
| 7th Round (R7): App3.read(B5: P17…P20) |
| r = 7. |
| **5 Buffers and 4 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| App4.read(B4: P13…P16) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B4: P13…P16) |
| App4.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B4: P13…P16) |
| App3.read(B1: P1…P4) |
| App4.read(B2: P5…P8) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| App4.read(B3: P9…P12) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| 6th Round (R6): App2.read(B5: P17…P20) |
| 7th Round (R7): App3.read(B5: P17…P20) |
| 8th Round (R8): App4.read(B5: P17…P20) |
| r = 8. |
| **5 Buffers and 5 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| App4.read(B4: P13…P16) |
| App5.read(B5: P17…P20) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B4: P13…P16) |
| App4.read(B5: P17…P20) |
| App5.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B4: P13…P16) |
| App3.read(B5: P17…P20) |
| App4.read(B1: P1…P4) |
| App5.read(B2: P5…P8) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B5: P17…P20) |
| App3.read(B1: P1…P4) |
| App4.read(B2: P5…P8) |
| App5.read(B3: P9…P12) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| App4.read(B3: P9…P12) |
| App5.read(B4: P13…P16) |
| r = 5. |

**Table 5.** Process of Reading Packets in 6 Buffers System

| |
|---|
| **6 Buffers and 1 Application** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| 6th Round (R6): App1.read(B6: P21…P24) |
| r = 6. |
| **6 Buffers and 2 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B4: P13…P16) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B3: P9…P12) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| App2.read(B6: P21…P24) |
| 6th Round (R6): App1.read(B6: P21…P24) |
| App2.read(B5: P17…P20) |
| r = 6. |
| **6 Buffers and 3 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B5: P17…P20) |
| App3.read(B6: P21…P24) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| App2.read(B6: P21…P24) |
| App3.read(B4: P13…P16) |
| 6th Round (R6): App1.read(B6: P21…P24) |
| App2.read(B4: P13…P16) |
| App3.read(B5: P17…P20) |
| r = 6. |
| **6 Buffers and 4 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| App4.read(B4: P13…P16) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B4: P13…P16) |
| App4.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B4: P13…P16) |
| App3.read(B1: P1…P4) |
| App4.read(B2: P5…P8) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| App4.read(B3: P9…P12) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| App2.read(B6: P21…P24) |
| 6th Round (R6): App1.read(B6: P21…P24) |
| App2.read(B5: P17…P20) |
| 7th Round (R7): App3.read(B5: P17…P20) |
| App4.read(B6: P21…P24) |
| 8th Round (R8): App3.read(B6: P21…P24) |
| App4.read(B5: P17…P20) |
| r = 8. |
| **6 Buffers and 5 Applications** |
| 1st Round (R1): App1.read(B1: P1…P4) |
| App2.read(B2: P5…P8) |
| App3.read(B3: P9…P12) |
| App4.read(B4: P13…P16) |
| App5.read(B5: P17…P20) |
| 2nd Round (R2): App1.read(B2: P5…P8) |
| App2.read(B3: P9…P12) |
| App3.read(B4: P13…P16) |
| App4.read(B5: P17…P20) |
| App5.read(B1: P1…P4) |
| 3rd Round (R3): App1.read(B3: P9…P12) |
| App2.read(B4: P13…P16) |
| App3.read(B5: P17…P20) |
| App4.read(B1: P1…P4) |
| App5.read(B2: P5…P8) |
| 4th Round (R4): App1.read(B4: P13…P16) |
| App2.read(B5: P17…P20) |
| App3.read(B1: P1…P4) |
| App4.read(B2: P5…P8) |
| App5.read(B3: P9…P12) |
| 5th Round (R5): App1.read(B5: P17…P20) |
| App2.read(B1: P1…P4) |
| App3.read(B2: P5…P8) |
| App4.read(B3: P9…P12) |
| App5.read(B4: P13…P16) |
| 6th Round (R6): App1.read(B6: P21…P24) |
| 7th Round (R7): App2.read(B6: P21…P24) |
| 8th Round (R8): App3.read(B6: P21…P24) |
| 9th Round (R9): App4.read(B6: P21…P24) |
| 10th Round (R10): App5.read(B6: P21…P24) |
| r = 10. |

## 3.7 Continuation of Number of Applications and Buffers Combination

The previous combinations of number of applications, buffers, and rounds were limited to 5 applications. But if proceed to higher number of applications, we will get the following results. In each column of number of buffers, the numbers consecutively represent number of applications and number of rounds (r) required.

**Table 6.** Relation of Number of Applications and Number of Rounds Required

| 2 Buffers | 3 Buffers | 4 Buffers | 5 Buffers | 6 Buffers |
|---|---|---|---|---|
| 1: 2 | 1: 3 | 1: 4 | 1: 5 | 1: 6 |
| 2: 2 | 2: 4 | 2: 4 | 2: 6 | 2: 6 |
| 3: 4 | 3: 3 | 3: 6 | 3: 7 | 3: 6 |
| 4: 4 | 4: 6 | 4: 4 | 4: 8 | 4: 8 |
| 5: 6 | 5: 7 | 5: 8 | 5: 5 | 5: 10 |
| 6: 6 | 6: 6 | 6: 8 | 6: 10 | 6: 6 |
| 7: 8 | 7: 9 | 7: 10 | 7: 11 | 7: 12 |
| 8: 8 | 8: 10 | 8: 8 | 8: 12 | 8: 12 |
| 9: 10 | 9: 9 | 9: 12 | 9: 13 | 9: 12 |
| 10: 10 | 10: 10 | 10: 12 | 10: 10 | 10: 14 |
| 11: 12 | 11: 12 | 11: 14 | 11: 15 | 11: 16 |
| 12: 12 | 12: 12 | 12: 12 | 12: 16 | 12: 12 |
| 13: 14 | 13: 15 | 13: 16 | 13: 17 | 13: 18 |
| 14: 14 | 14: 16 | 14: 16 | 14: 18 | 14: 18 |
| 15: 16 | 15: 15 | 15: 18 | 15: 15 | 15: 18 |

In Table 6 above, we can take an example from the fifth entry of the first column that states '5: 6'. It means that for 2 buffers with 5 applications, there will be 6 rounds required. It can be compared with the fifth entry in third column that mentions '5: 8', which means that for 4 buffers with 5 applications, 8 rounds are required.

## 4. Mathematical Models of Buffering Rounds

From the previous section of buffering simulation and Table 6, we can generate mathematical models for each number of buffers. The models will generate the required number of buffering rounds (r) as described below.

For 2 buffers:

$$r = \begin{cases} Napp & \text{if Napp mod } 2 = 0 \\ Napp + 1 & \text{otherwise} \end{cases} \quad (11)$$

For 3 buffers:

$$r = \begin{cases} Napp & \text{if Napp mod } 3 = 0 \\ Napp + 2 & \text{otherwise} \end{cases} \quad (12)$$

For 4 buffers:

$$r = \begin{cases} Napp & \text{if Napp mod } 4 = 0 \\ Napp + 2 & \text{if Napp mod } 2 = 0 \\ Napp + 3 & \text{otherwise} \end{cases} \quad (13)$$

For 5 buffers:

$$r = \begin{cases} Napp & \text{if Napp mod } 5 = 0 \\ Napp + 4 & \text{otherwise} \end{cases} \quad (14)$$

For 6 buffers:

$$r = \begin{cases} Napp & \text{if Napp mod } 6 = 0 \\ Napp + 3 & \text{if Napp mod } 3 = 0 \\ Napp + 4 & \text{if Napp mod } 2 = 0 \\ Napp + 5 & \text{otherwise} \end{cases} \quad (15)$$

## 5. Example Case

This section describes a buffering example. The transmission size ($s$), buffer capacity ($b$), and packet size ($p$) are assumed in a way that there are 8 packets ($n_p$) in the transmission and $f$ = 1. Furthermore, there are 3 applications ($Napp$), and the time required to read a packet is 10ms ($t_p$). The buffering illustrations are for 2 sample compared solutions. One with 2 buffers (B1 and B2) and another with 3 buffers (B1, B2, and B3). They are depicted in Figure 2 and Figure 3.

| B1 | B2 |
|---|---|
| P1 | P5 |
| P2 | P6 |
| P3 | P7 |
| P4 | P8 |

**Figure 2.** Buffering Illustration of Solution with 2 Buffers.

| B1 | B2 | B3 |
|---|---|---|
| P1 | P4 | P7 |
| P2 | P5 | P8 |
| P3 | P6 | |

**Figure 3.** Buffering Illustration of Solution with 3 Buffers.

In Figure 2 and Figure 3, the 8 packets (P1 until P8) are distributed among the buffers. Next in Table 7 below is the procedural calculation of total transmission duration ($D$) for both solutions. The mathematical models in section 4 are referred to find the number of rounds ($r$) accordingly.

**Table 7.** Comparison of 2 Different Solutions

| Steps | Solution A ($n_b$ = 2) | Solution B ($n_b$ = 3) |
|---|---|---|
| $T = t \times r$ | (10ms x 4) x 4 = 160ms | (10ms x 3) x 3 = 90ms |
| $d_b = t_b \times n_b$ | 40ms x 2 = 80ms | 40ms x 3 = 120ms |
| $d_p = (t_p \times n_{p)} \times n_b$ | (10ms x 4) x 2 = 80ms | (10ms x 3) x 3 = 90ms |
| $d_a = \sum_{1}^{Napp} t_s \times n_s$ | $\sum_{1}^{3}$ 20ms x (2-1) = 60ms | $\sum_{1}^{3}$ 20ms x (3-1) = 120ms |
| $D = d_b + (d_p \times f) + (d_a \times f) + W_t$ | 80 + (80 x 1) + (60 x 1) + (160 x 1) = 380ms | 120 + (90 x 1) + (120 x 1) + (90 x 1) = 420ms |

From the comparison above, solution A is the better one since it takes lesser time ($D$) to complete the receiving of data transmission. Therefore, we conclude that more

buffers are not necessarily the better solution. The optimal number of buffers mainly depends on the transmission size, packet size, number of packets, buffer capacity, and the number of applications.

## 6. Conclusion

Our mathematical models reflect the buffering process in network card with multiple buffers and multiple applications. These models can be useful for network card optimization setting, when the transmission characteristics are known.

Since our models use generic parameters, we expect them to be expandable with additional properties of network card buffers.

## 7. References


1. Gijsbers B, Akkoorath DD. Performance Simulation of Buffer Bloat in Routers. Technical Report. Faculty of Science, Universiteit van Amsterdam; 2011 Mar.
2. Odendahl M, Goens A, Leupers R, Ascheid G, Ries B, Vocking B, Henriksson T. Optimized Buffer Allocation in Multicore Platforms. Proc. Conference on Design, Automation and Test in Europe (DATE). 2014 Mar.p. 1–6. Crossref
3. Cohen A, Cohen R. A Dynamic Approach for Efficient TCP Buffer Allocation. IEEE Transactions on Computers. 2002 Mar; 51(3): 303–12. Crossref
4. Quintana AA, Casilari E, Lopez JH. An Integrated OMNeT++ Implementation of 802.11. Proc. International ICST Conference on Simulation Tools and Techniques (SIMUTools 2012), 2012 Mar.
5. Hasama M, Song Y, Ito T, Matsuno S. Optimization of Buffer-size Allocation Using Dynamic Programming. International Journal of Systems Applications, Engineering and Development. 2011; 5(4): 461–68.
6. Pandya AS, Sen E, Hsu S. Buffer Allocation Optimization in ATM Switching Networks Using ALOPEX Algorithm. Neurocomputing. 1999 Feb; 24(1-3): 1–11. Crossref
7. He Y, Gao L, Liu GK, Liu YZ. A Dynamic Round-Robin Packet Scheduling Algorithm. Applied Mechanics and Materials. 2013 Aug; 347-350: 2203–07. Crossref
8. NTOP. PF_RING User Guide: Linux High Speed Packet Capture. 2012.