Improving queueing implementation on high speed switches

Wendy Hernandez
University of Texas-Pan American

IMPROVING QUEUEING IMPLEMENTATION ON HIGH SPEED SWITCHES

A Thesis

by

WENDY HERNANDEZ

Submitted to the Graduate School of
The University of Texas-Pan American
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2015

Major Subject: Electrical Engineering

IMPROVING QUEUEING PERFORMANCE ON HIGH SPEED SWITCHES

A Thesis
by
WENDY HERNANDEZ

COMMITTEE MEMBERS

Dr. Sanjeev Kumar
Chair of Committee

Dr. Jae Son
Committee Member

Dr. Wenjie Dong
Committee Member

May 2015

ABSTRACT

Hernandez, Wendy, Improving Queueing Implementation on High Speed Switches. Master of Science (MS), May, 2015, 85 pp., 77 figures, references, 45 titles.

In this thesis two different conventional shared memory allocation schemes - Dynamic Threshold (DT) and Threshold-based Filtering (TF) - are evaluated under varied traffic conditions in order to determine the optimal configuration for each tested scenario. The effect that a changing ABL, load, and ratio between buffer size and ports have on the packet loss is observed for buffer sharing schemes DT and TF schemes. This allowed to easily determining the Alpha and Thresholds required by DT and TF schemes respectively to obtain an optimal configuration under each of the different tested scenarios.

A new shared memory allocation scheme referred to in this thesis as ‘Shortest Queue First Lite’ (SQFL) scheme is evaluated. SQFL scheme aims at decreasing the complexity of SQF in order to facilitate its hardware implementation. Comparisons are drawn between SQFL, SQF, DT and TF in terms of packet loss ratio.

DEDICATION

First and most foremost, I would like to thank God for giving the knowledge, which allowed me complete this thesis. Gracias a mis padres por su amor infinito y apoyo a lo largo de mi vida. Gracias ambos, Miguel y María Hernández, por darme la fuerza y enseñarme que “querer es poder”. My sisters and little brothers deserve heartfelt thanks as well. Also I big thank you to my husband Javier Castillo for his unconditional love and support, thank you for always pushing me to give the best of me, all your help and encouragement are strong factors that made the completion of my thesis possible. To all my friends, thank you for your understanding and motivation. Last but not least, I want to thank Mr. Jorge Garcia and Mrs. Sandoval for keeping in touch after I graduated from high school, I wouldn’t be here today if it wasn’t for the two of you, always telling me I could do it. The sky is the limit!

ACKNOWLEDGEMENTS

I will always be grateful to Dr. Sanjeev Kumar, chair of my thesis committee, for all his mentoring and advice. From NSF funding, research design, and data processing, to manuscript editing, he encouraged me to complete this process through his infinite patience, guidance, encouragement and advice provided throughout my time as his student. I have been very lucky to have a mentor always interested in knowing about my progress and who always responded all my questions so promptly.

My thanks go to my thesis committee members: Dr. Jae Son, and Dr. Wenjie Dong. Their advice, input, and comments on my dissertation helped guarantee the quality of my academic work.

I would also like to thank my colleagues at the UTPA library who helped me locate supporting books and documents for my research.

Work in this thesis is supported in part by NSF's Major Research Instrumentation Program Award #CNS-1040430.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In the very short period of time since the Internet came to exist we have become witnesses of its accelerated evolution [1]. The Internet has changed our lives by providing us with limitless information readily available, literally at the tip of our fingers. The Internet has come to influence the way we act, the way we learn, the way we conduct business, the way we experience consumable media, and so many other things[2] [3].

The Internet is ever-changing and ever-growing, and the speed at which changes occur is accelerating as the years come [4]. Internet traffic has been growing exponentially, and in Cisco's recent paper it is shown that this trend is likely to continue for the foreseeable future [5]. In Cisco's visual networking index it is studied and presented to us that the growth Internet for the last 2 decades. In 1992, global Internet networks carried approximately 100 Gigabytes of data per day; in 1997 that amount had already increased to 100 Gigabytes per hour; by the year 2013 the amount of data being transmitted over the internet had reached a staggering 28,875 Gigabytes per second, and it is predicted that by the year 2018 those numbers will be reaching upwards of 50,000 Gigabytes per second [6].

Similarly, the global mobile data traffic saw a growth of 81% just in the year 2013 alone, and it is expected to continue growing thanks to the ubiquity of mobile devices such as smartphones and tablets. It is predicted that by the end of the year 2014 the number of mobile devices connected to the Internet will exceed the world's population [7]. As stated in the famous

Moore’s law the speed and processing power of computers will double every 18 to 24 months, and advances in technology now a days have an almost direct correlation to easier access to the Internet [5] [8], which in turn translates into higher internet traffic.

Circuit Switching and Packet Switching

In circuit switching a dedicated connection has to be established between a sender and receiver before any transmission occurs; this connection should be able to last as long as the entire conversation lasts, at each node the incoming data are sent through the most suitable path without any delay [9] [10]. Circuit switching is employed mostly on telephone technology, since the telephone service provides a dedicated connection between two telephones [11].

Packet switching is an efficient way of transmitting data from one point to another, that unlike circuit switching, doesn’t establish a path between the sender and the receiver, instead it uses statistical multiplexing were communication from multiple users competes for the use of a shared media [11]. In packet switching the data is divided into smaller packets and each one of those packets is sent separately. Packets need to be able to reach the correct destination as well as in the right order; therefore, each single packet should have an address destination, SYN flag, Frame Check Sequence, and a sequence number [12]. The address destination tells the network to what port the packet should be deliver, SYN flag is for synchronization pattern, Frame Check sequence assures that there has not been an error in the sequence of packets, and last but not least we have the sequence number which helps us to reassemble the message that was divided into smaller packets [9] [10].

Routers and Switches

Routers

A router is an electronic device that operates at layer three of the Open System Interconnection (OSI) architecture which function is to link computers to the internet and enable users to share a connection by interconnecting multiple networks together. This device consists of a processor, a memory and an independent I/O interface for each network it is a part of; because of this a router requires the use of more than one IP address, more specifically, one IP address for each of the networks the router is a part of. The sole task of a router is to forward packets from their destination network to their respective final destination network -most of the times requiring the packets to go through more than one router or hop- through a calculated shortest path stored in its routing table. A router effectively acts as a postal sorting office, determining the best route for a packet based on its destination and the information stored in its routing table [11] [13].

Switches

Switches are hardware devices that connect multiple computers over a Local Area Network (LAN); they work on layer 2 or data-link layer of the OSI Seven Layer Reference Model, which uses MAC addresses. Switches are composed of several ports, each of them potentially connected to a single computer, allowing these computers to send frames between one another. A switch is capable of sending frames in two different ways; frames can be either forwarded or broadcast. Each time a new computer is connected to the network the switch broadcasts a message, so that every other computer knows of the existence of the device being just added. On the other hand we have frames being only forwarded from one source computer to its destination computer. Frames are only forwarded when the destination address is already known. Switches are very sophisticated devices that incorporate memory to help them be more efficient. The memory will prevent the loss of packets whenever two incoming packets have the same output destination [14] [15].

For instance when two packets come into the switch from two different input ports, but these packets share as destination the same output port. Since the throughput of the switch is limited by the speed of the line, which means that of the two arriving packets only one of them is able to be outputted, while the other packet would have to be dropped. The addition of memory mitigates the amount of packets lost in these scenarios, thus playing an important role in reducing the excess traffic generated by retransmission of packets due to drop in the switch.

Buffering Strategies

Input Buffered Switches

This type of switch is composed of a set of individual buffers, each one assigned to a single input port as shown in figure 1. This input buffer mitigates packet loss, since in the case when multiple packets sharing as destination the same output port arrive on the switch, only one of these packets will be launched into the fabric, while the rest will be stored at their respective input port of arrival [16]. Because each input port is coupled to a single independent buffer, the required bandwidth for these buffers is two times the *L* line speed. The introduction of input buffers creates a new problem however, known as Head-of-Line (HoL) blocking, which causes the throughput to be limited to 0.586 [11] [17].

HoL blocking is a problem present only in input buffered switches due to the First-In-First-Out (FIFO) nature of the queues. HoL blocking occurs when at the top of the queues there are more than one packets destined to the same output port, only one of those packets will enter the fabric and be outputted through its respective port, the rest of the packets will remain in their queues, blocking the next packets in line from entering the fabric and being outputted. HoL blocking may be exemplified as a case in which at the top of the queues there are packets going to ports 1, 2 and 3; however there are two packets going to port 2, if the packet queued in port 1 is launched into the fabric, then the packet at the top of the queue on port 2 creates a blockage, because even though the second packet in line could be sent through the fabric, it is unable to due to the FIFO nature of the queue [18].

A number of ways to mitigate or eliminate HoL blocking have been proposed such as Virtual Output Queuing (VOQ) [19] [20] [21], where the input buffers are organized as a set of queues where packets are stored according to their destination output port; each input port containing a separate buffer allocated for each of the switch output ports. This however provides very little scalability, since the buffer size required grows quadratically as the number of ports increases.

In order to improve upon VOQ, Destination Based Buffer Management (DBBM) is proposed. Here packets are also stored in queues sorted by destination, but unlike VOQ, DBBM contains a fewer number of queues than the number of ports in the switch [22]. DBBM is characterized by 4 parameters [23]: queue sharing--indicates whether packets should be allowed to be stored in the same queue, when no space is available in the buffers for arriving packets, then these are stored in an auxiliary buffer called overflow buffer--, mapping function--computes the queue where incoming packet will be stored--, replacement--indicates whether packets may be removed from a queue when arriving packets request allocation in it--, restoration--indicates whether packets in the overflow buffer may be allowed to be allocated in a normal queue as space becomes available.

Dynamic DBBM [24] improves upon DBBM by providing a more fair allocation of memory through the enabling of the switch to transmit a congestion notification back to the transmitting source, which will in term transmit packets with a congestion bit-flag on, enabling the switch to separate congested from non-congested traffic. Congested packets are allocated in a special buffer called dynamic queue, while non-congested packets are allocated on the remaining queues called DBBM queues according to the before mentioned DBBM scheme.

Regional Explicit Congestion Notification (RECN) [25] completely eliminates HoL blocking while requiring minimal resources, making it efficient, scalable and cost effective. RECN identifies congested flows and places them in a special dynamically assigned Set Aside Queue (SAQ) located at each of the input ports, while standard queues hold non-congested traffic. Congestion is detected by setting a threshold for every input queue, whenever this threshold is reached, a SAQ starts allocating new incoming packets, and whenever this newly allocated SAQ becomes full a new SAQ allocates the nest set of incoming packets.

Burst-Aware HoL-Blocking Injection Avoidance (BAHIA) [26] dynamically detects bursty traffic in the network and isolates it ensuring non-bursty traffic is unaffected, thus mitigating HoL blocking that may be produced by long bursts of data.

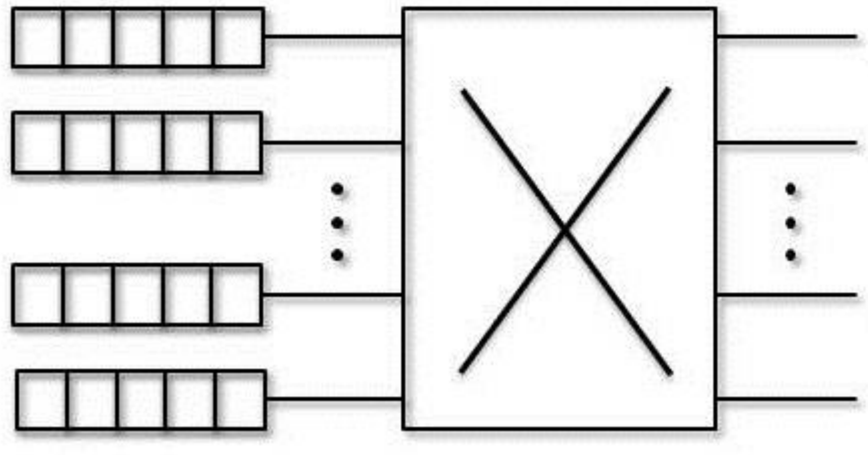


Figure 1-Input Buffered Switch

Output Buffered Switches

This type of switch, much like input buffer, contains an individual buffer associated to each output port as shown in figure 2 [27]. Output buffer switches, unlike input buffer switches, do not suffer from HoL blocking, it is however very susceptible to bursty traffic [28], this is because in the case when there a several streams of data arriving at different input ports destined to the same output port, there is a very high chance of this particular buffer being overflown, after which incoming packets will begin being dropped, regardless of the amount of memory left unused by the queues at the remaining output buffers, effectively wasting available memory on the switch, resulting in these particular scenarios on a higher packet loss ratio than input buffer switches.

Since on the worst case scenario any given output buffer may be required to store an incoming packet from all input ports the minimum memory speed required, assuming the switch is composed of *N* ports, with each port supporting *L* line speed, is *L (N + 1)* [29].

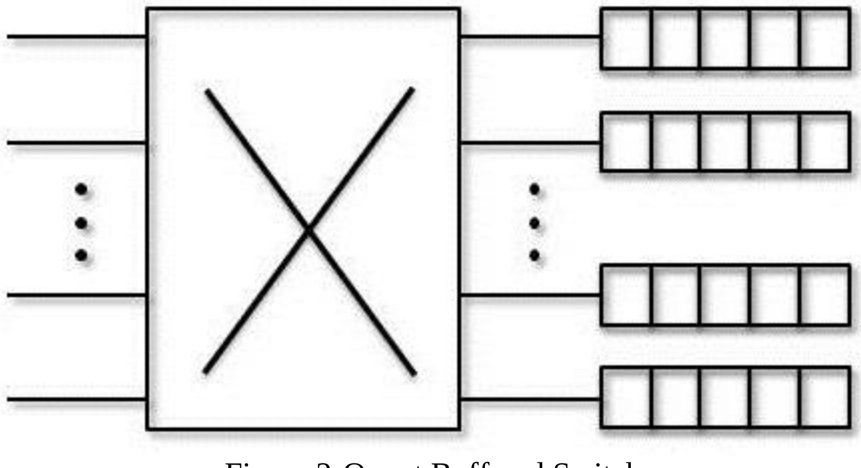


Figure 2-Ouput Buffered Switch

Shared-Memory Switches

Shared memory switches are similar to output buffer switches in that their buffers are located on the output ports, as shown in figure 3. They however differ in the utilization of the utilization of their output buffers; where output buffer switches have separate memory modules to allocate each individual output port queue, shared memory switches consist of a single memory unit in which all queues are allocated [30]. Allocation of incoming packets is handled in a first come first served basis, meaning that regardless of the output port incoming packets are destined to, they will be admitted provided there is available space on the buffer, otherwise they will be dropped [31].

Sharing the available memory amongst all output ports results in a more efficient utilization of resources; this same characteristic however poses a disadvantage in itself, as the number of reads and writes required for the memory to handle increases twofold resulting in a minimum required memory bandwidth equal to $L * (2N)$, increasing costs and reducing scalability [32].

Another downside of shared memory switches is the lack of fairness, especially under bursty traffic conditions. In order to explain this lets think of the worst case scenario, in which all arriving packets are destined to the same output port for a period of time long enough so that all available memory in the switch is allocated to the same queue, in the event a packet destined to a different port arrives at the switch it will most likely be dropped due to lack of space in memory to be allocated.

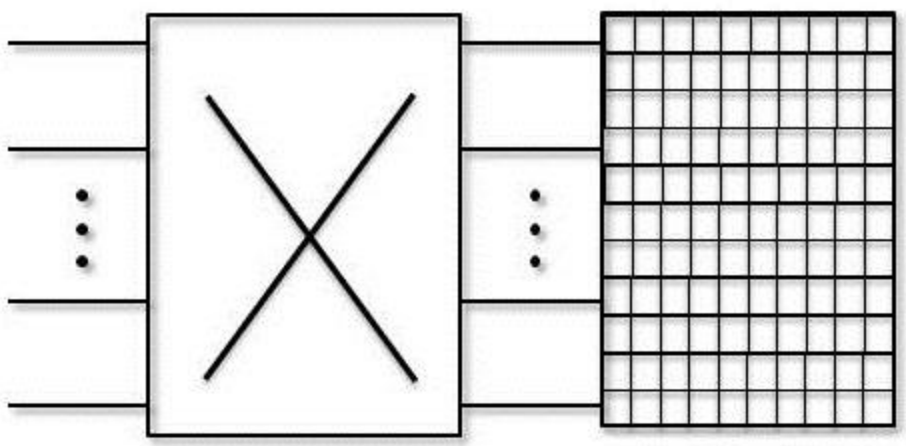


Figure 3-Shared-Memory Switch

Motivation and Problem Statement

As demonstrated in [33], Shortest Queue First (SQF) is a sharing memory scheme that presents a better performance in terms of packet loss ratio than existing sharing memory schemes such as Shared with Maximum Queue lengths (SMXQ), Shared with Minimum Allocation (SMA), as well as Dynamic Queue length Thresholds (DT). Upon closer inspection however, it starts to become evident that in order to realize SQF a largely complex hardware would be required due to the necessity to continuously sort its priority list utilized in order to insure that incoming packets always get stored first on the queues with the smallest amount of packets. This thesis attempts to solve that problem by simplifying SQF enough so that implementation becomes plausible, while maintaining the signature feature of SQF, giving greater priority to queues with the shortest length and allowing packets being destined to these ports to be stored first. The proposed scheme dubbed SQFL for SQF Lite is described in detail in Chapter III.

Thesis Outline

Chapter I serves as an introductory chapter, providing explanation for a series of basic concepts in an effort to facilitate the comprehension of the materials covered in Chapter II, Chapter III, and Chapter IV, as well as defining the propose of the thesis. Chapter II contains the detailed descriptions of the simulated conventional sharing buffer schemes, as well as for the SQF scheme. Chapter III provides the detailed explanation of the operation of the proposed scheme SQFL, as well as the importance and algorithms of different sorting mechanisms. Chapter IV provides in-depth knowledge regarding how data was obtained, including descriptions of the traffic model utilized, how the buffer available in the switch was simulated, the scenarios under which the sharing memory schemes were tested, and the use of the cluster computer in order to speed up the process of gathering of data. Chapter V aims to determine the optimal configurations for the schemes Dynamic Threshold (DT) and Threshold-Based Filtering (TF) under each of the simulated scenarios. Chapter VI draws performance comparisons in terms of packet loss ratio first between both conventional schemes, then between the conventional schemes and the proposed scheme SQFL. In Chapter VII we test and evaluate the proposed SQFL) versus SQF, as well as show the number of comparison required by SQFL and SQF to sort a list using three different sorting algorithms. In Chapter VIII we conclude this thesis and suggest possible future work.

CHAPTER II

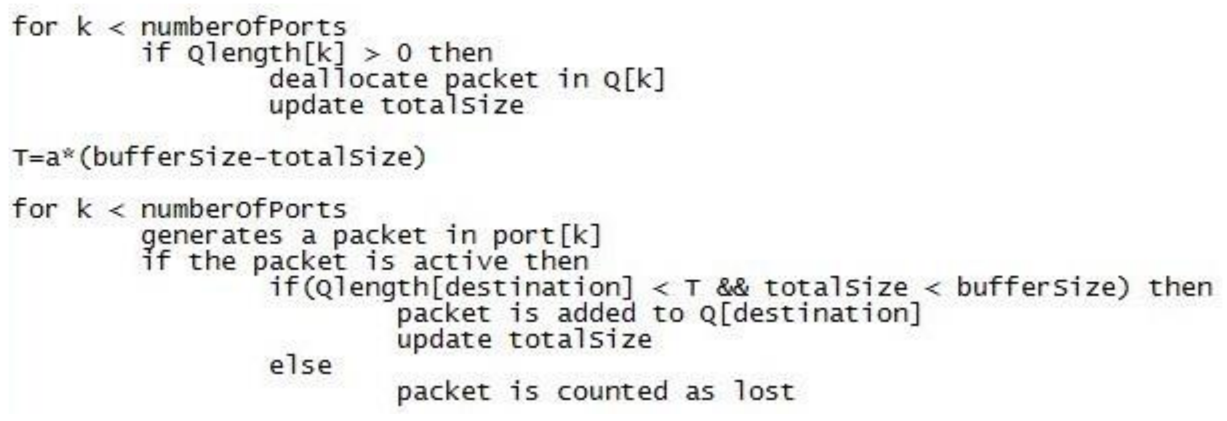
CONVENTIONAL MEMORY SCHEMES AND SQF

In this chapter we will describe three conventional schemes: Dynamic Threshold, Threshold Based Filtering and Shortest Queue First, which were implemented in this thesis. We will first explain the way each scheme works in detail, we will then follow with a pseudocode of the algorithm, as well as a flowchart of the code utilized for the simulation of these schemes.

Dynamic Threshold (DT)

Dynamic Threshold (DT) has one of the best ways to make an efficient use of the memory available for buffering in a switch. In DT, whenever one output port is active, the scheme will try to optimize the usage of memory by allowing it to use all the memory needed as long as the dynamic control threshold is not exceeded. If the length of a queue reaches the control threshold, any packets going to that queue will be dropped during that time slot, in a sense giving a higher priority to smaller queues. The control threshold $T(t)$ is equal to the unused buffer space times a user defined multiple α, and it can be calculated with the formula $T(t) = \alpha \cdot (B - Q(t)) = \alpha \cdot (B - \sum i\ Qi(t))$; where B represents the total available buffer in the switch, $Q(t)$ represents the sum of all the queue lengths, and $Qi(t)$ represents the length of queue i.

A new control threshold is calculated on every time slot, allowing DT to adapt to changing traffic conditions. The control threshold may be easily calculated if α is equal to a power of two, which would only require the use of a shifter to implement. However, the use of DT does not allow for a complete allocation of the memory available. The equation that give us the amount of unused space in memory is given by *(B – S) / (1 + α · M)*; where S represents the space occupied by the queues below the control threshold, and M represents the number of heavily active queues [34] [35] [36]. The algorithm for the DT sharing scheme is shown in figure 4, and a flowchart for the operation of the scheme's simulation is shown in figure 5.



```
for k < numberOfPorts
        if Qlength[k] > 0 then
                deallocate packet in Q[k]
                update totalSize

T=a*(bufferSize-totalSize)

for k < numberOfPorts
        generates a packet in port[k]
        if the packet is active then
                if(Qlength[destination] < T && totalSize < bufferSize) then
                        packet is added to Q[destination]
                        update totalSize
                else
                        packet is counted as lost
```

Figure 4-Algorithm for DT scheme

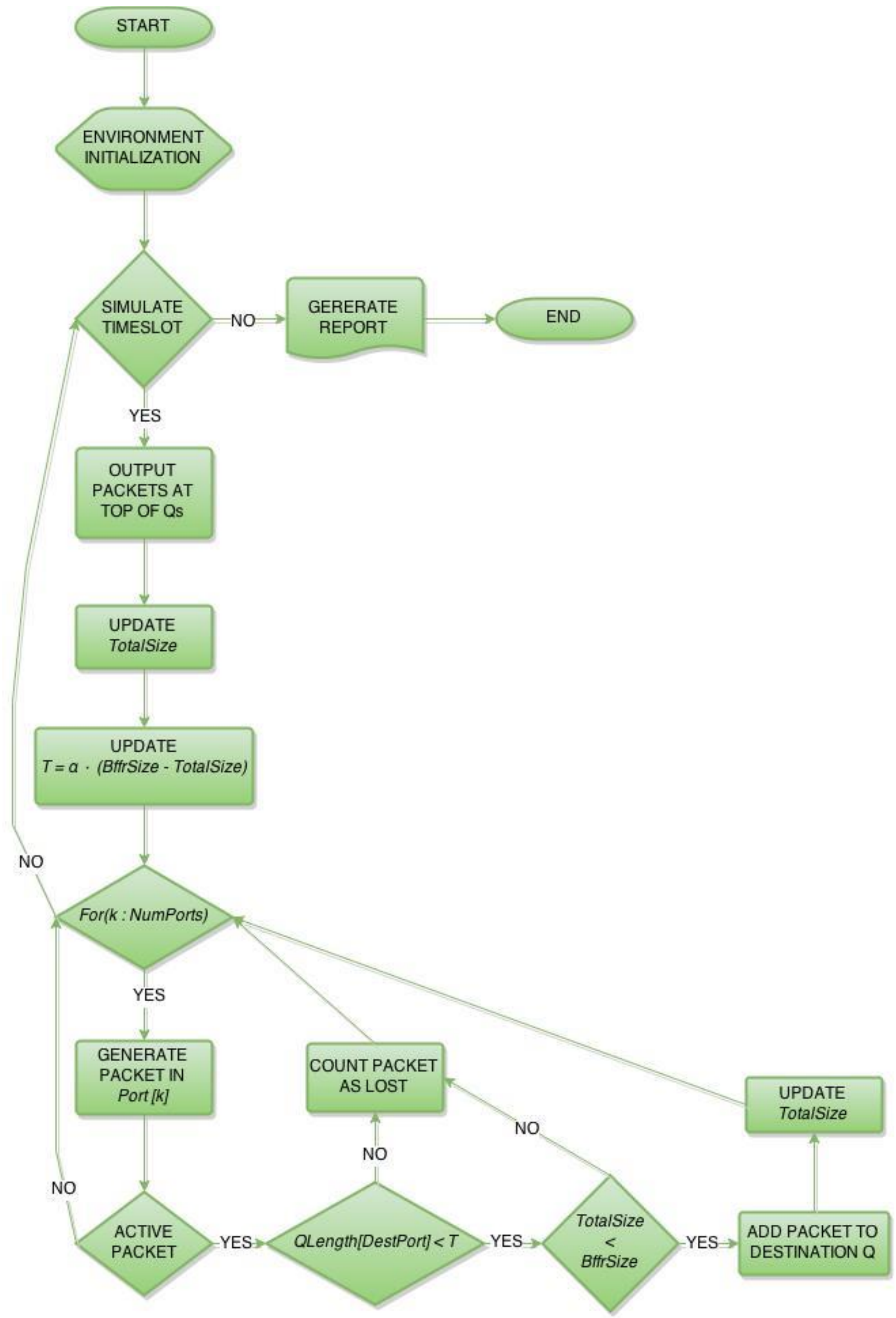


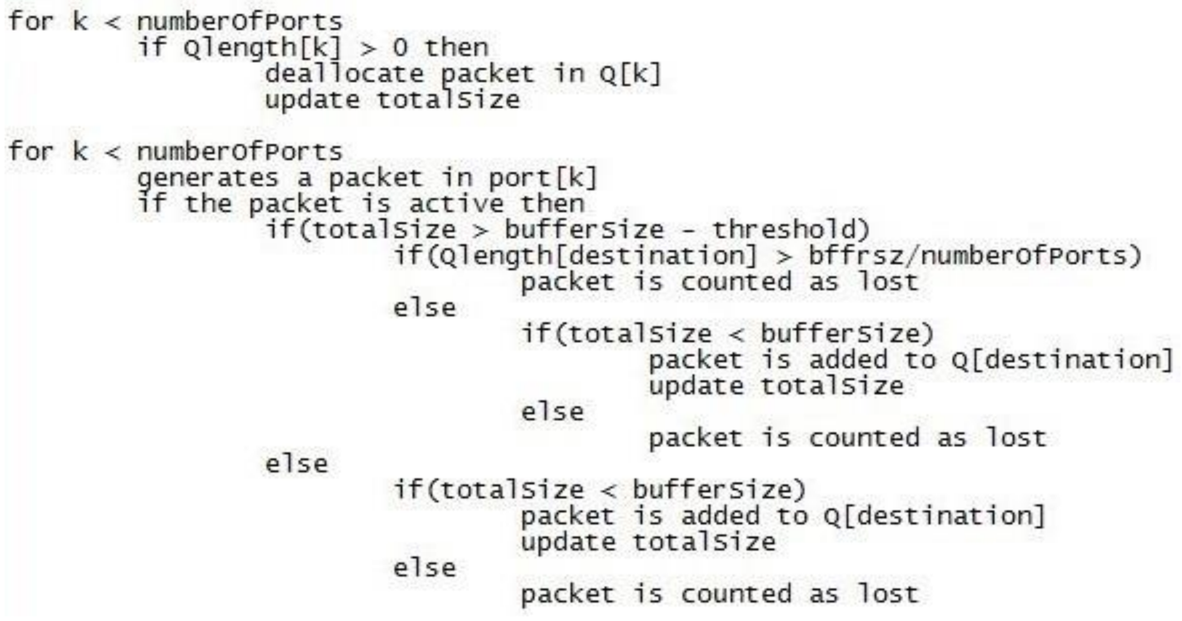
Figure 5-Flowchart for DT scheme

Threshold-Based Filtering (TF)

Threshold-based Filtering (TF) aims at making a more efficient use of the shared buffer while providing a fair distribution of the available resources. In this scheme each port is to be considered in one of two states: active or inactive. A port is considered to be active if its queue length is larger than the dedicated buffer allocation factor *B/N*; where *B* represents the total buffer available for the whole switch, and *N* represents the number of output ports. A port is considered to be inactive if its queue length does not exceed *B/N*.

The switch, like the ports, also has two different states: overloaded and non-overloaded. The switch will be considered to be overloaded whenever the total queue length *Q(t)* is larger than *B-T*; where *T* represents the threshold factor imposed on the switch. The switch will be considered to be non-overloaded if the total queue length *Q(t)* does not exceed *B-T*.

TF operates in such a way that when the state of the switch is non-overloaded there are no restrictions for the incoming packets; every single arriving packet will be allocated in the buffer regardless of whether its destination port is active or inactive. However, when the state of the switch is overloaded only packets destined to inactive ports will be allocated in the buffer; every packet with an active port as destination will be dropped [37] [38]. The algorithm and flowchart for the operation of the TF scheme are shown in figures 6 and 7 respectively.



```
for k < numberOfPorts
        if Qlength[k] > 0 then
                deallocate packet in Q[k]
                update totalSize

for k < numberOfPorts
        generates a packet in port[k]
        if the packet is active then
                if(totalSize > bufferSize - threshold)
                        if(Qlength[destination] > bffrsz/numberOfPorts)
                                packet is counted as lost
                        else
                                if(totalSize < bufferSize)
                                        packet is added to Q[destination]
                                        update totalSize
                                else
                                        packet is counted as lost
                else
                        if(totalSize < bufferSize)
                                packet is added to Q[destination]
                                update totalSize
                        else
                                packet is counted as lost
```

Figure 6-Algorithm for TF scheme

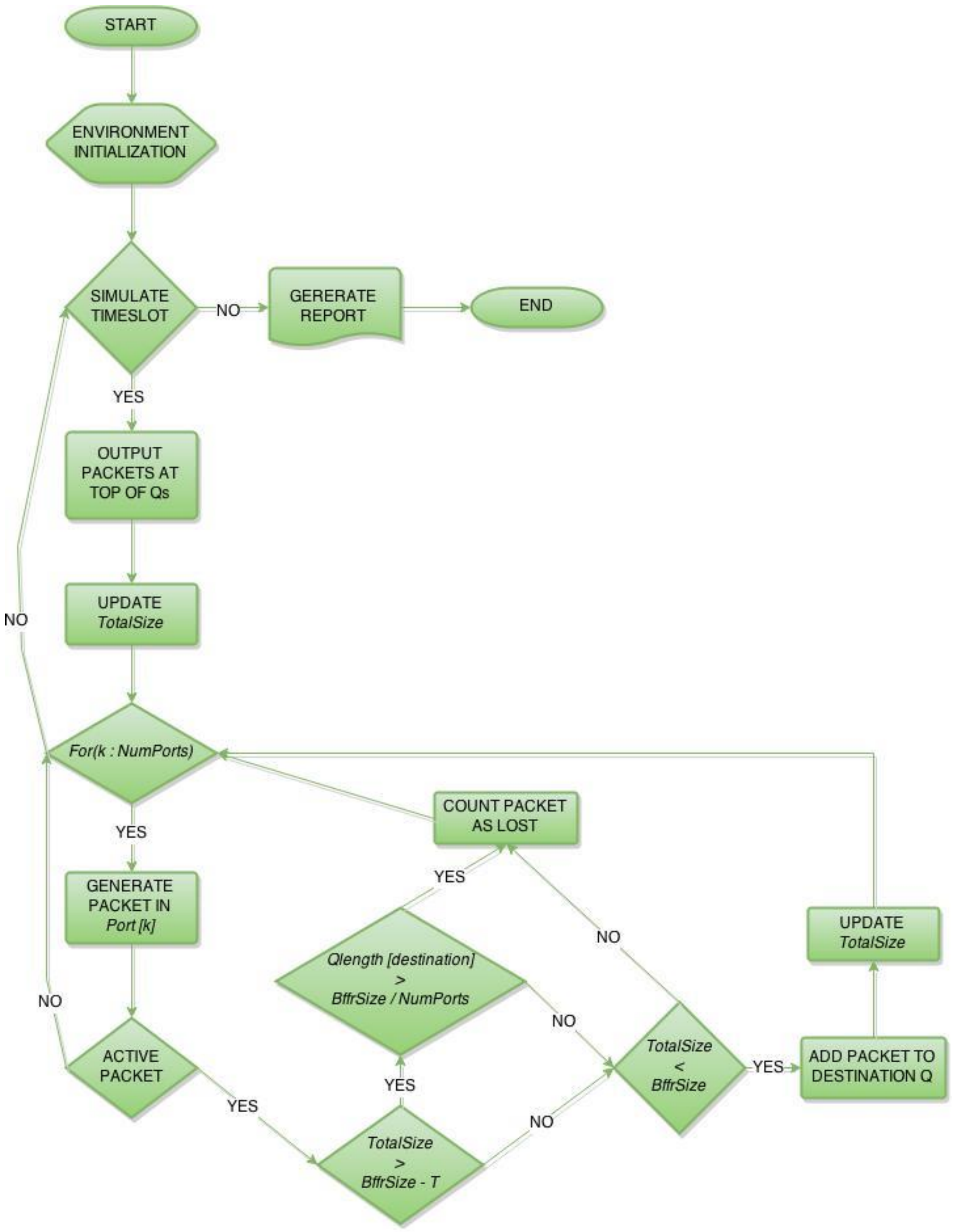
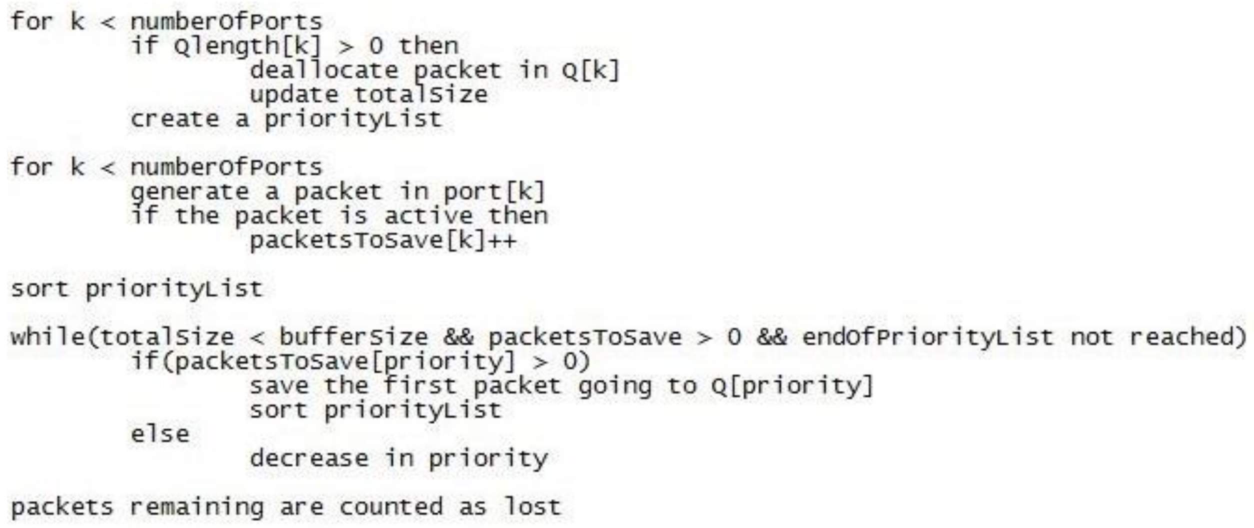

START
ENVIRONMENT INITIALIZATION
SIMULATE TIMESLOT
NO
GERERATE REPORT
END
YES
OUTPUT PACKETS AT TOP OF Qs
UPDATE *TotalSize*
NO
For(k : NumPorts)
YES
COUNT PACKET AS LOST
GENERATE PACKET IN *Port [k]*
YES
NO
UPDATE *TotalSize*
Qlength [destination] > BffrSize / NumPorts
NO
NO
ACTIVE PACKET
YES
TotalSize < BffrSize
YES
ADD PACKET TO DESTINATION Q
YES
NO
TotalSize > BffrSize - T

Figure 7-Flowchart for TF scheme

Shortest Queue First (SQF)

Shortest Queue First is a buffering scheme where all memory available to the switch is shared. In SQF a list of every output port queue is kept sorted from smallest queue to longest queue. When the switch receives incoming packets they are stored one packet at a time, constantly updating its sorting list in order to ensure that packets going to the shortest queue are stored first. When no more packets received in that particular time slot are destined to the shortest queue, SQF proceeds to move on to storing packets on the second shortest queue and so on until there are no more packets to store or no more memory to store packets in.

Unlike other schemes SQF does achieve full occupancy of the memory available, while at the same time maintaining an unmatched level of fairness. This however comes at great cost, requiring the list of queues to be constantly sorted – every time a packet is stored – in order to ensure that packets are at all times stored in the shortest queue first [33]. Figures 8 and 9 show the algorithm and flowchart of the operation of the SQF scheme.



```
for k < numberOfPorts
        if Qlength[k] > 0 then
                deallocate packet in Q[k]
                update totalSize
        create a priorityList

for k < numberOfPorts
        generate a packet in port[k]
        if the packet is active then
                packetsToSave[k]++

sort priorityList

while(totalSize < bufferSize && packetsToSave > 0 && endOfPriorityList not reached)
        if(packetsToSave[priority] > 0)
                save the first packet going to Q[priority]
                sort priorityList
        else
                decrease in priority

packets remaining are counted as lost
```

Figure 8-Algorithm for SQF scheme

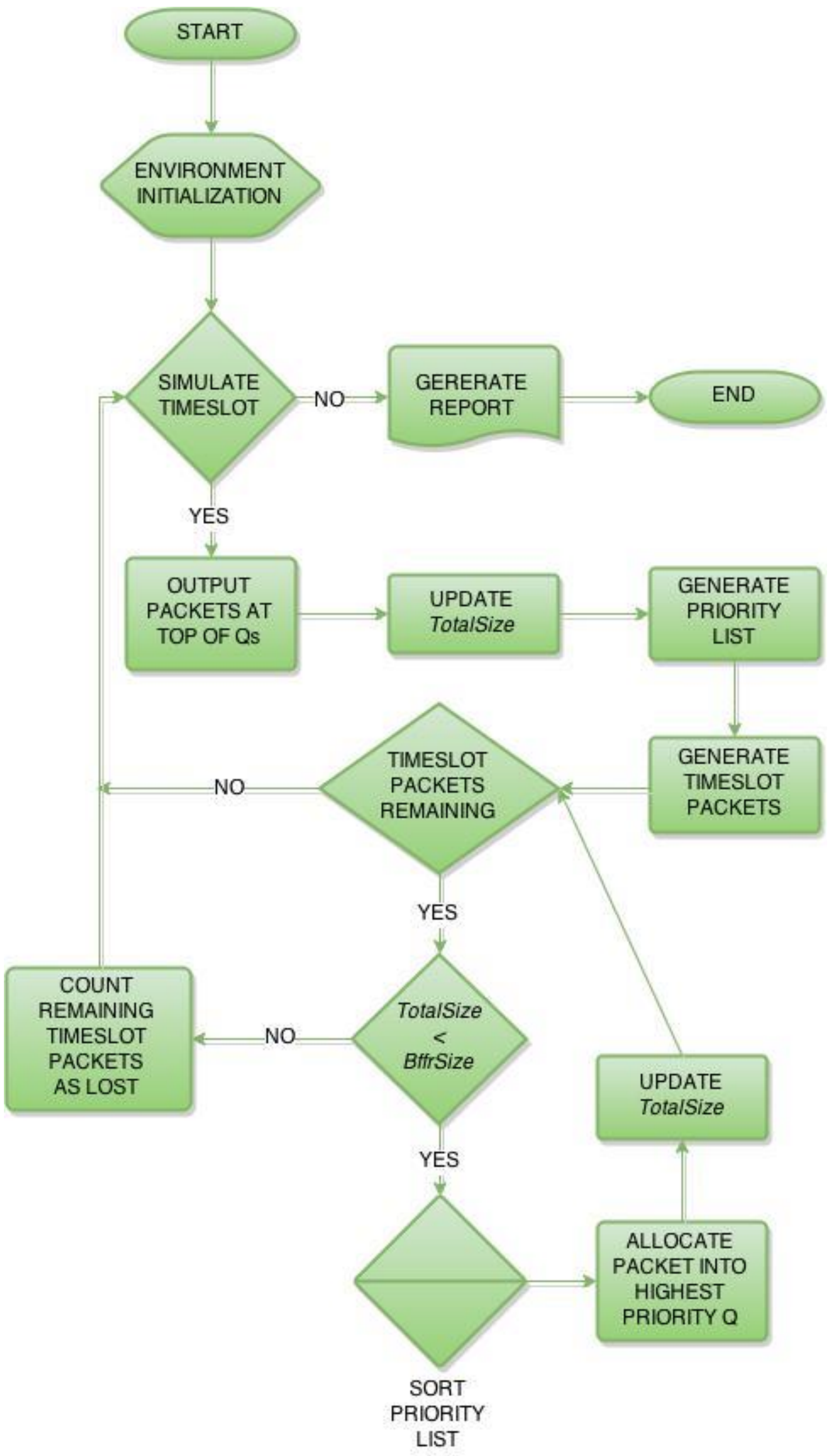

START
ENVIRONMENT INITIALIZATION
SIMULATE TIMESLOT
NO
GERERATE REPORT
END
YES
OUTPUT PACKETS AT TOP OF Qs
UPDATE *TotalSize*
GENERATE PRIORITY LIST
GENERATE TIMESLOT PACKETS
TIMESLOT PACKETS REMAINING
NO
YES
COUNT REMAINING TIMESLOT PACKETS AS LOST
NO
TotalSize < *BffrSize*
YES
UPDATE *TotalSize*
ALLOCATE PACKET INTO HIGHEST PRIORITY Q
SORT PRIORITY LIST

Figure 9-Flowchart for SQF scheme

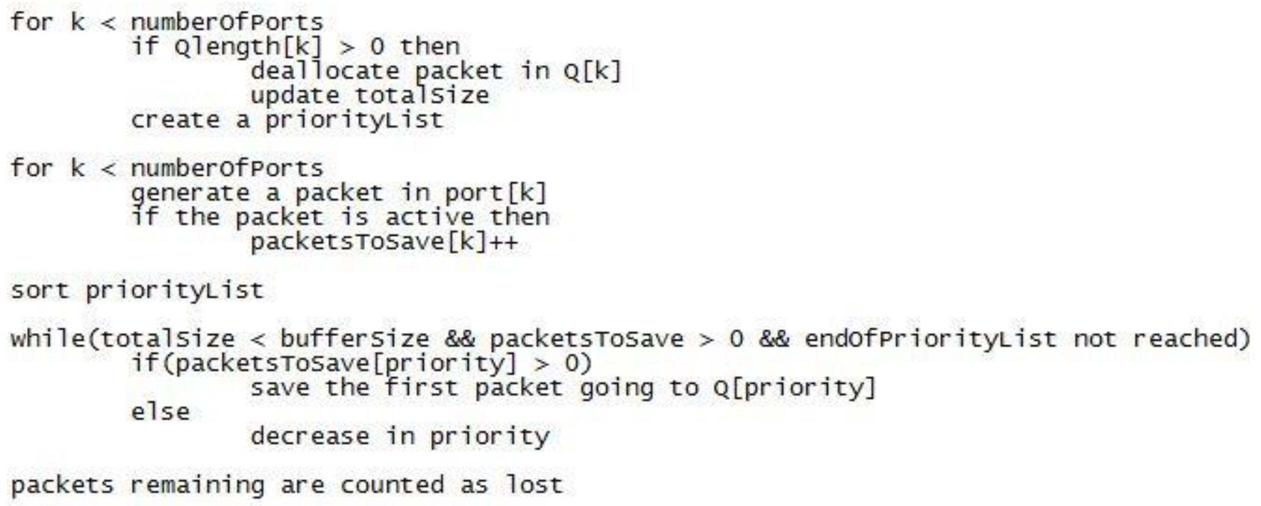
CHAPTER III

THE PROPOSED SCHEME

In this chapter we will analyze the sorting algorithm used on the implementation of the SQFL Scheme, and how the performance of the scheme can be improved by employing a more advance sorting algorithm.

Shortest Queue First Lite (SQFL)

Shortest Queue First Lite is a version of SQF derived from the idea to make its implementation more plausible. SQFL follows the same basic principle of maximizing fairness in the memory distribution amongst ports in a switch, but using a slightly more lax approach. Like SQF, SQFL maintains a list of all its output ports sorted from smallest queue to largest queue. Unlike SQF, SQFL grabs all packets destined to the smallest queue and stores them, it then moves on the second smallest queue, and so on until either there are no more packets to store or no more available memory to stores the packets in. When the storing process is completed, and only then, SQFL updates and sorts the new priority table, thus requiring only one update per time slot versus the many updates to the priority list required by SQF. Reducing the complexity of the SQF scheme makes the implementation of SQFL more feasible, as well as allows for the reduction of manufacturing costs. The algorithm and flowchart for the operation of the SQFL scheme are shown in figures 10 and 11 respectively.



```
for k < numberOfPorts
        if Qlength[k] > 0 then
                deallocate packet in Q[k]
                update totalSize
        create a priorityList

for k < numberOfPorts
        generate a packet in port[k]
        if the packet is active then
                packetsToSave[k]++

sort priorityList

while(totalSize < bufferSize && packetsToSave > 0 && endOfPriorityList not reached)
        if(packetsToSave[priority] > 0)
                save the first packet going to Q[priority]
        else
                decrease in priority

packets remaining are counted as lost
```

Figure 10-Algorithm for SQFL scheme

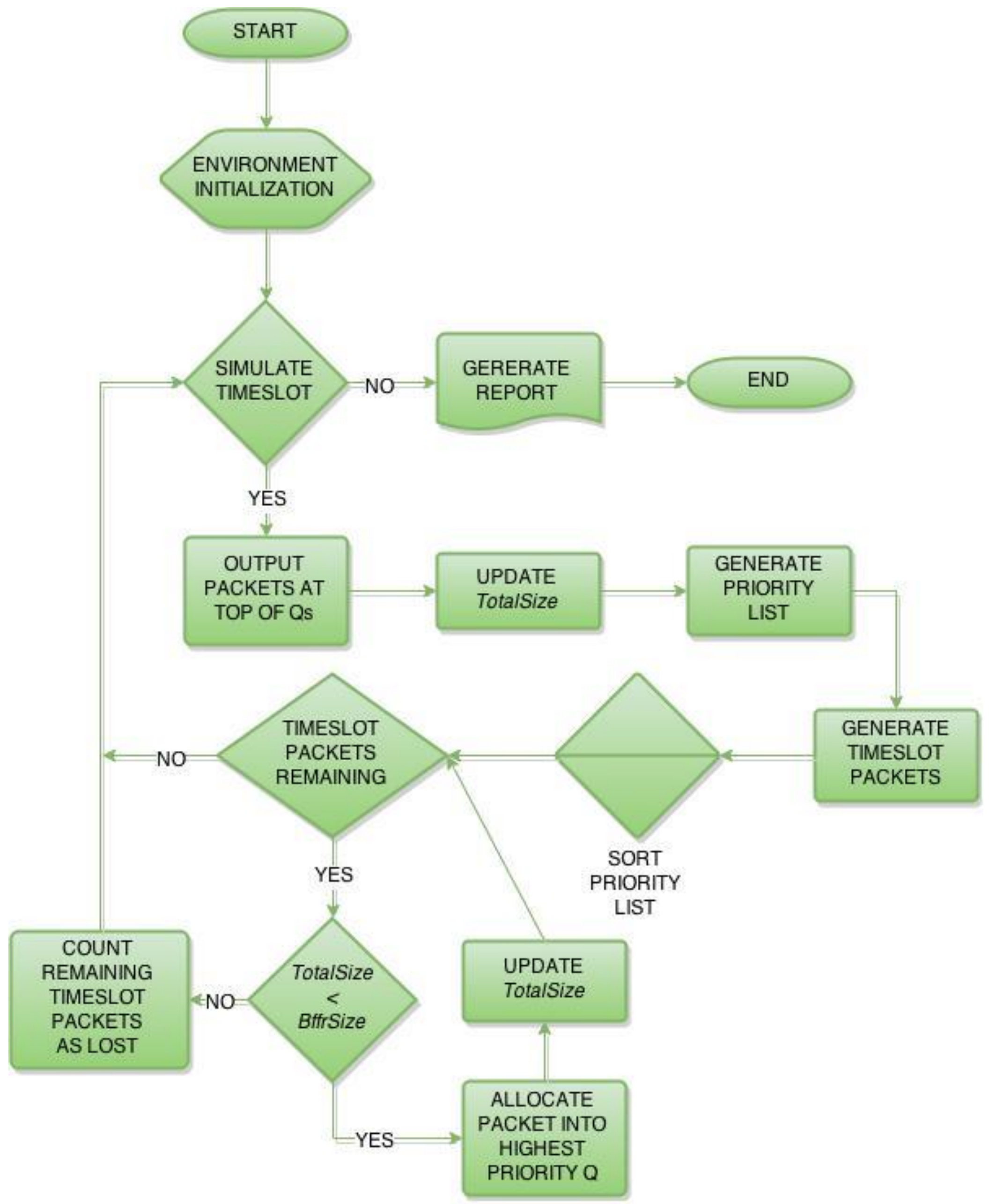

START
ENVIRONMENT INITIALIZATION
SIMULATE TIMESLOT
NO
GERERATE REPORT
END
YES
OUTPUT PACKETS AT TOP OF Qs
UPDATE *TotalSize*
GENERATE PRIORITY LIST
GENERATE TIMESLOT PACKETS
SORT PRIORITY LIST
TIMESLOT PACKETS REMAINING
NO
YES
TotalSize < BffrSize
NO
COUNT REMAINING TIMESLOT PACKETS AS LOST
YES
ALLOCATE PACKET INTO HIGHEST PRIORITY Q
UPDATE *TotalSize*

Figure 11-Flowchart for SQFL scheme

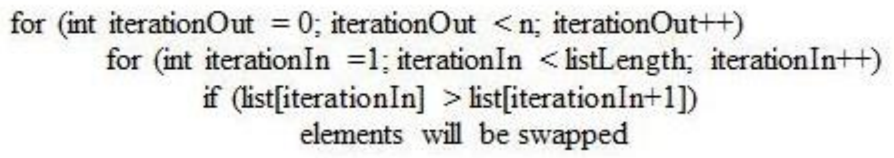
Sorting Algorithms

In this section we will discuss three sorting algorithms, each one of them having different run times. Sorting Algorithm is referring to a process that is use mostly in computer science to sort an unordered list of integers either from smallest to largest or in vice versa. We will mention

Bubble Sort, Merge Sort, and Radix Sort. Sorting Algorithms can have a very big impact on how efficient your program works whenever it is necessary.

Bubble Sort

Bubble sort is the simplest sorting algorithm in terms of implementation, which represents its biggest advantage. Bubble sort however has a very inefficient algorithm, the way this algorithm works is by iterating down an array element by element. Bubble sort compares two adjacent elements at a time and arranges them in such a way that they are ordered in ascending or descending order as required. Assuming the number of elements needed to be sorted is equal to n, then the number of sweeps required to do over the array is also equal to n. For Bubble sort the best case scenario has a time complexity of *O(n)*, whereas both the average case and worst case scenario have a time complexity of $O(n^2)$ [39] [40]. Therefore, to sort a list of size of 1024 elements, bubble sort worst case scenario will require 524288 comparisons. Figure 12 shows the algorithm for bubble sorting.

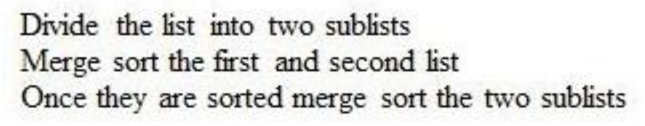


```
for (int iterationOut = 0; iterationOut < n; iterationOut++)
        for (int iterationIn =1; iterationIn < listLength; iterationIn++)
                if (list[iterationIn] > list[iterationIn+1])
                        elements will be swapped
```

Figure 12-Bubble Sort Algorithm

Merge Sort

Merge sort is a more efficient way to sort a list of integers than bubble sort achieved by utilizing recursion and a divide and conquer approach to sort a list. In the merge sort algorithm the list is recursively divided into sublists of equal size until the number of elements on each sublist equals 1. Sorting is carried out by recursively merging adjacent sublists while at the same time ordering their elements until the final result is stored in a single array completely sorted. Merge sort represents an improvement over bubble sort on both average and worst case scenarios with a time complexity of *O(n*log(n))*. The best case scenario however sees a drop in performance when compared to bubble sort with a time complexity of *O(n*log(n))*[39] [40], but this drop is not relevant because in the vast majority of cases the list to be sorted will fall under the average or worst case scenario. For a list size of 1024, merge sort will make 10230 comparisons. Figure 13 shows the algorithm for merge sorting.

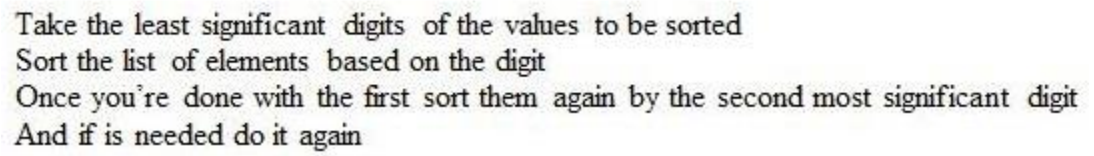


```
Divide the list into two sublists
Merge sort the first and second list
Once they are sorted merge sort the two sublists
```

Figure 13-Merge Sort Algorithm

Radix Sort

Radix sort is an algorithm that requires no comparisons, instead it uses distribution as means to achieve the sorting of an array. Radix sort employs a multiple pass distribution sorting algorithm that distributes each item in the list to a bucket according to the item's key, usually starting with the least significant part of the key. After finishing each round all the elements contained in the buckets are collected, keeping the items in the same order as they were originally stored in these buckets. The process is repeated as many times as there are radixes contained in the biggest element part of the list to be sorted. Radix sort is more efficient than both merge and bubble sort with a time complexity of *O(kn)* for its best, average, and worst case scenarios [41]. Therefore, if we intend to sort a list of 1024 elements making use of the radix sort algorithm it will take 1536 steps. Figure 14 shows the algorithm for radix sorting.



```
Take the least significant digits of the values to be sorted
Sort the list of elements based on the digit
Once you're done with the first sort them again by the second most significant digit
And if is needed do it again
```

Figure 14-Radix Sort Algorithm

Chapter Summary

As demonstrated in this chapter, by making use of a more efficient sorting algorithm, the number of iterations required in order to sort a list can be drastically reduced. If we can reduce the number of operations performed during the sorting phase of the SQFL algorithm, then we can make use of slower components, which in term reduces the cost of the entire hardware unit. Table 1draws a comparison in terms of iterations needed to sort a list composed of *n* elements between SQF and SQFL.

	SQF			SQFL		
Sort	Best	Average	Worst	Best	Average	Worst
Bubble	O(n^2)	O(n^3)	O(n^3)	O(n)	O(n^2)	O(n^2)
Merge	O(n^2 log(n))	O(n^2 log(n))	O(n^2 log(n))	O(n log(n))	O(n log(n))	O(n log(n))
Radix	O(k n^2)	O(k n^2)	O(k n^2)	O(kn)	O(kn)	O(kn)

Table 1-Sort Algorithm Performance

CHAPTER IV

SIMULATION METHODOLOGIES

In this chapter we describe the methodology followed, as well as the tools utilized during the simulation of the different buffer allocation schemes. We will start by detailing the traffic model employed in the generation of packets, we will then move on to the implementation of the buffer, next we will review the architecture and use of the cluster computer utilized for the submission of jobs during simulation, and lastly we will discuss the array of scenarios under which the different buffer allocation schemes were chosen to be evaluated.

Bursty Traffic Model

The bursty traffic is generated using a two state ON-OFF model as shown in figure 15. The ON state represents a geometrically distributed active period where packets or cells are generated in a Bernoulli fashion. The OFF state represents a geometrically distributed idle period in which no packets or cells are generated. Throughout the course of this thesis we will make use of a unit measure of time that will be referred to as a time slot. This unit represents a period of time in which no more than one packet or cell may arrive at each of the input ports.

If we use r and p to denote the duration of active and idle periods respectively, then the probability that an active state last i time slots is given by $P(i) = p \cdot (1 - p)i\text{-}1$, for $i \geq 1$, and the Average Burst Length (ABL) is given by $EB[i] = 1 / p$. In a similar way, the probability that an idle period lasts for j time slots is given by $R(j) = r \cdot (1 - r)j$, for $j \geq 0$, and the corresponding mean idle period is given by *Et[j] = (1 – r) /r*. Therefore, if p and r are given, then the offered load can be calculated using the expression *L = r / (r +p + r · p)* [42] [43].

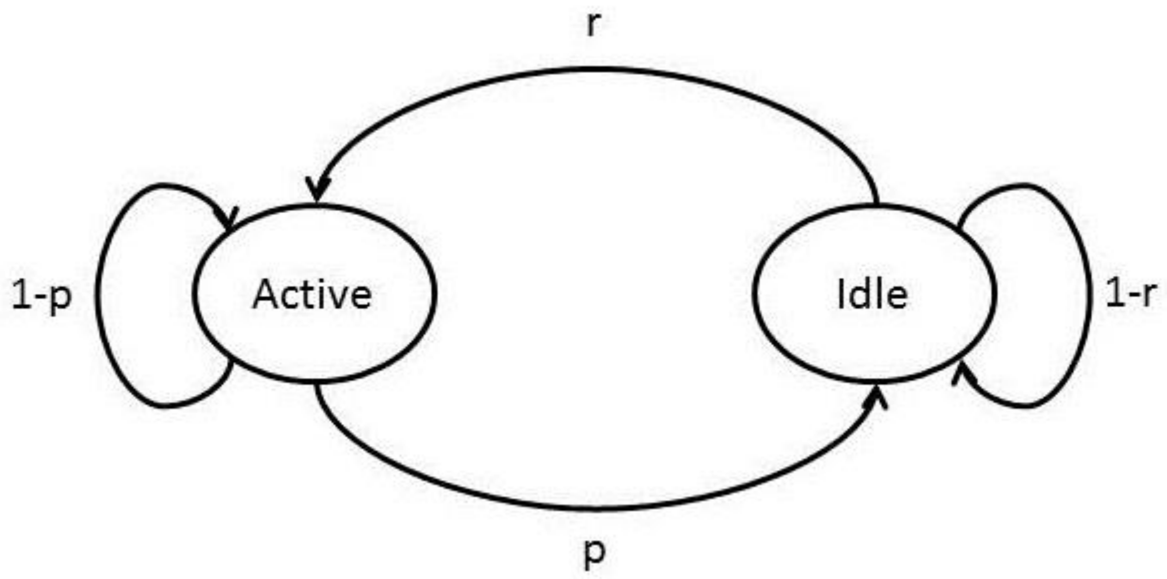


Figure 15-Bursty Traffic ON-OFF Model

Implementation

In order to implement the Bursty Traffic model a class consisting of 8 functions was implemented. The first function is initialize which receives 3 values: ABL, Load and Number of Ports. A call to this function calculates the duration idle where *p = 1 / ABL*; it calculates the duration of active where *r = (Load * p) / (1 – Load + (Load * p))*; it also determines the initial state of the model: if a randomly generated number between 0 and 1 is greater than *r* then the model starts on idle, otherwise the model starts on active.

The second function is cycle which generates a random value between 0 and 1 and determines the state of the next time slot. When the current state of the model is idle and the random number is greater than *r* the next state will be idle, otherwise the next state will be active. When the current state of the model is active and the random number is greater than *p* the next state will be active, otherwise the next state will be idle. If the state of the model changes from active to idle and the random number is greater than *r* then that idle period will last at least one time slot, otherwise the idle period will last zero time slots. Every time the Bursty Traffic model state changes from idle to active the destination of the packet generated will change. As long as the state of the model remains active the destination of the packets generated will not change. A series of active periods in a row generates a burst or train where all packets share the same destination. In the instance where the Bursty Traffic model state changes from active to idle and immediately back to active, i.e. when the duration of the idle period is zero time slots, we have what is known as back to back trains; this looks like a single group of active periods where at one point the destination of the packet generated changes, thus implying the presence of an idle period lasting zero time slots.

Six more functions are implemented that provide statistics about the traffic generated by the Bursty Traffic model.

"retout" returns the destination of the past packet generated, where 0 signifies idle and any other number signifies the port in the switch to which the packet is destined.

"retactive" returns the number of active time slots.

"retidle" returns the number of idle time slots.

"retrains" returns the number of trains.

"retL" returns the calculated Load where *L = active / (active + idle)*.

"retABL" returns the calculated ABL where *ABL = active / # of trains*.

Buffer

A buffer is physical memory where data can be temporarily stored while it waits to be sent out to its destination. Buffers are required due to instances in which more than one packet arriving to the switch share their destination port, only one of these packets will be able to be outputted, while the other will remain in the buffer waiting for the next time slot [11] [44].

Implementation

The buffer was implemented as a collection of queues, one for each output port. The data structure used to implement each of the necessary queues was a linked list. A linked list is a dynamic data structure, which means that it does not have a fixed number of elements; it can grow or shrink as needed [40]. Four functions were implemented when defining our queue:

"retsize" returns the size of the queue, which is used to monitor the total occupied space in the available buffer.

"retavgdelay" returns the calculated average delay; the average time packets spend in the queue, from the moment they arrive to the moment they depart.

"add" adds a new element to the queue.

"update" deletes the oldest element of the queue, updates the size variable, as well as calculates the new average delay.

Cluster Computer

In order to complete this thesis a cluster computer was utilized to run the simulations on. Because of its inherit ability to run multiple tasks at once, the use of a cluster was crucial in the process of simulating and generating data. Without it the number of computations required to collect the data necessary to write this thesis would have taken months to complete, instead of weeks. Ahead the architecture of the utilized cluster computer, as well as the method for job submission are discussed.

Architecture

In the cluster computer at The University of Texas-Pan American the main system is composed of 68 compute nodes running Red Hat Linux. Of the 68 compute nodes 8 are dedicated for use as a single “virtual SMP node”, and the remaining 60 are regular compute nodes. In total there are 816 cores available for computation, each node consisting of 12 dual-core Intel Xeon processors as well as 4 GB of memory. Four more nodes are used for internal purposes like the cluster controller, the primary and backup login servers, and the file server. The login node is as a job scheduler, submitting jobs for processing to the compute nodes [45]. Figure 16 shows the diagram of the cluster computer architecture.

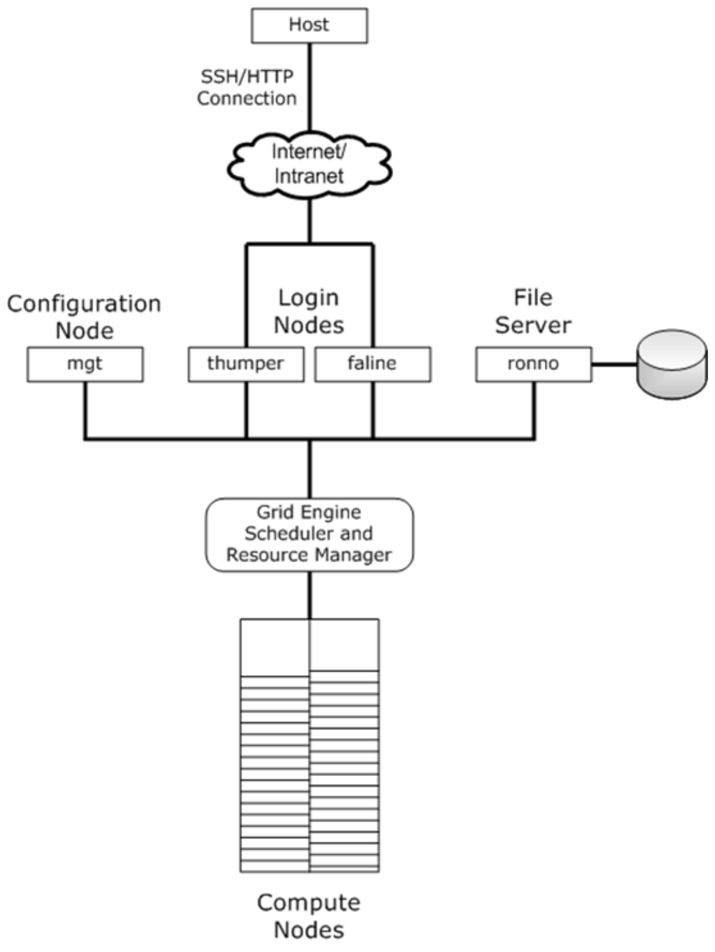
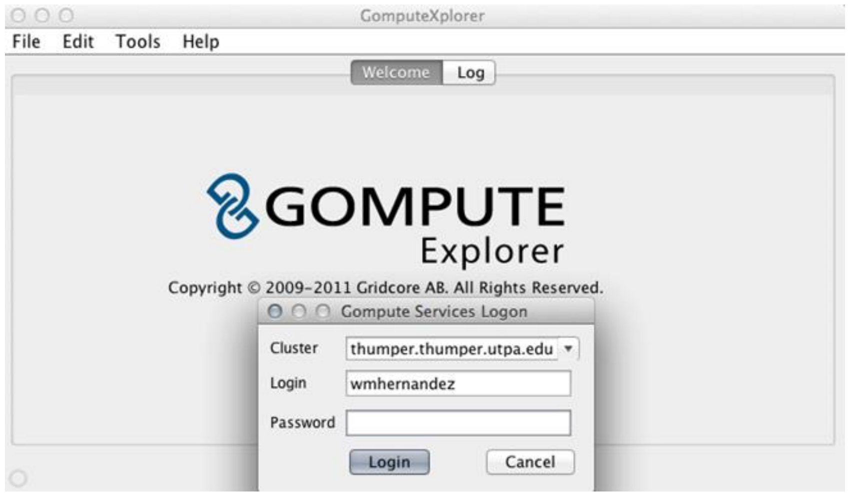


Figure 16-Cluster Computer Diagram

Job Submission

In order to submit jobs to the cluster computer the following steps are required:

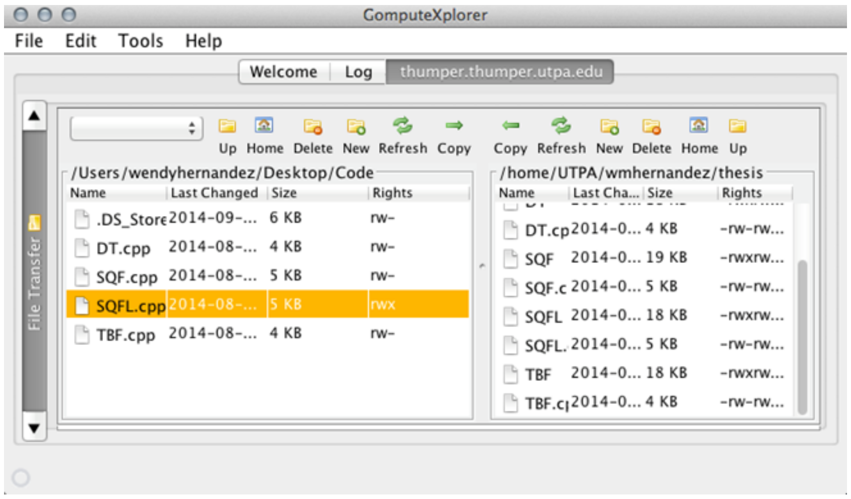
- Download and install the software GOMPUTE Explorer. This will serve as a guided user interface (GUI) that will allow for the upload and download of code and results to and from the cluster.
- Upon launching the GOMPUTE Explorer a prompt for the cluster's account username and password as shown in figure 17.
- For every different scenario simulated a file has to be uploaded and submitted to the cluster to be executed utilizing the GUI as shown in figure 18.
- Once a code is loaded into the cluster access to the cluster's bash shell is required, such as Putty.
- The first command inputted into Putty is "g++ -o NameOfFile NameOfFile.cpp" which compiles the code and generates a binary executable file.
- The second command inputted into Putty is "qsub –b y ~/nameOfFolder/NameOfFile" which takes the binary executable file and feeds it to the job scheduler to be submitted to a computing core.
- Once a program finishes execution it will generate an output file in plain text format containing the packet loss ratio and average delay from that particular run.
- The generated results file is then downloaded from the cluster computer by making use of the GUI as shown in figure 19.



Copyright © 2009-2011 Gridcore AB. All Rights Reserved.

Cluster	thumper.thumper.utpa.edu
Login	wmhernandez
Password	

Figure 17-GOMPUTE Login


GomputeXplorer
File Edit Tools Help
Welcome Log thumper.thumper.utpa.edu
Up Home Delete New Refresh Copy
Copy Refresh New Delete Home Up
File Transfer

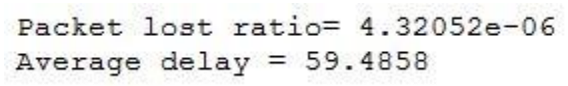
/Users/wendyhernandez/Desktop/Code

Name	Last Changed	Size	Rights
.DS_Store	2014-09-...	6 KB	rw-
DT.cpp	2014-08-...	4 KB	rw-
SQF.cpp	2014-08-...	5 KB	rw-
SQFL.cpp	2014-08-...	5 KB	rwx
TBF.cpp	2014-08-...	4 KB	rw-

/home/UTPA/wmhernandez/thesis

Name	Last Cha...	Size	Rights
DT.cp	2014-0...	4 KB	-rw-rw...
SQF	2014-0...	19 KB	-rwxrw...
SQF.c	2014-0...	5 KB	-rw-rw...
SQFL	2014-0...	18 KB	-rwxrw...
SQFL.	2014-0...	5 KB	-rw-rw...
TBF	2014-0...	18 KB	-rwxrw...
TBF.c	2014-0...	4 KB	-rw-rw...

Figure 18-GOMPUTE File Transfer



```
Packet lost ratio= 4.32052e-06
Average delay = 59.4858
```

Figure 19-Output File Sample

Simulation Scenarios

The simulated sharing memory allocation schemes were subjected to a wide range of scenarios including three different sizes of switches: 16, 32, and 64 ports; 3 different buffer sizes: 1024, 2048, and 4096 packets. For DT the chosen alphas were 1, 4, 8, and 32; while for TF the utilized threshold was relative to the total memory: 1/16, 1/8, and 1/4.

As for the traffic generated, the memory allocation schemes where subjected to 3 different average burst lengths: 8, 32, and 64 packets per burst, as well as loads ranging from 10% to 100% in step increments of 10%.

CHAPTER V

OPTIMAL CONFIGURATIONS FOR CONVENTIONAL SCHEMES

In this chapter we will present the effects of the four main variables we used to simulate the Dynamic Threshold and Threshold-Based Filtering schemes, which are: Average Burst Length and load of the traffic fed into the switch, and the total size of the buffer as well as number of ports available on the switch. We will determine the optimal configurations in terms of packet loss ratio for both conventional schemes under each of the different tested scenarios. The last section of the chapter provides a summary of the results presented on this chapter.

Dynamic Threshold (DT)

Average Burst Length (ABL)

Increasing the ABL extends the range of high end loads under which the high load optimal configuration remains the most effective at reducing the packet loss. As an example we look at the scenario in which the number of ports on the switch equals 48, the total available buffer is 1024 cells. We can observe that when subjected to traffic with an ABL of 8 the optimal configuration on the higher loads requires an alpha equal to 1; this is the optimal configuration under loads of 100% down to 80%. When we quadruple the ABL of the traffic to 32 we can observe that the alpha required for an optimal configuration is again equal to 1, however this configuration now yields the best results in terms of packets loss ratio under loads of 100% down to 60%. If we increase the ABL of the traffic further, this time to 64, then we are able to observe how by only doubling the ABL the range of effectiveness for which the configuration in which alpha is equal to 1 is not different from the scenario in which the ABL is 32. The difference however can be noticed on the second range of loads. When the ABL is 32 the configuration with an alpha equal to 4 is optimal under loads of 50%, whereas when the ABL is 64 this same configuration is optimal for loads of 50% and 40%. The results are shown in figures 20, 21 and 22.

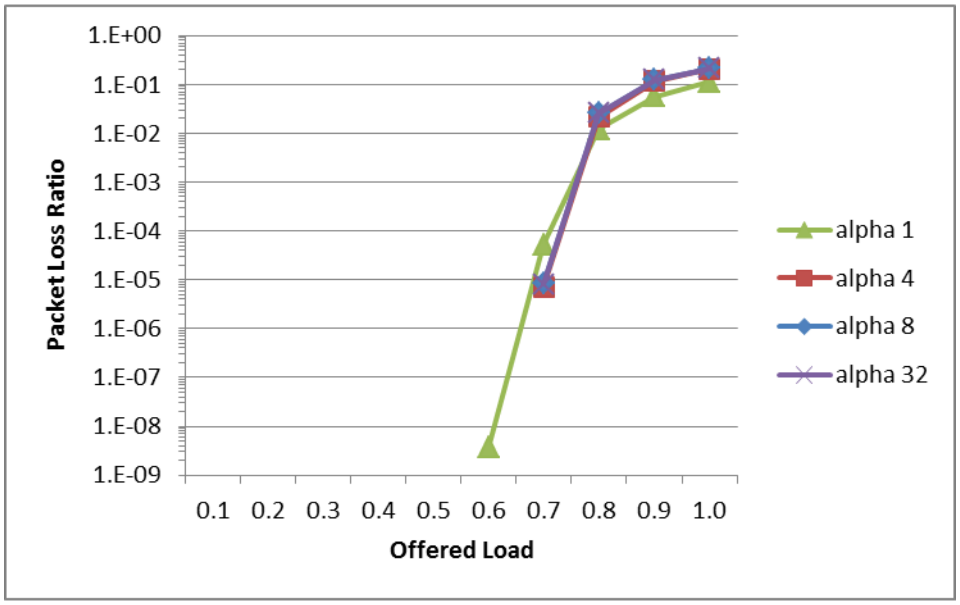


Figure 20-DT; Ports=48; Buffer=1024; ABL=8

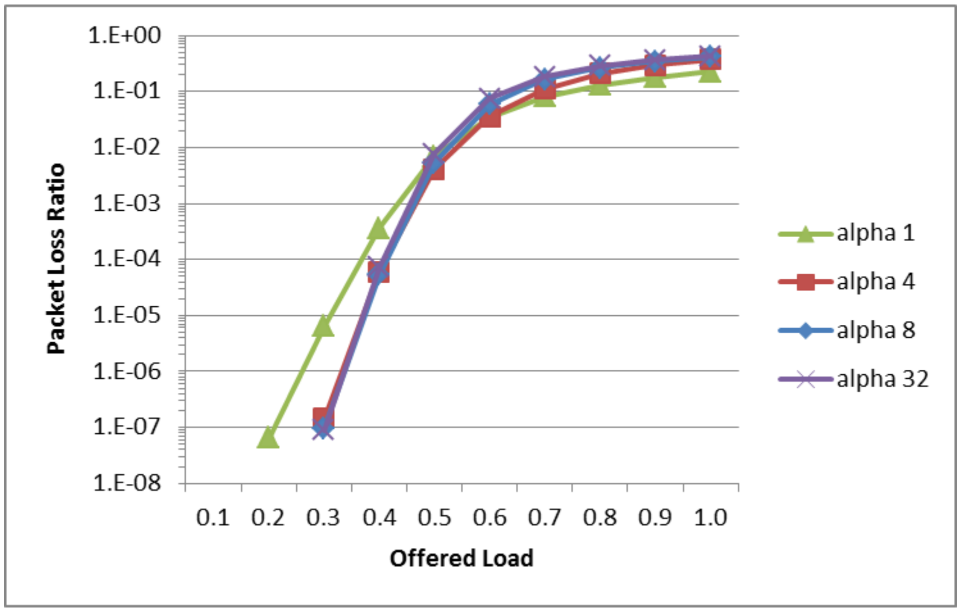


Figure 21-DT; Ports=48; Buffer=1024; ABL=32

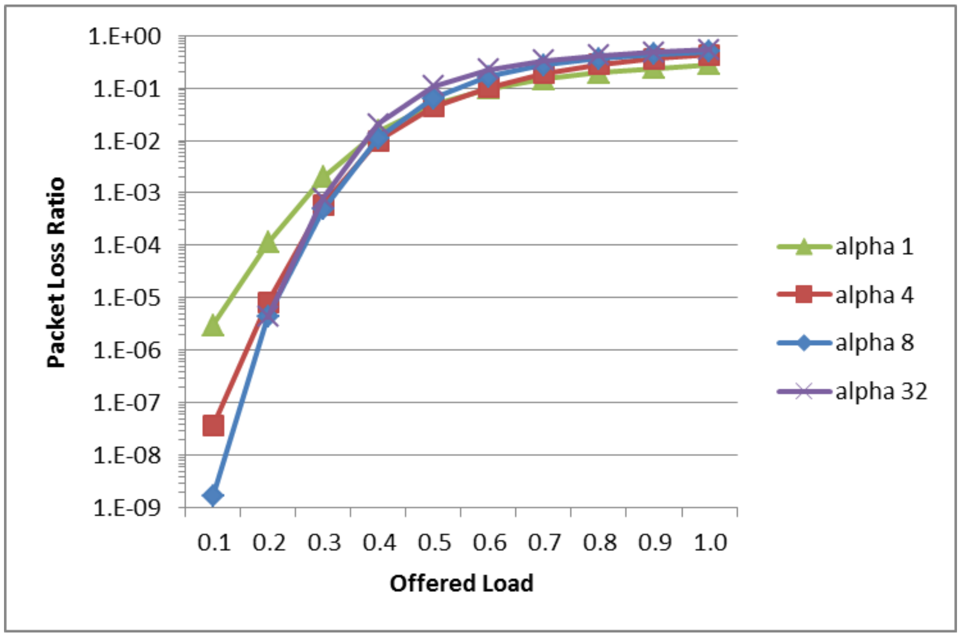


Figure 22-DT; Ports=48; Buffer=1024; ABL=64

Buffer Versus Ports

As the ratio between the buffer size and the number of ports increases, the alpha required to obtain the optimal configuration becomes larger. If for example we observe all the simulated buffer/ports ratio under a load of 100 percent, we find that with a ratio of 21.33 and 42.67 the alpha required in order to achieve optimal configuration is equal to 1; this particular configuration shows an improvement of 48.09% over the remaining tested scenarios. With a ratio of 85.33 the alpha required in order to achieve optimal configuration is equal to 4; this particular configuration shows an improvement of 47.24% over the remaining tested scenarios. With a ratio of 128 and 170.67 the alpha required in order to achieve optimal configuration is equal to 8; this particular configuration shows an improvement of 43.60% over the remaining tested scenarios. With a ratio of 256 and 512 the alpha required in order to achieve optimal configuration is equal to 32; this particular configuration shows an improvement of 17.40% over the remaining tested scenarios. The results are shown in figures 23 to 27.

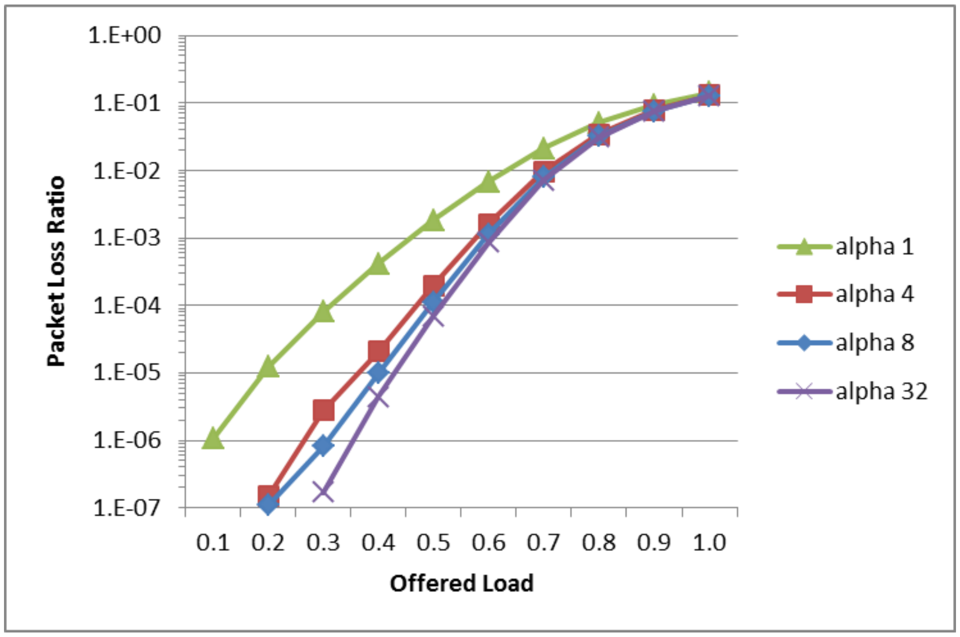


Figure 23-DT; Ports=8; Buffer=1024; ABL=64

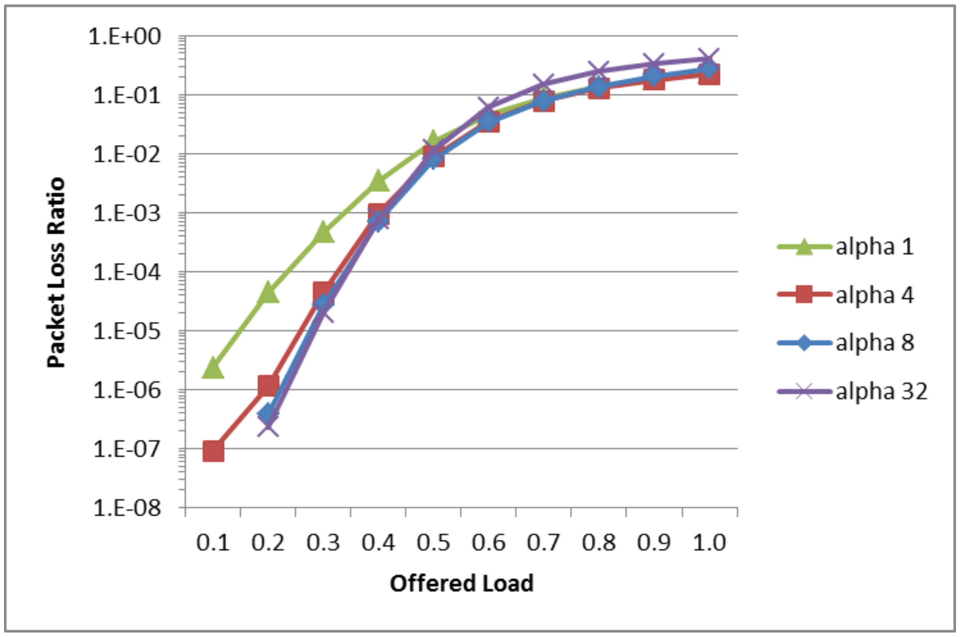


Figure 24-DT; Ports=24; Buffer=1024; ABL=64

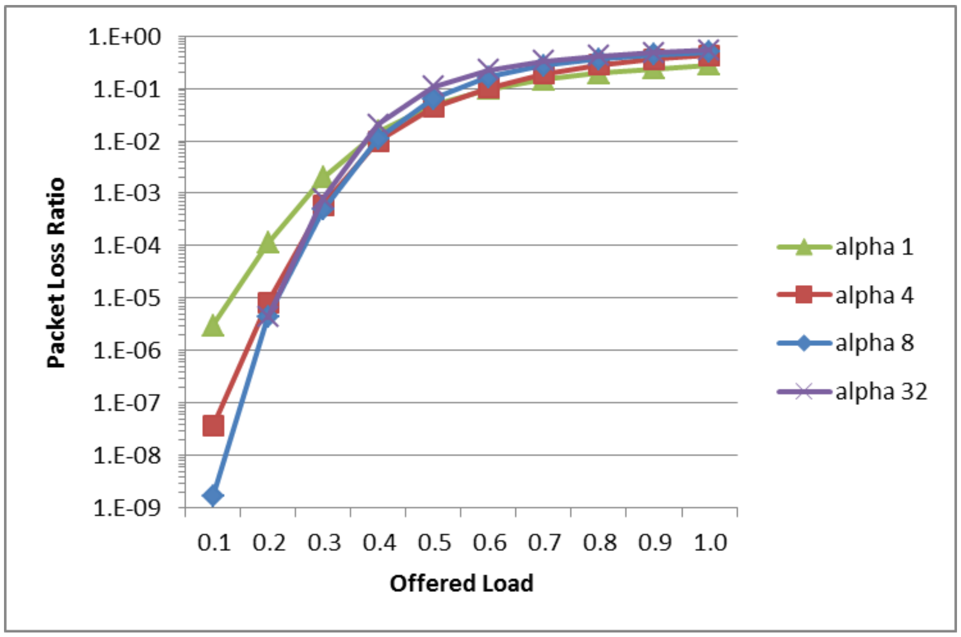


Figure 25-DT; Ports=48; Buffer=1024; ABL=64

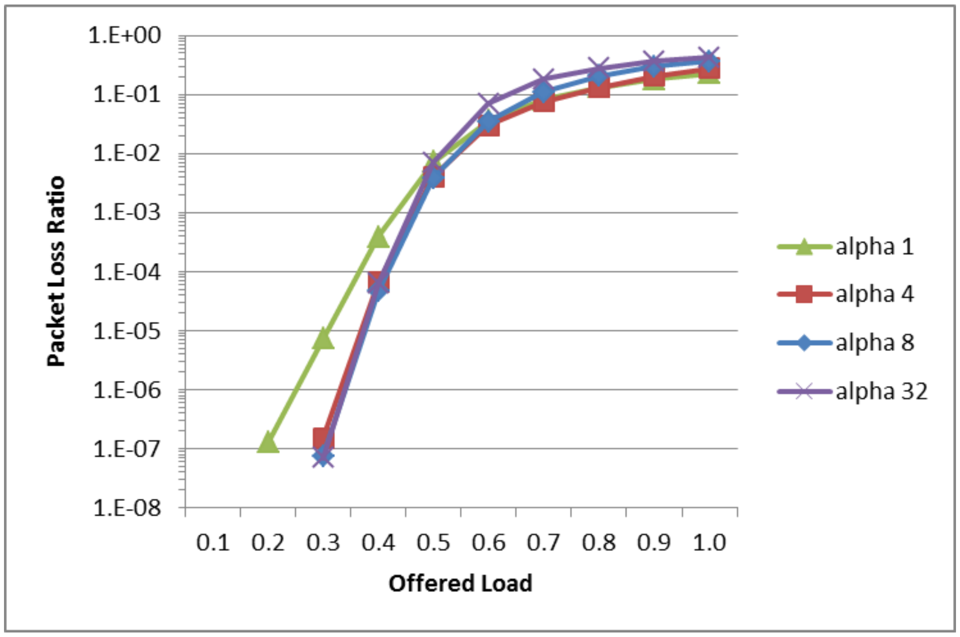


Figure 26-DT; Ports=48; Buffer=2048; ABL=64

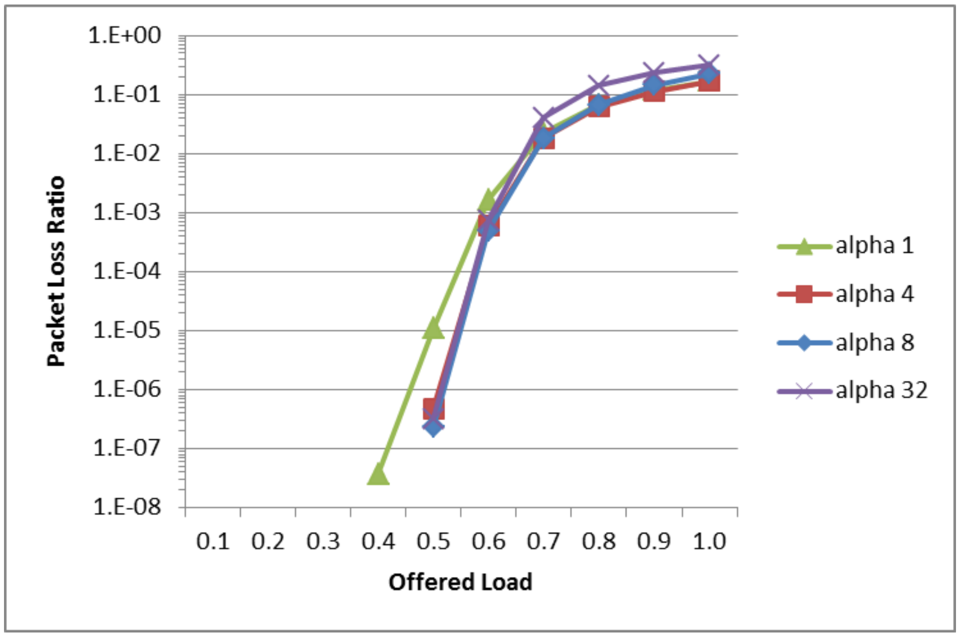


Figure 27-DT; Ports=48; Buffer=4096; ABL=64

Load

As the load decreases the alpha required in order to achieve the optimal configuration increases. If we look at the scenario where the number of ports equals 48, the ABL is set to 64, and the available buffer size is 1024, then we can appreciate the effect. in this particular we can see that for loads ranging from 100% down to 60% the alpha required for an optimal configuration is 1, being up to 57.67% better than the other configurations; we can appreciate how as the load to which the switch is subjected gets lower the difference between alpha 1 and alpha 4 becomes smaller, until eventually at a load of 50% alpha 4 becomes the optimal configuration. Alpha 4 is optimal only on loads of 40% and 50%, being up to 59.31% better than the other configurations. Under loads of 30% the optimal configuration is with an alpha of 8, being up to 75.92% better than the other configurations. And under loads of 10% and 20% the best configuration was with an alpha of 32, being up to 100% better than the other configurations. The results are shown in figures 28, 29 and 30.

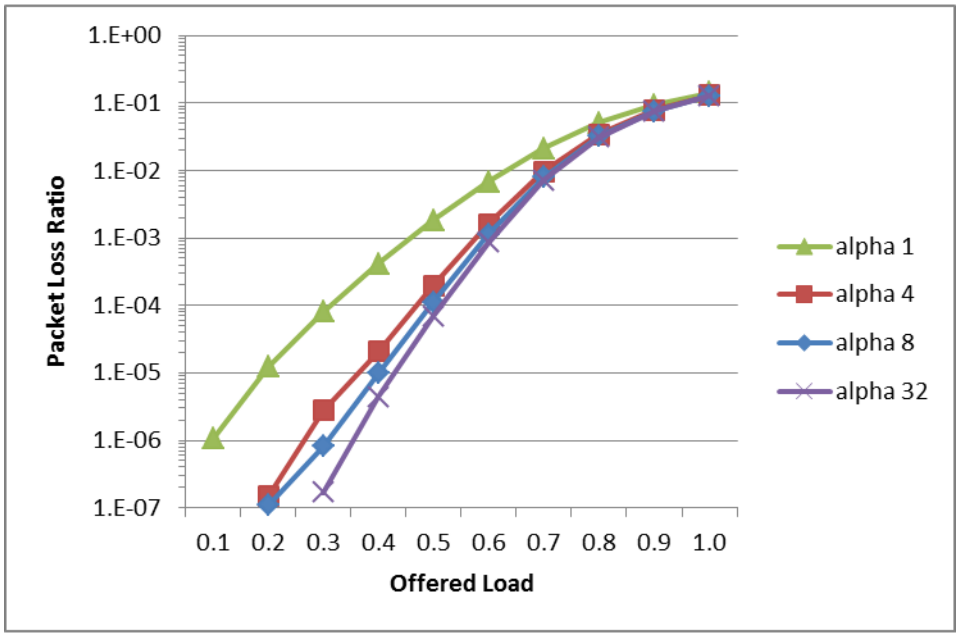


Figure 28-DT; Ports=8; Buffer=1024; ABL=64

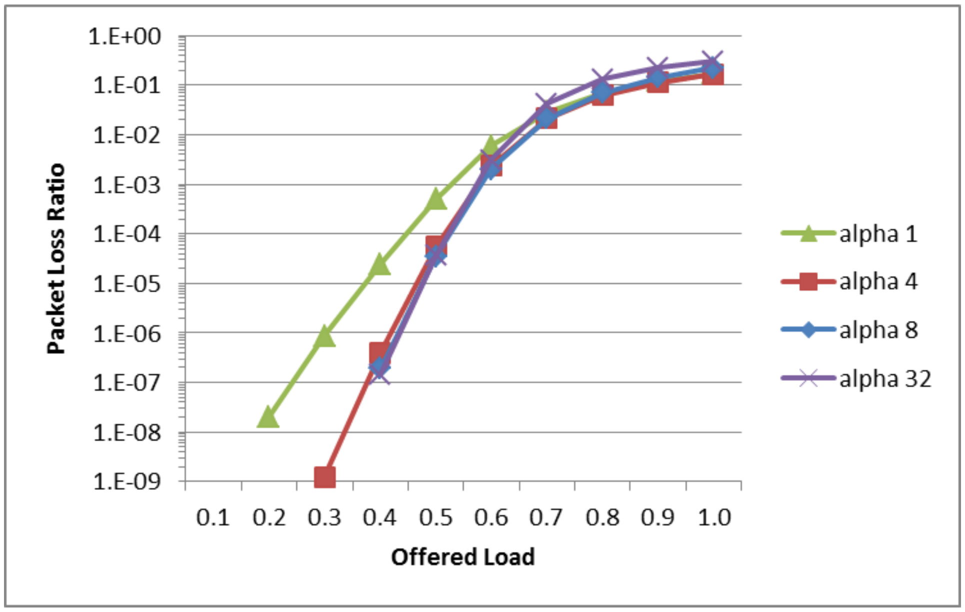


Figure 29-DT; Ports=24; Buffer=1024; ABL=32

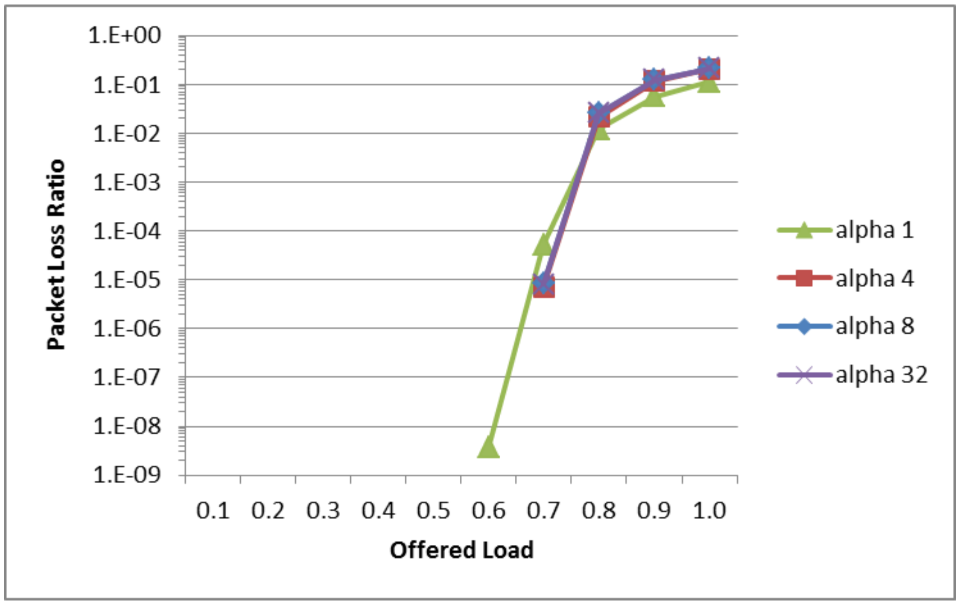


Figure 30-DT; Ports=48; Buffer=1024; ABL=8

Threshold-Based Filtering (TF)

Average Burst Length (ABL)

After simulating the different configurations selected for the TF scheme it was found that under all tested scenarios a threshold of 1/16 of the total available memory provided the optimal configuration, followed by a threshold of 1/8, and a threshold of 1/4 in that order. If for example we are to take as an example the scenario in which the number of ports equals 8 and the total available memory is 1024, then we find that under a traffic with an ABL of 8, 32 and 64, regardless of the load to which the switch is subjected, the optimal configuration requires a threshold of 1/16 of the total available memory. However it may be noticed how as the ABL of the traffic subjected increases, the extent to which this configuration outperforms the remaining simulated configurations diminishes. We found that under an ABL of 8 the optimal configuration outperforms the others by up to 8.71%; under an ABL of 32 the optimal configuration outperforms the others by up to 4.06%; under an ABL of 64 the optimal configuration outperforms the others by up to 2.46%. The results are shown in figures 31, 32 and 33.

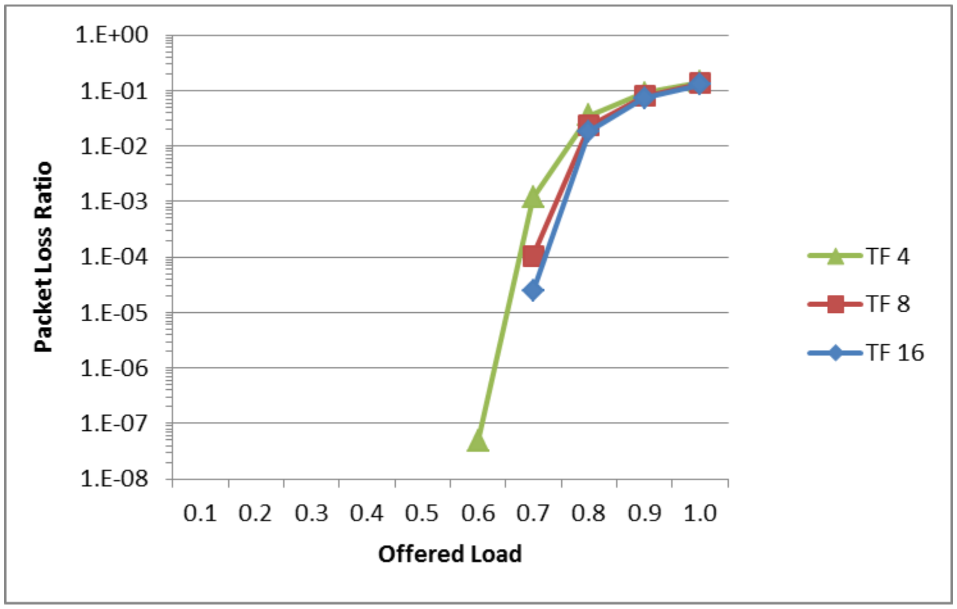


Figure 31-TF; Ports=48; Buffer=1024; ABL=8

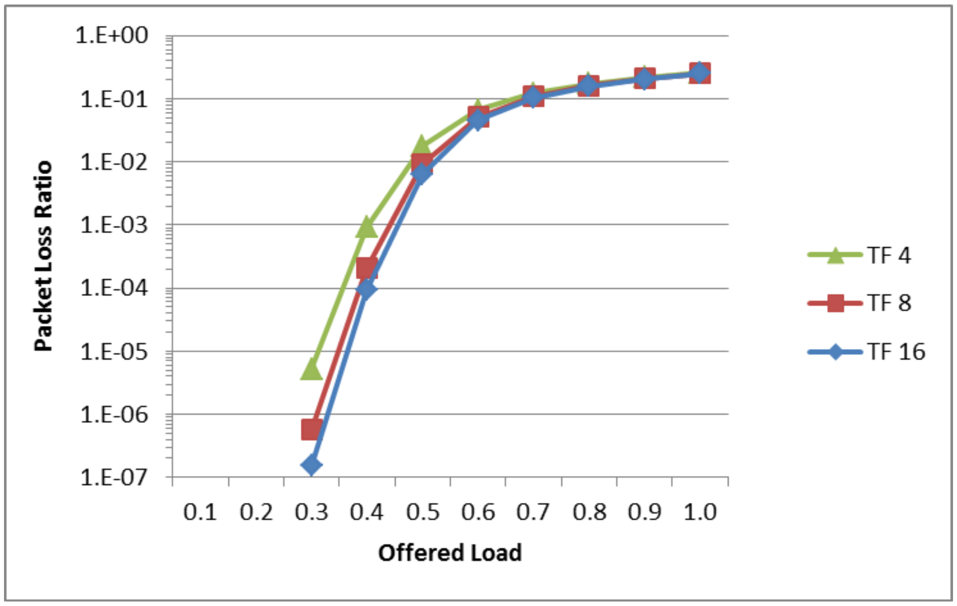


Figure 32- TF; Ports=48; Buffer=1024; ABL=32

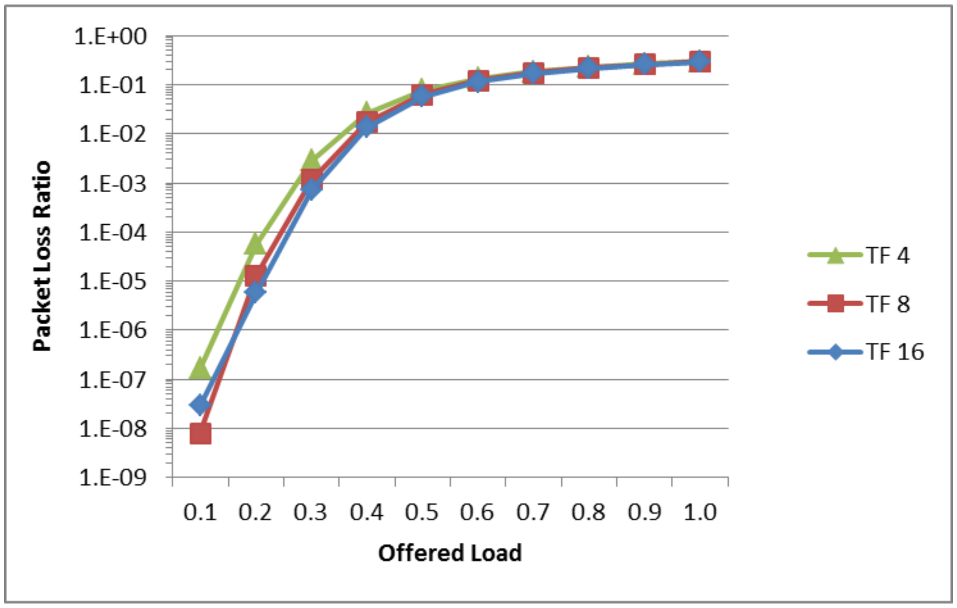


Figure 33- TF; Ports=48; Buffer=1024; ABL=64

Buffer Versus Ports

After simulating the different configurations selected for the TF scheme it was found that under all tested scenarios a threshold of 1/16 of the total available memory provided the optimal configuration, followed by a threshold of 1/8, and a threshold of 1/4 in that order. If for example we look at all the scenarios in which the traffic load and ABL are equal to 100% and 64 respectively, then we find that regardless of the ratio of buffer versus number of ports, the optimal configuration requires a threshold of 1/16 of the total available memory. However it may be noticed how as the ratio between buffer and ports increases, the extent to which this configuration outperforms the remaining simulated configurations increases. We found that with a ratio of 21.33 the optimal configuration outperforms the others by up to 2.46%; with a ratio of 85.33 the optimal configuration outperforms the others by up to 6.42%; with a ratio of 170.67 the optimal configuration outperforms the others by up to 8.74%; with a ratio of 512 the optimal configuration outperforms the others by up to 13.04%. The results are shown in figures 34 to 38.

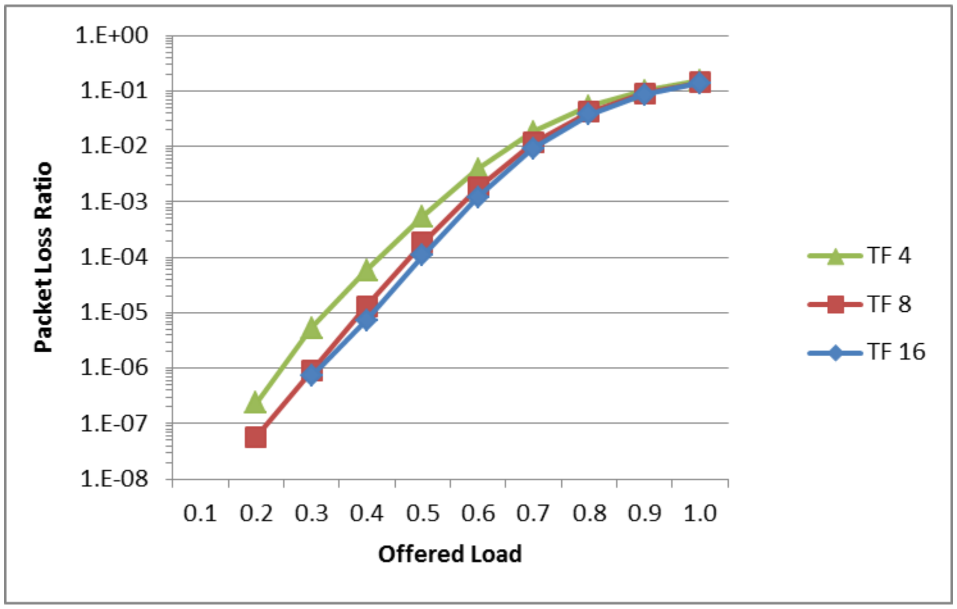


Figure 34- TF; Ports=8; Buffer=1024; ABL=64

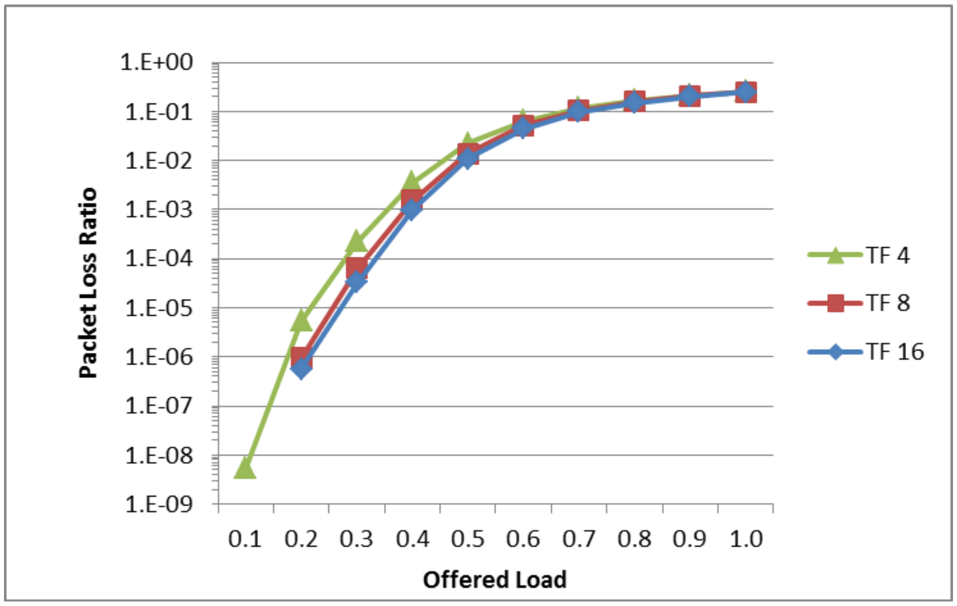


Figure 35- TF; Ports=24; Buffer=1024; ABL=64

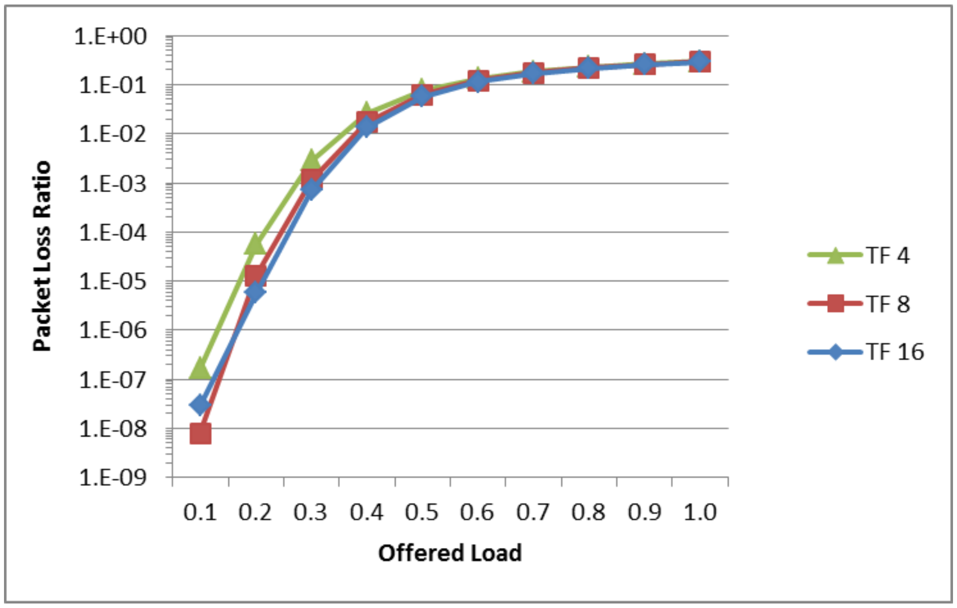


Figure 36- TF; Ports=48; Buffer=1024; ABL=64

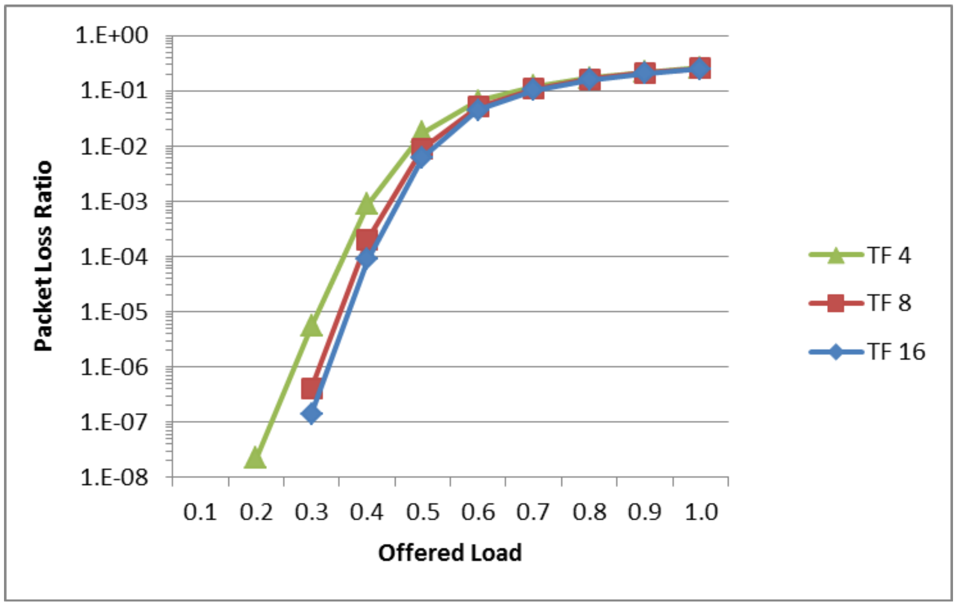


Figure 37- TF; Ports=48; Buffer=2048; ABL=64

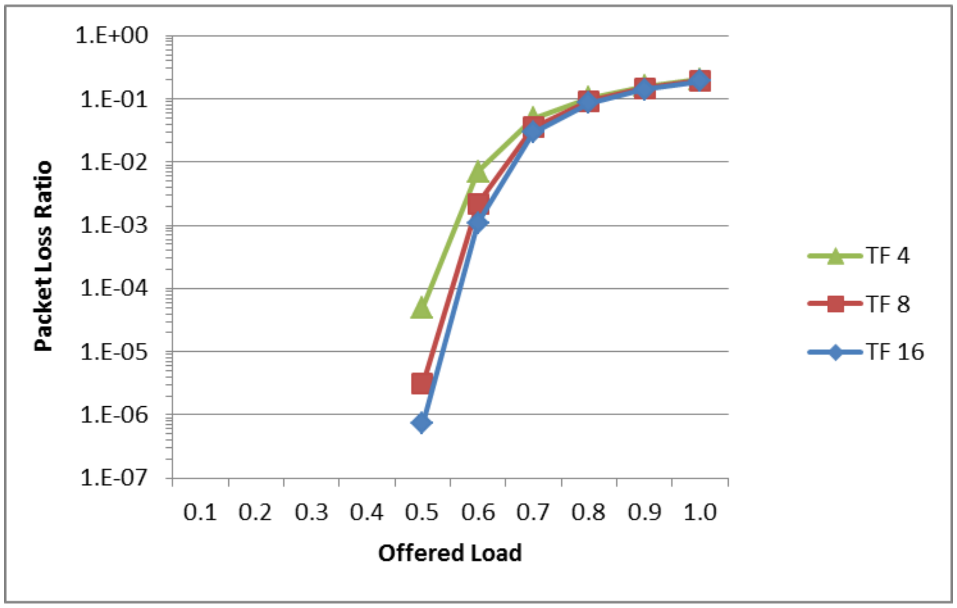


Figure 38- TF; Ports=48; Buffer=4096; ABL=64

Load

After simulating the different configurations selected for the TF scheme it was found that under all tested scenarios a threshold of 1/16 of the total available memory provided the optimal configuration, followed by a threshold of 1/8, and a threshold of 1/4 in that order. If for example we look at the scenario in which the number of ports and total buffer size available in a switch equal 48 and 1024 respectively, and the ABL is equal to 64, then we find that regardless of the load to which the switch is subjected, the optimal configuration requires a threshold of 1/16 of the total available memory. However it may be noticed that as the load decreases, the extent to which this configuration outperforms the remaining simulated configurations increases. It was found that under a load of 100% this configuration outperformed the other simulated configurations by up to 2.46%; under a load of 80% this configuration outperformed the other simulated configurations by up to 4.94%; under a load of 60% this configuration outperformed the other simulated configurations by up to 13.13%; under a load of 40% this configuration outperformed the other simulated configurations by up to 48.47%; under a load of 20% this configuration outperformed the other simulated configurations by up to 89.57%. The results are shown in figure 39, 40 and 41.

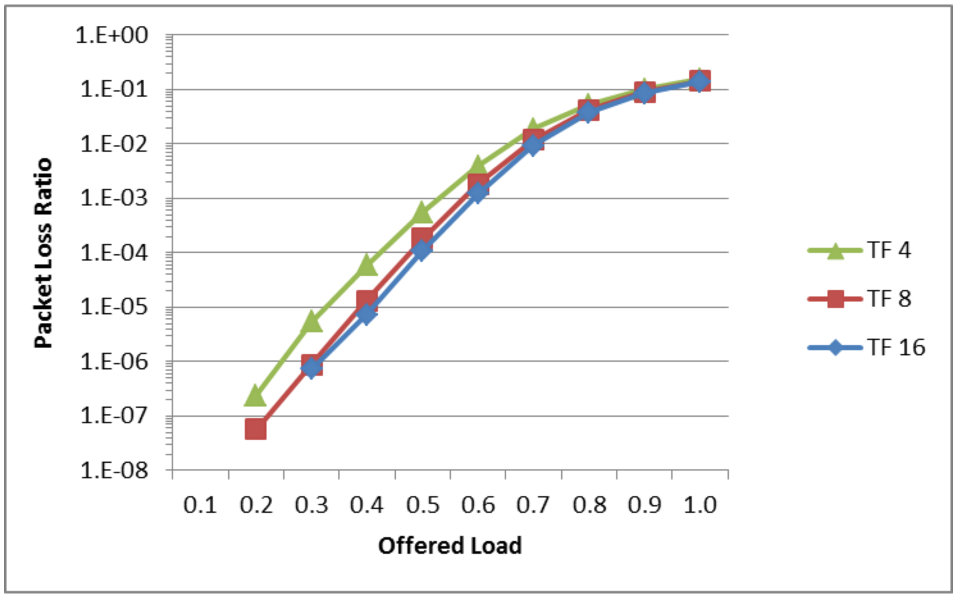


Figure 39-TF; Ports=8; Buffer=1024; ABL=64

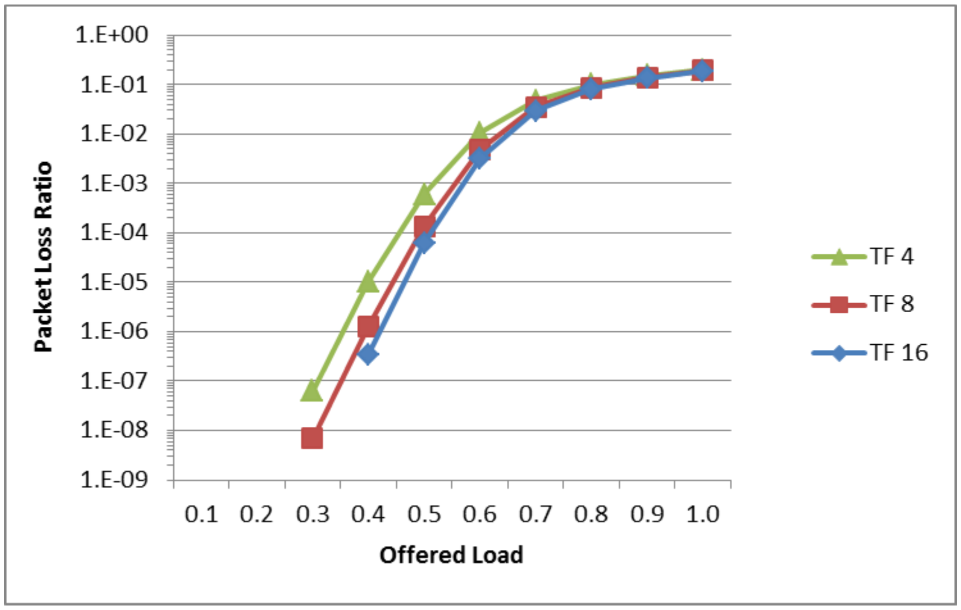


Figure 40-TF; Ports=24; Buffer=1024; ABL=32

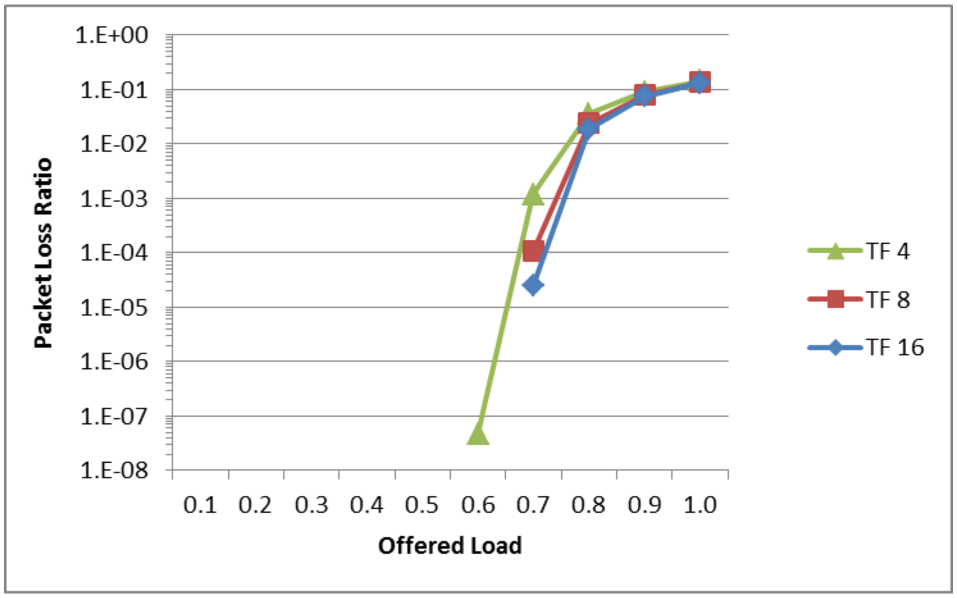


Figure 41-TF; Ports=48; Buffer=1024; ABL=8

Summary of Results

Out of the simulated shared memory schemes, TF is the easiest to configure, because regardless of the average burst length, load, number of ports or size of available buffer, the manner in which it performs is always consistent. Under all tested scenarios TF required a threshold of 1/16 of the total available buffer in order to achieve the lowest packet loss ratio. The threshold required for an optimal performance for TF under the different tested scenarios is shown in table 2.

DT on the other hand requires a more granular fine tuning, having a required Alpha ranging from 1 to 32 depending on the switch configuration and traffic load to which they are subjected. The alpha required for an optimal performance for DT under the different tested scenarios is shown in table 3.

			L									
Ports	ABL	Bffr	10	20	30	40	50	60	70	80	90	100
8	8	1024	32	32	32	32	32	32	32	32	32	8
8	8	2048	32	32	32	32	32	32	32	32	32	32
8	8	4096	32	32	32	32	32	32	32	32	32	32
8	32	1024	32	32	32	32	32	32	32	32	32	8
8	32	2048	32	32	32	32	32	32	32	32	32	32
8	32	4096	32	32	32	32	32	32	32	32	32	32
8	64	1024	32	32	32	32	32	32	32	32	32	8
8	64	2048	32	32	32	32	32	32	32	32	32	32
8	64	4096	32	32	32	32	32	32	32	32	32	32
24	8	1024	8	8	8	8	8	8	8	8	4	4
24	8	2048	8	8	8	8	8	8	8	8	8	4
24	8	4096	8	8	8	8	8	8	8	8	8	8
24	32	1024	32	32	32	32	8	8	8	4	4	4
24	32	2048	32	32	32	32	32	32	8	8	8	4
24	32	4096	32	32	32	32	32	32	32	32	8	8
24	64	1024	32	32	32	8	8	8	4	4	4	4
24	64	2048	32	32	32	32	32	8	8	8	8	4
24	64	4096	32	32	32	32	32	32	32	8	8	8
48	8	1024	4	4	4	4	4	4	4	1	1	1
48	8	2048	8	8	8	8	8	8	8	8	1	1
48	8	4096	8	8	8	8	8	8	8	8	8	1
48	32	1024	32	32	32	8	4	1	1	1	1	1
48	32	2048	8	8	8	8	8	8	4	1	1	1
48	32	4096	8	8	8	8	8	8	8	4	4	4
48	64	1024	32	32	8	4	4	1	1	1	1	1
48	64	2048	32	32	32	8	8	4	4	1	1	1
48	64	4096	8	8	8	8	8	8	8	4	4	4

Table 2-Optimal Configurations for DT

			L									
Ports	ABL	Bffr	10	20	30	40	50	60	70	80	90	100
8	8	1024	16	16	16	16	16	16	16	16	16	16
8	8	2048	16	16	16	16	16	16	16	16	16	16
8	8	4096	16	16	16	16	16	16	16	16	16	16
8	32	1024	16	16	16	16	16	16	16	16	16	16
8	32	2048	16	16	16	16	16	16	16	16	16	16
8	32	4096	16	16	16	16	16	16	16	16	16	16
8	64	1024	16	16	16	16	16	16	16	16	16	16
8	64	2048	16	16	16	16	16	16	16	16	16	16
8	64	4096	16	16	16	16	16	16	16	16	16	16
24	8	1024	16	16	16	16	16	16	16	16	16	16
24	8	2048	16	16	16	16	16	16	16	16	16	16
24	8	4096	16	16	16	16	16	16	16	16	16	16
24	32	1024	16	16	16	16	16	16	16	16	16	16
24	32	2048	16	16	16	16	16	16	16	16	16	16
24	32	4096	16	16	16	16	16	16	16	16	16	16
24	64	1024	16	16	16	16	16	16	16	16	16	16
24	64	2048	16	16	16	16	16	16	16	16	16	16
24	64	4096	16	16	16	16	16	16	16	16	16	16
48	8	1024	16	16	16	16	16	16	16	16	16	16
48	8	2048	16	16	16	16	16	16	16	16	16	16
48	8	4096	16	16	16	16	16	16	16	16	16	16
48	32	1024	16	16	16	16	16	16	16	16	16	16
48	32	2048	16	16	16	16	16	16	16	16	16	16
48	32	4096	16	16	16	16	16	16	16	16	16	16
48	64	1024	16	16	16	16	16	16	16	16	16	16
48	64	2048	16	16	16	16	16	16	16	16	16	16
48	64	4096	16	16	16	16	16	16	16	16	16	16

Table 3-Optimal Configurations for TF

CHAPTER VI

COMPARATIVE PERFORMANCE EVALUATION

In this chapter we test and evaluate the proposed scheme Shortest Queue First Lite (SQFL) under the same simulated scenarios as the conventional schemes tested in the previous chapter. We will compare their optimal configurations against SQFL and determine the extent to which SQFL outperforms DT and TF in terms of packet loss ratio.

DT Versus TF

Average Burst Length (ABL)

We can appreciate how when comparing DT versus TF that under all tested average burst lengths DT outdid TF. The degree to which DT enhanced upon the performance of TF fluctuates however. For instance, we found that in the case when the tested switch was composed of 48 ports, with a total available buffer of 1024 is subjected to a load of 100%. When the ABL of the traffic equals 8 DT has an improvement of 13.97 % over TF. When we increase the ABL from 8 to 32 TF performed better leaving a smaller gap between DT and TF, reducing the gap in performance a difference of 8.61%. If the ABL is increased furthermore, this time to 64, then we find that the gap between DT and TF becomes even smaller, resulting in a performance difference of 5.08%. The results are shown in figures 42, 43 and 44.

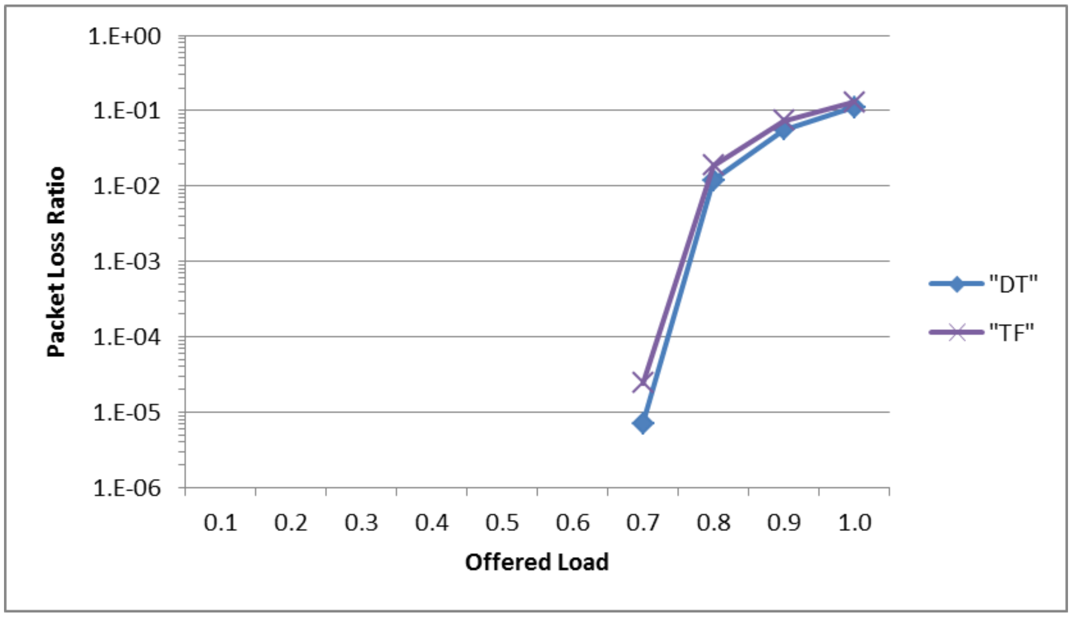


Figure 42-DT vs TF; Ports=48; Buffer=1024; ABL=8

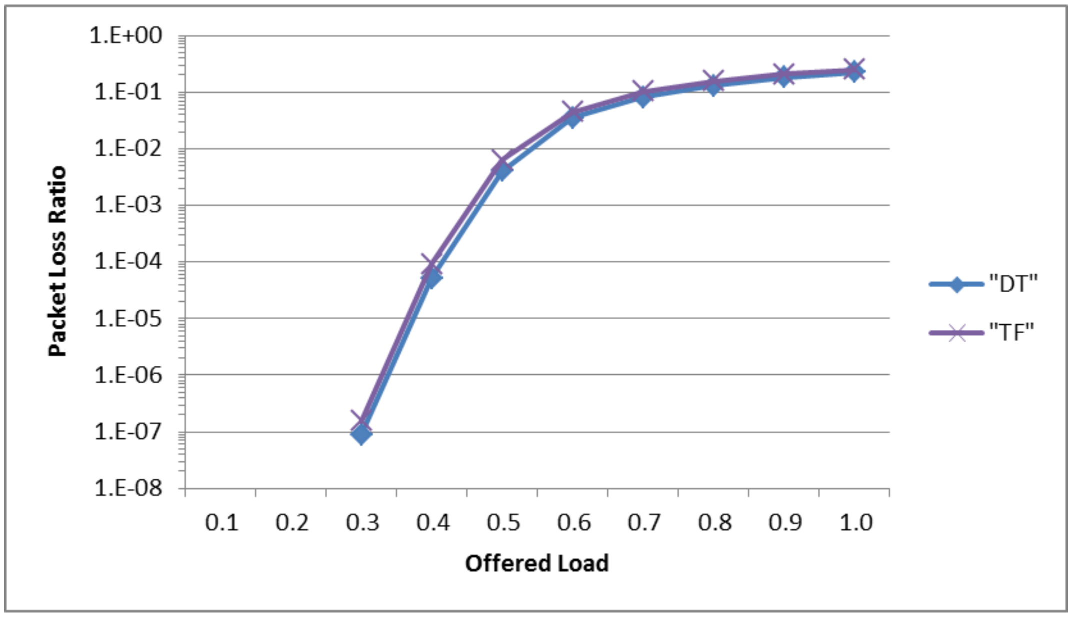


Figure 43-DT vs TF; Ports=48; Buffer=1024; ABL=32

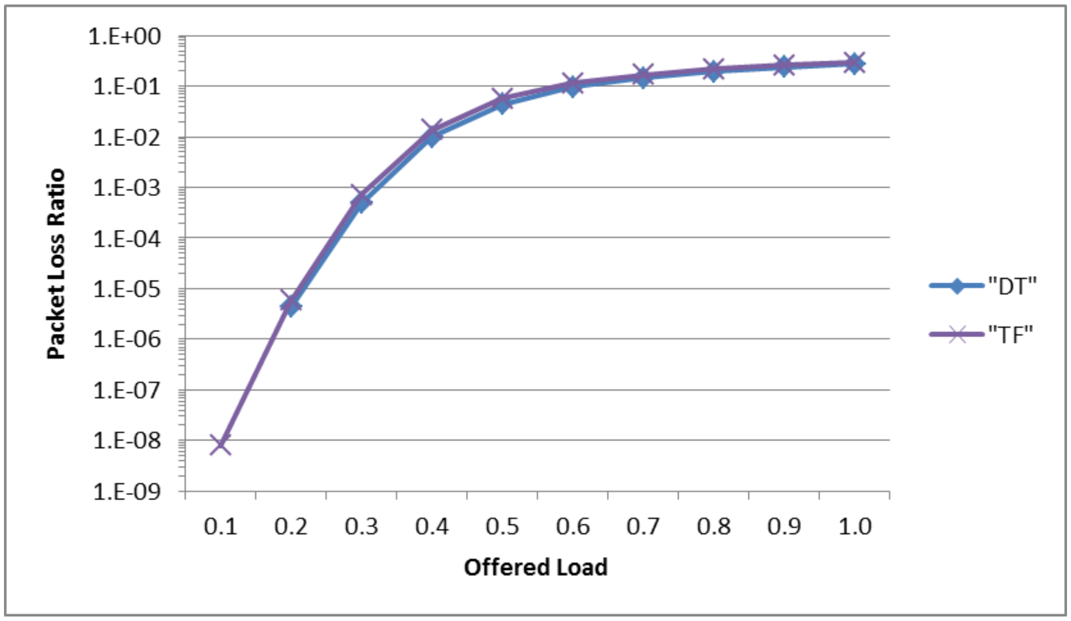


Figure 44-DT vs TF; Ports=48; Buffer=1024; ABL=64

Buffer Versus Ports

We realize when comparing DT versus DT that under all tested buffer versus port ratios DT exceled TF. The amount that DT improved upon the performance of TF varies however. We discovered that for instance in the case when the tested switch was subjected to a load of 100%, and an ABL of 64: when the ratio of available buffer to ports equals 21.33 DT has an improvement of 5.08% over TF; when the ratio to available buffer to ports equals 42.67 DT has an improvement of 8.76% over TF; when the ratio to available buffer to ports equals 85.33 DT has an improvement of 11.49% over TF; when the ratio to available buffer to ports equals 128 DT has an improvement of 8.47% over TF; when the ratio to available buffer to ports equals 170.67 DT has an improvement of 14.15% over TF; when the ratio to available buffer to ports equals 256 DT has an improvement of 10.82% over TF; and when the ratio to available buffer to ports equals 512 DT has an improvement of 11.70% over TF. The results are shown in figures 45 to 49.

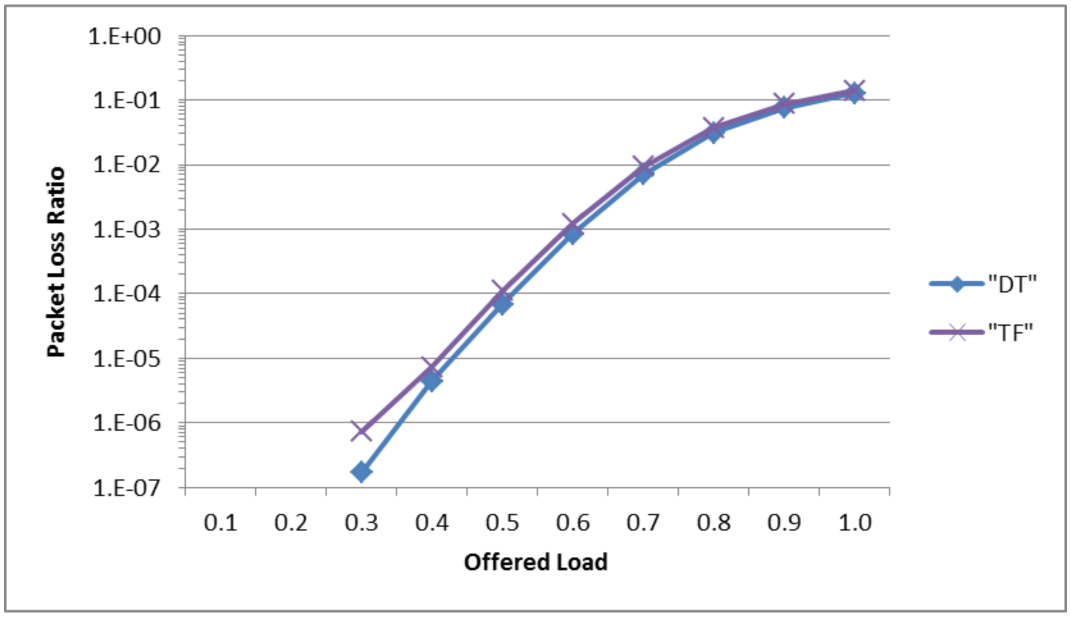


Figure 45-DT vs TF; Ports=8; Buffer=1024; ABL=64

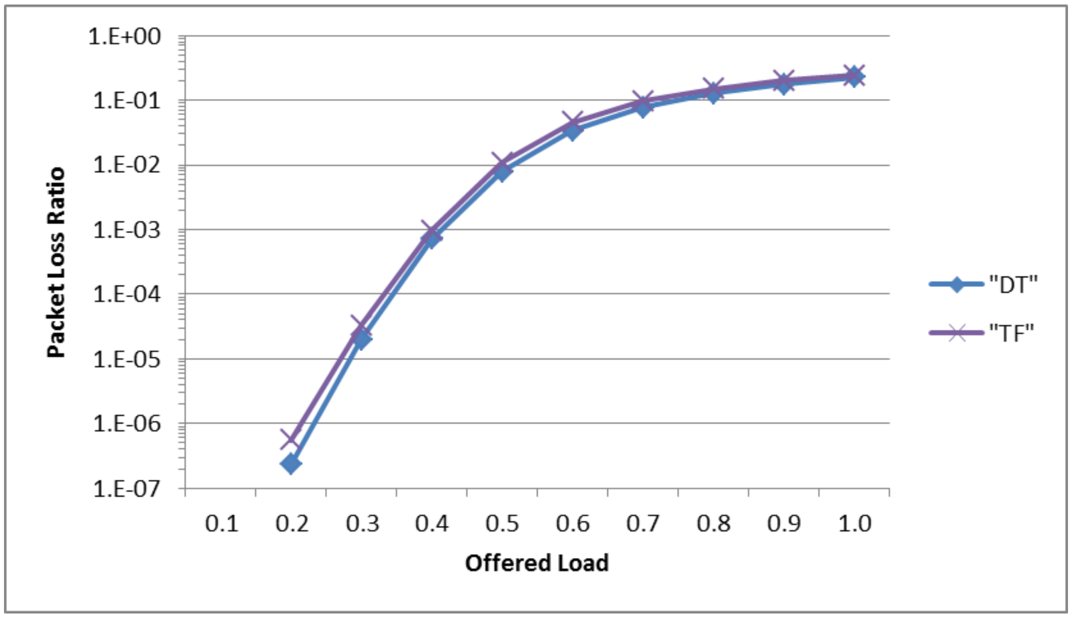


Figure 46-DT vs TF; Ports=24; Buffer=1024; ABL=64

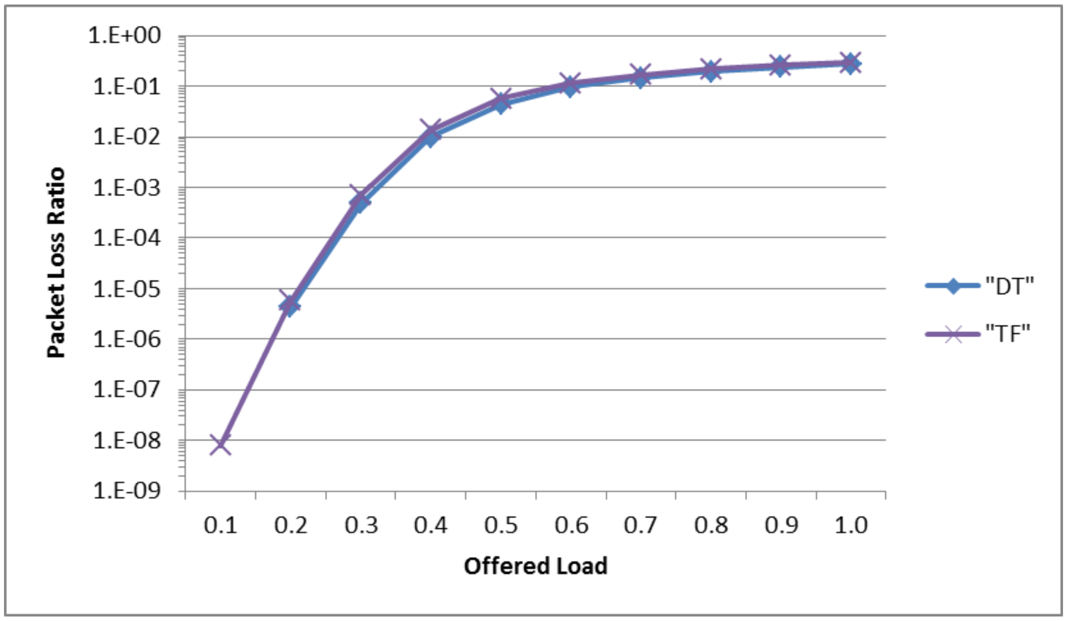


Figure 47-DT vs TF; Ports=48; Buffer=1024; ABL=64

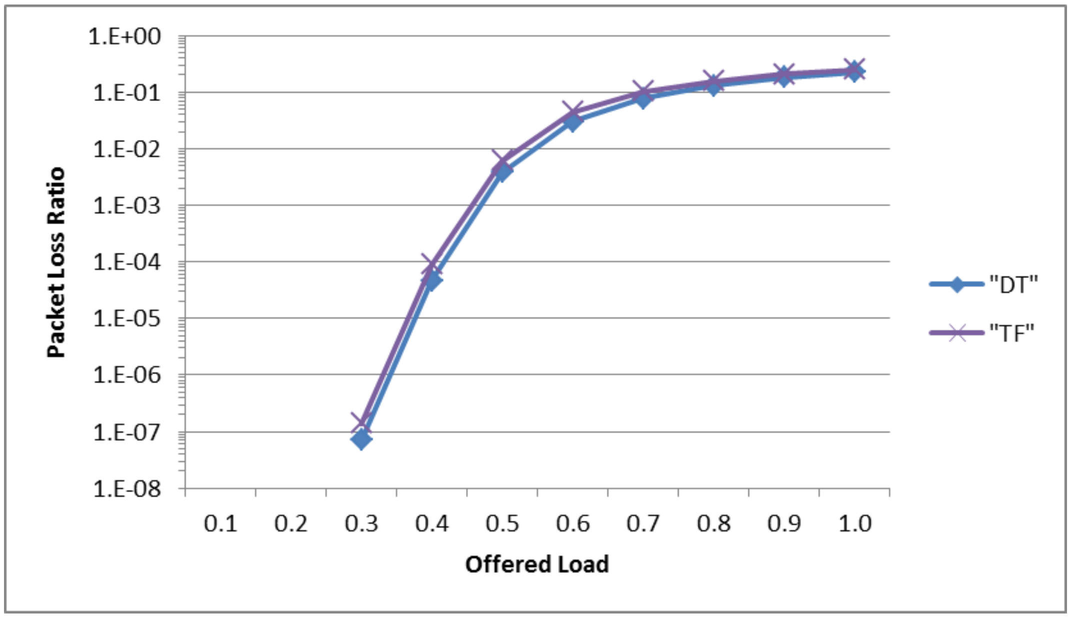


Figure 48-DT vs TF; Ports=48; Buffer=2048; ABL=64

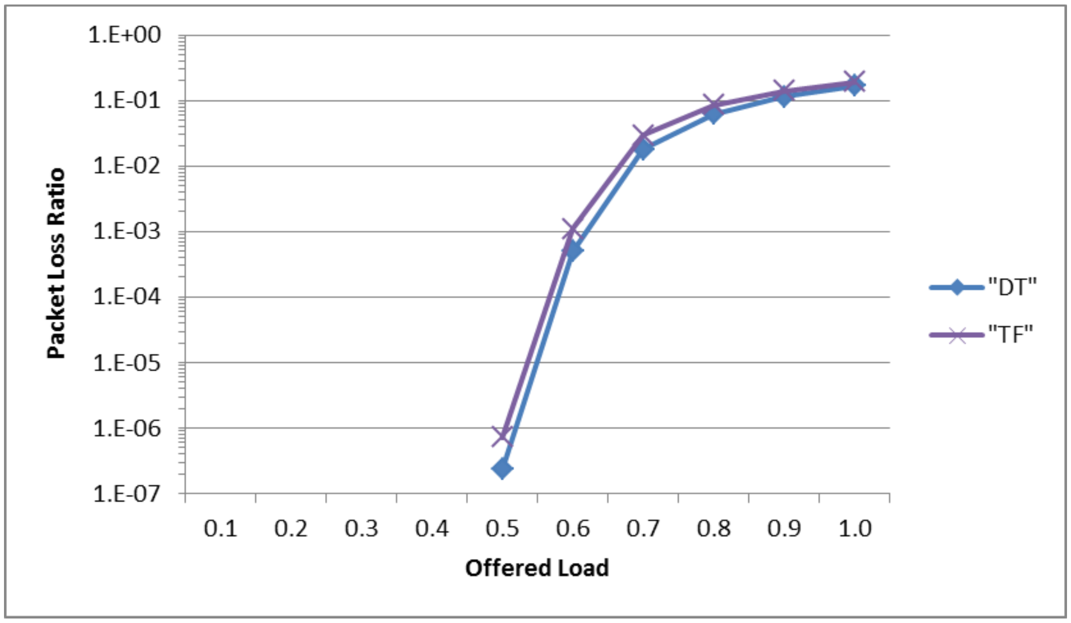


Figure 49-DT vs TF; Ports=48; Buffer=4096; ABL=64

Load

Observing the data gathered from this simulation it was discovered that as the load to which the switch was subjected decreased, the difference between DT and TF increased. Looking at the data collected for the scenario in which the number of ports equals 48, the ABL of the traffic to which the switch was exposed equal 64, and the total buffer available on the switch is for 1024 cells, then we realized that the difference between DT and TF range from 5.08% improvement at loads of 100%, to a 25.63% improvement under the loads of 10%. The results are shown in figures 50, 51 and 52.

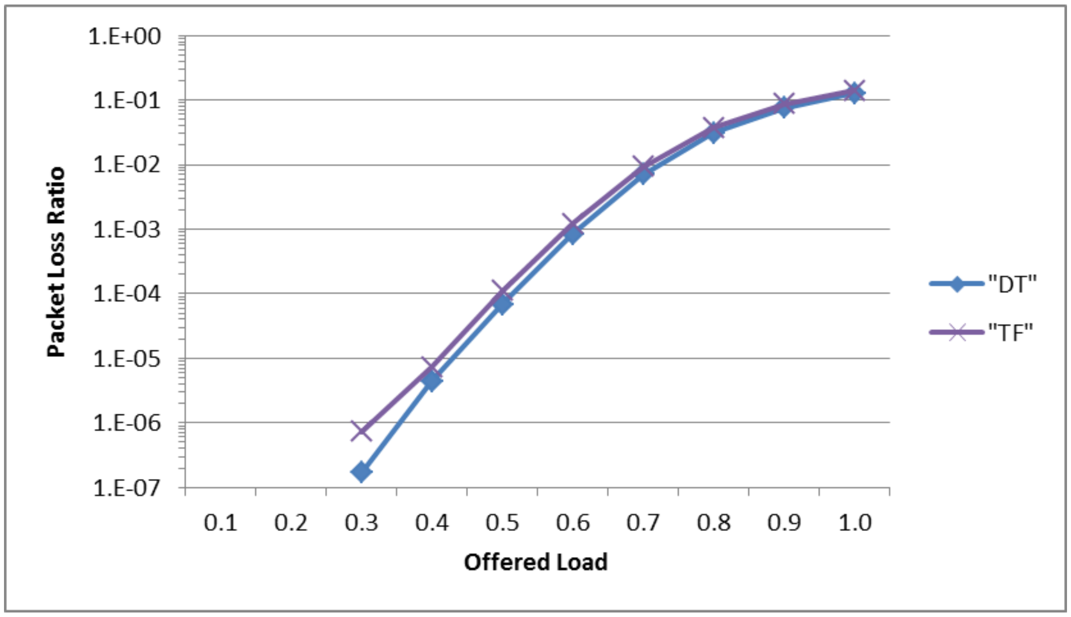


Figure 50-DT vs TF; Ports=8; Buffer=1024; ABL=64

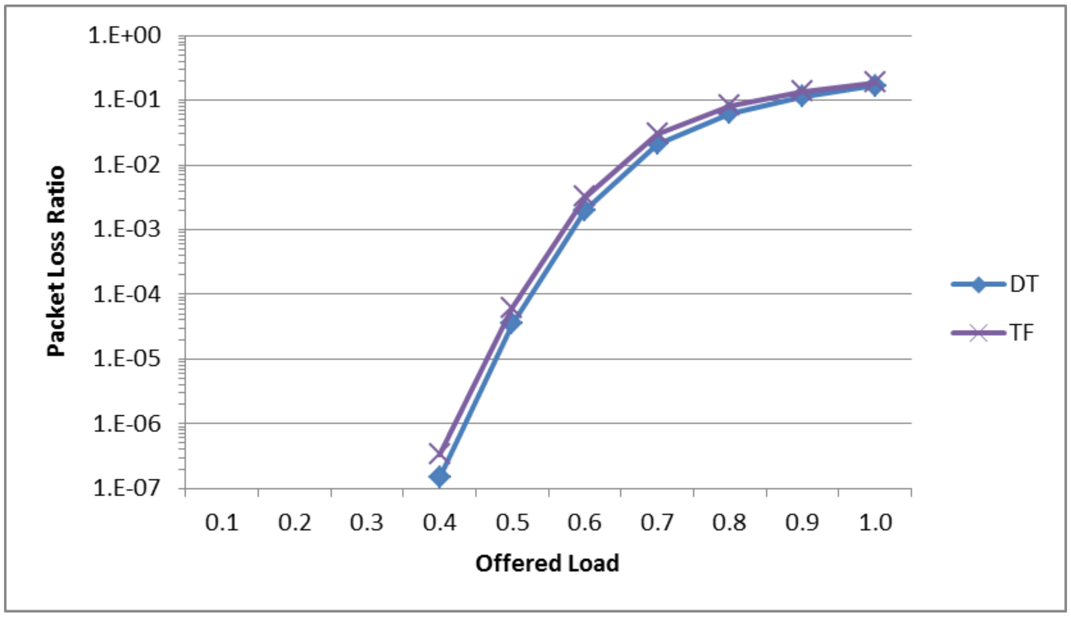


Figure 51-DT vs TF; Ports=24; Buffer=1024; ABL=32

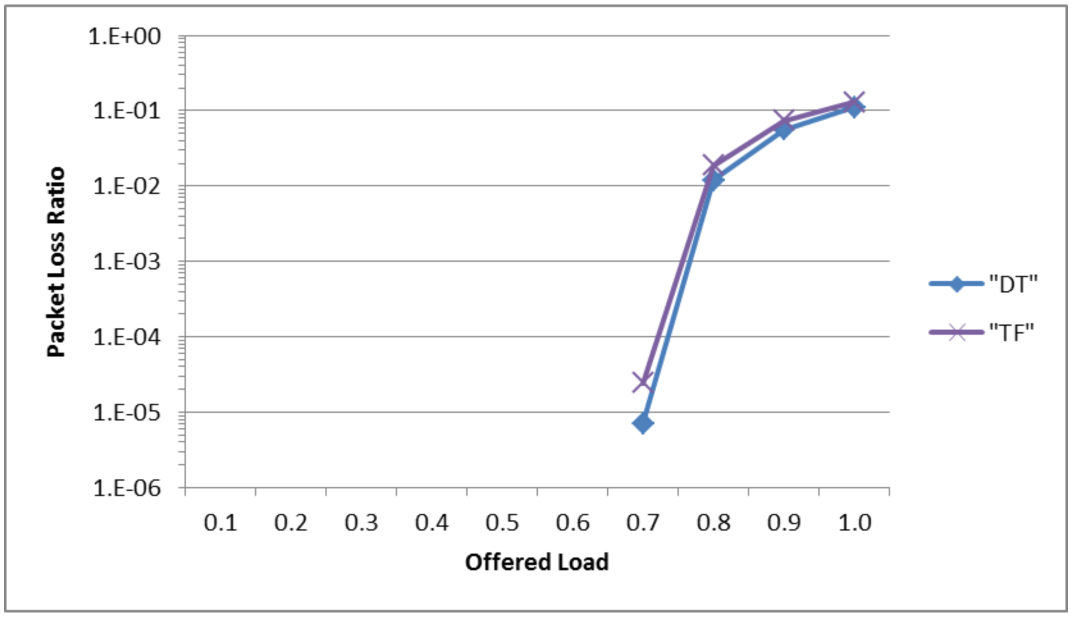


Figure 52-DT vs TF; Ports=48; Buffer=1024; ABL=8

SQFL Versus Conventional Schemes

Average Burst Length (ABL)

We can see how when comparing SQFL against DT that under all tested average burst lengths SQFL outperformed DT. The degree to which SQFL improved upon the performance of DT varies however. We found that for instance in the case when the tested switch was composed of 48 ports, with a total available buffer of 1024 is subjected to a load of 100%. When the ABL of the traffic equals 8 SQFL has an improvement of 1.43% over DT. When we increase the ABL from 8 to 32 there is a slight spike in the performance of DT, reducing the gap in performance to a difference of 1.18%. If the ABL is increased further, this time to 64, then we find that the gap between SQFL and DT widens once more, resulting in a performance difference of 1.36%.

If SQFL is compared against TF, we find that once again SQFL outperforms this conventional scheme. It was observed that as the ABL to which the switch is submitted increases the performance of TF improves. If we look at the case where the switch consists of 48 ports with a total available memory of 1024, and we subject this switch to a load of 100%. We find that for an ABL of 8 SQFL outperforms TF by 15.20%, if we increment the ABL to 32 we find that SQFL outperforms TF by 9.69%, and if we are to subject the switch to an ABL of 64 we find that SQFL outperforms TF by 6.37%. The results are found in figures 53, 54 and 55.

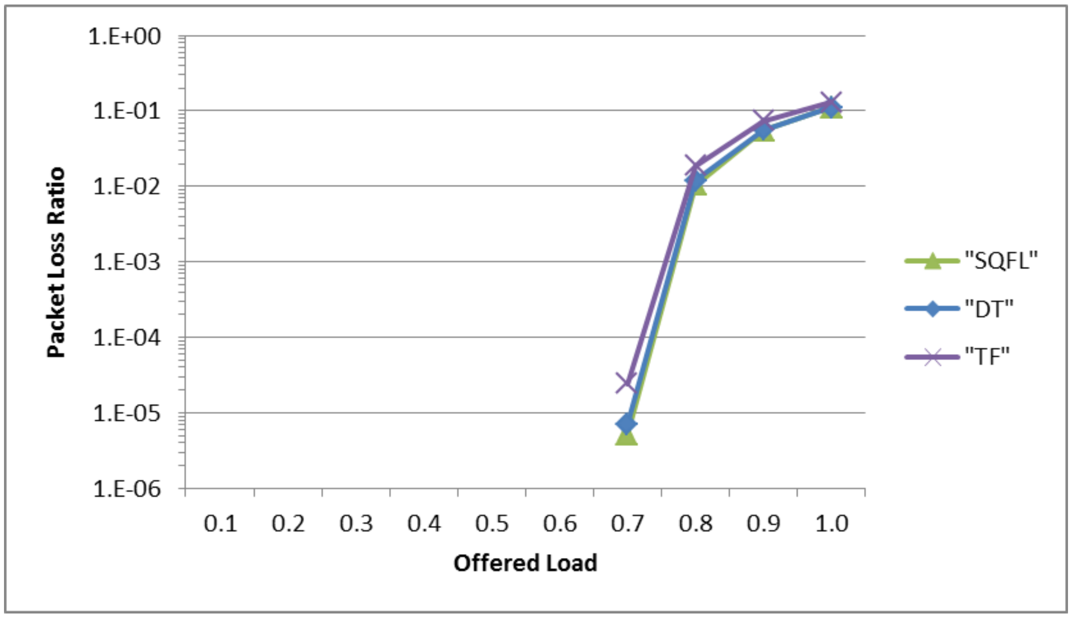


Figure 53-Comparison; Ports=48; Buffer=1024; ABL=8

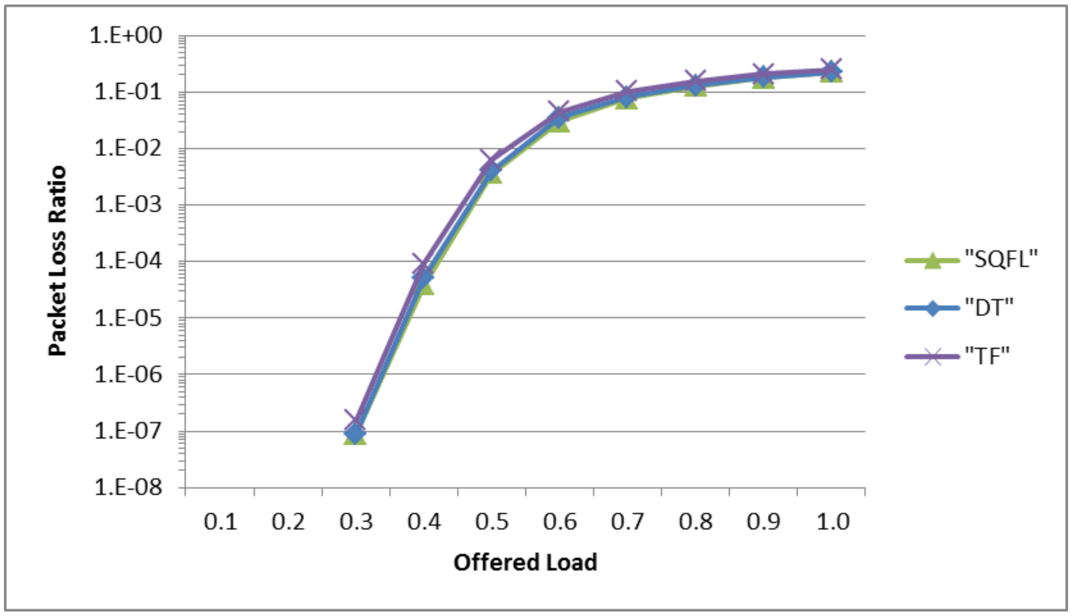


Figure 54-Comparison; Ports=48; Buffer=1024; ABL=32

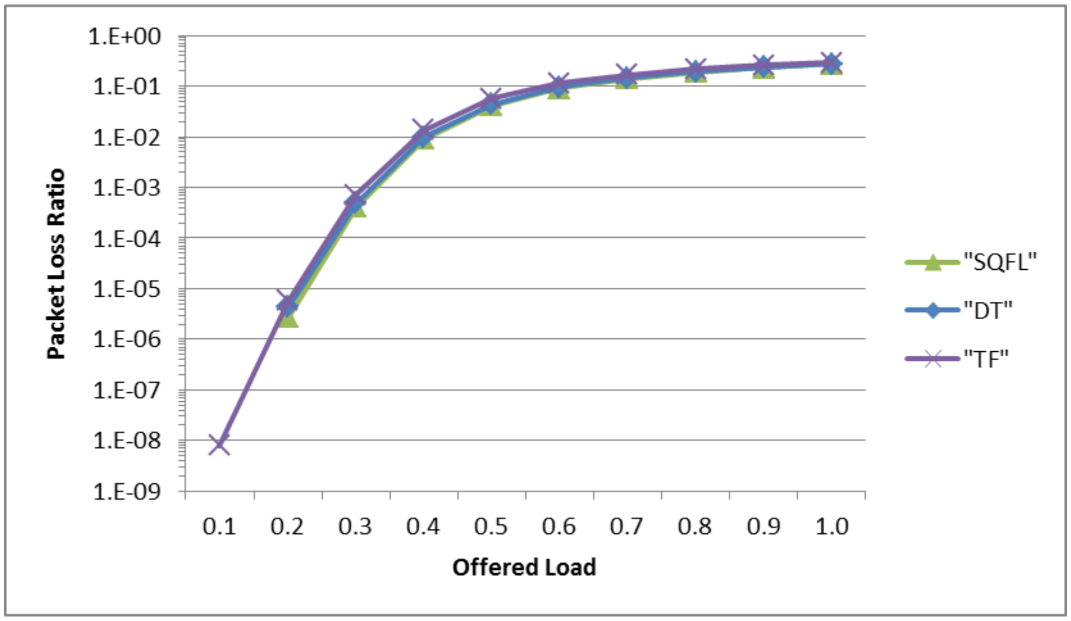


Figure 55- Comparison; Ports=48; Buffer=1024; ABL=64

Buffer Versus Ports

We can see how when comparing SQFL against DT that under all tested buffer versus port ratios SQFL outperformed DT. The degree to which SQFL improved upon the performance of DT varies however. We found that for instance in the case when the tested switch was subjected to a load of 100%, and an ABL of 64: when the ratio of available buffer to ports equals 21.33 SQFL has an improvement of 1.36% over DT; when the ratio of available buffer to ports equals 42.67 SQFL has an improvement of 1.48% over DT; when the ratio of available buffer to ports equals 85.33 SQFL has an improvement of 0.89% over DT; when the ratio of available buffer to ports equals 128 SQFL has an improvement of 1.15% over DT; when the ratio of available buffer to ports equals 170.67 SQFL has an improvement of 0.47% over DT; when the ratio of available buffer to ports equals 256 SQFL has an improvement of 0.10% over DT; and when the ratio of available buffer to ports equals 512 SQFL has an improvement of 0.05% over DT.

We can see how when comparing SQFL against TF that under all tested buffer versus port ratios SQFL outperformed TF. The degree to which SQFL improved upon the performance of TF varies however. We found that for instance in the case when the tested switch was subjected to a load of 100%, and an ABL of 64: when the ratio of available buffer to ports equals 21.33 SQFL has an improvement of 6.37% over TF; when the ratio of available buffer to ports equals 42.67 SQFL has an improvement of 10.11% over TF; when the ratio of available buffer to ports equals 85.33 SQFL has an improvement of 12.28% over TF; when the ratio of available buffer to ports equals 128 SQFL has an improvement of 9.52% over TF; when the ratio of available buffer to ports equals 170.67 SQFL has an improvement of 14.55% over TF; when the ratio of available buffer to ports equals 256 SQFL has an improvement of 10.92% over TF; and when the ratio of available buffer to ports equals 512 SQFL has an improvement of 11.66% over TF. The results are shown in figures 56 to 60.

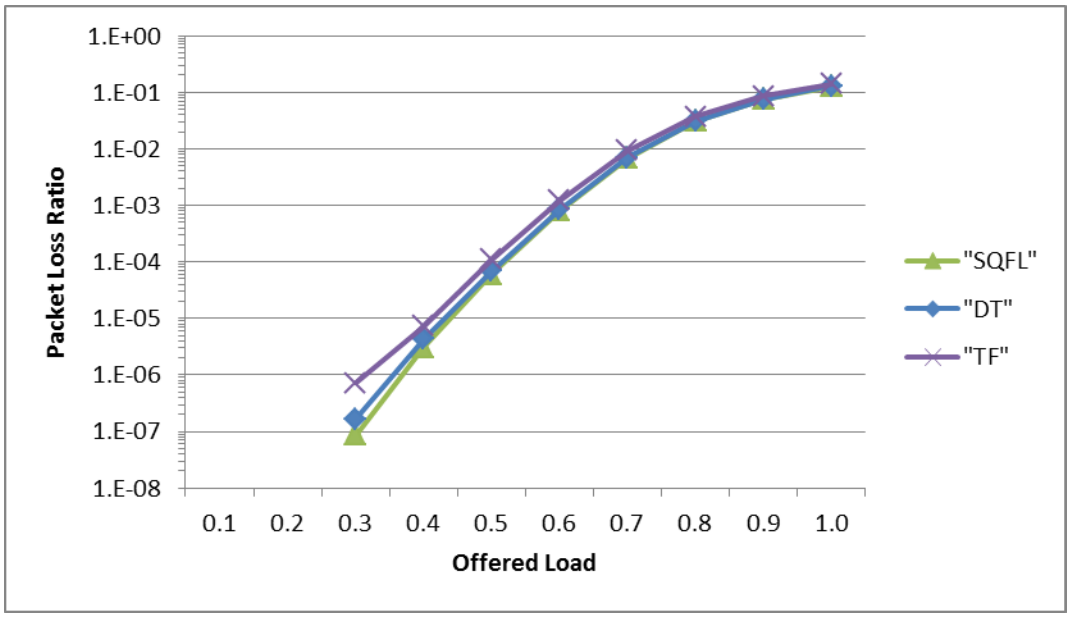


Figure 56- Comparison; Ports=8; Buffer=1024; ABL=64

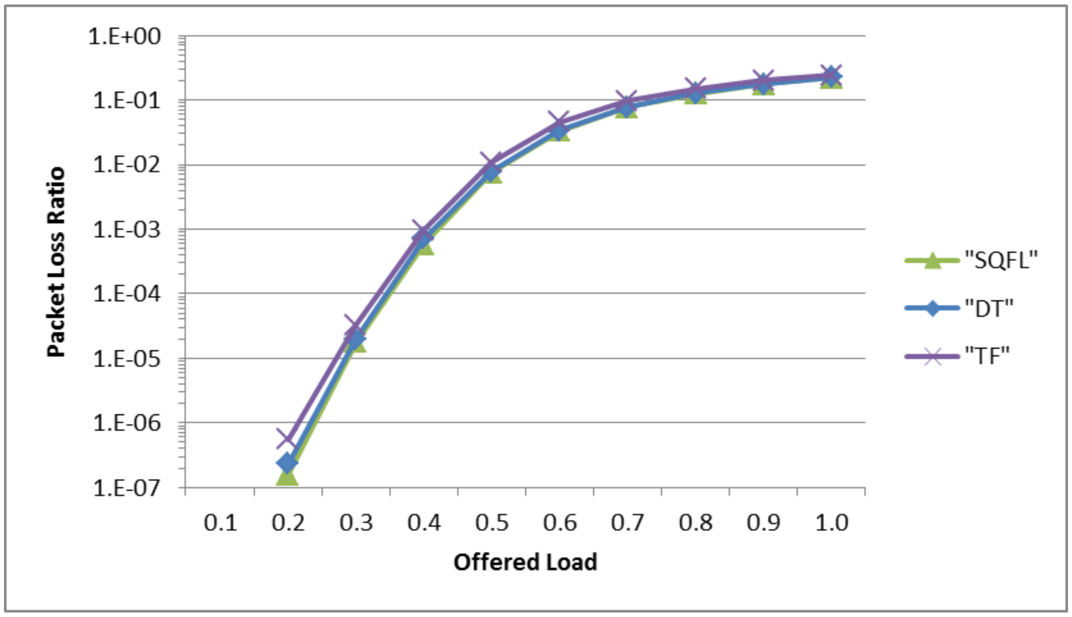


Figure 57- Comparison; Ports=24; Buffer=1024; ABL=64

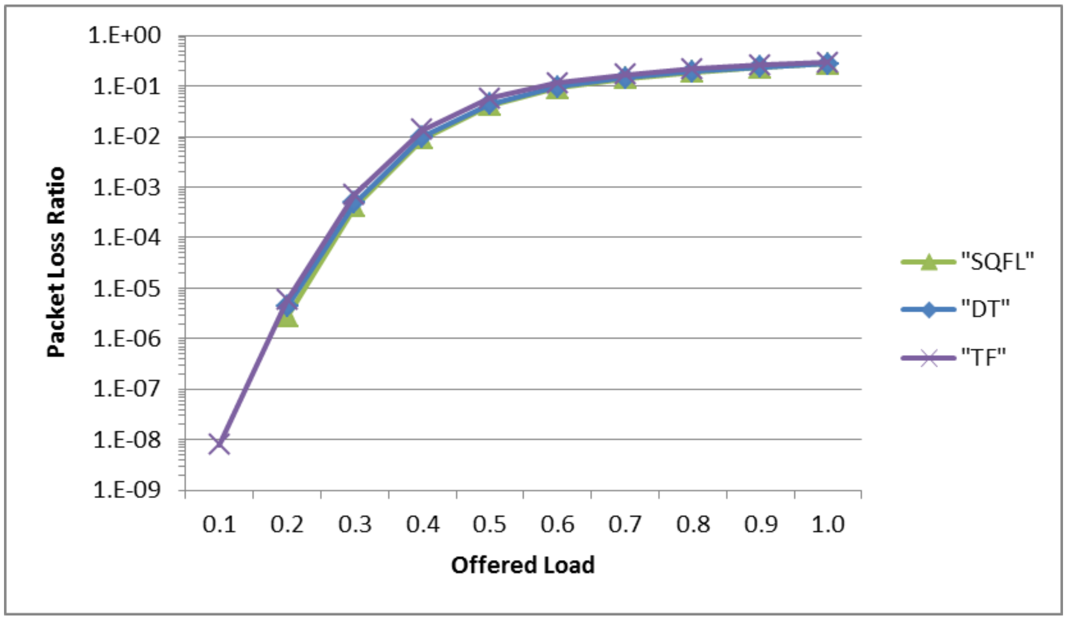


Figure 58- Comparison; Ports=48; Buffer=1024; ABL=64

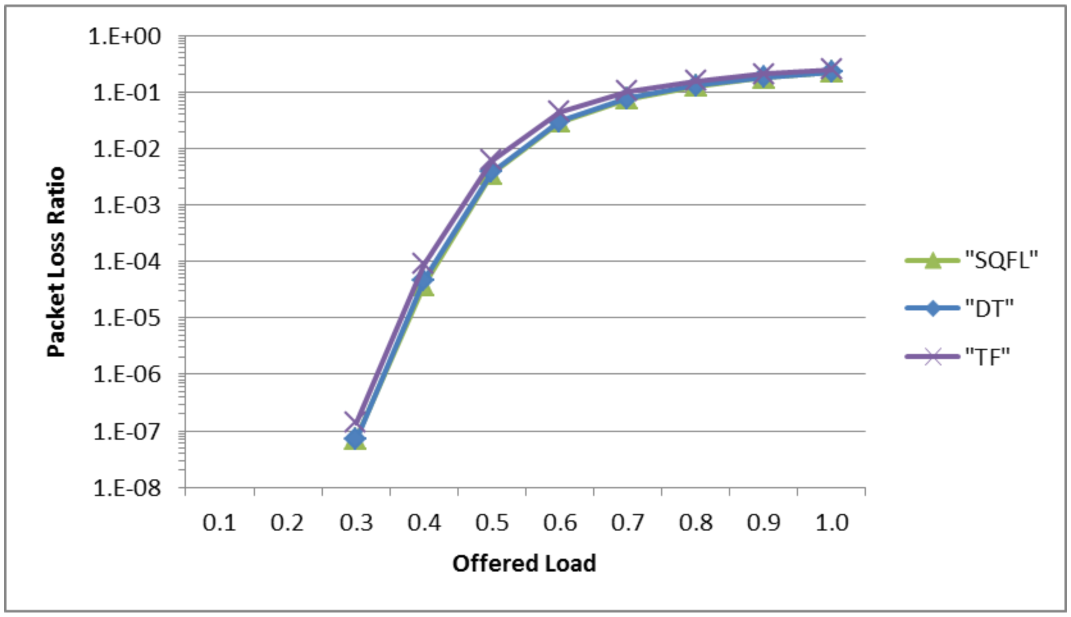


Figure 59- Comparison; Ports=48; Buffer=2048; ABL=64

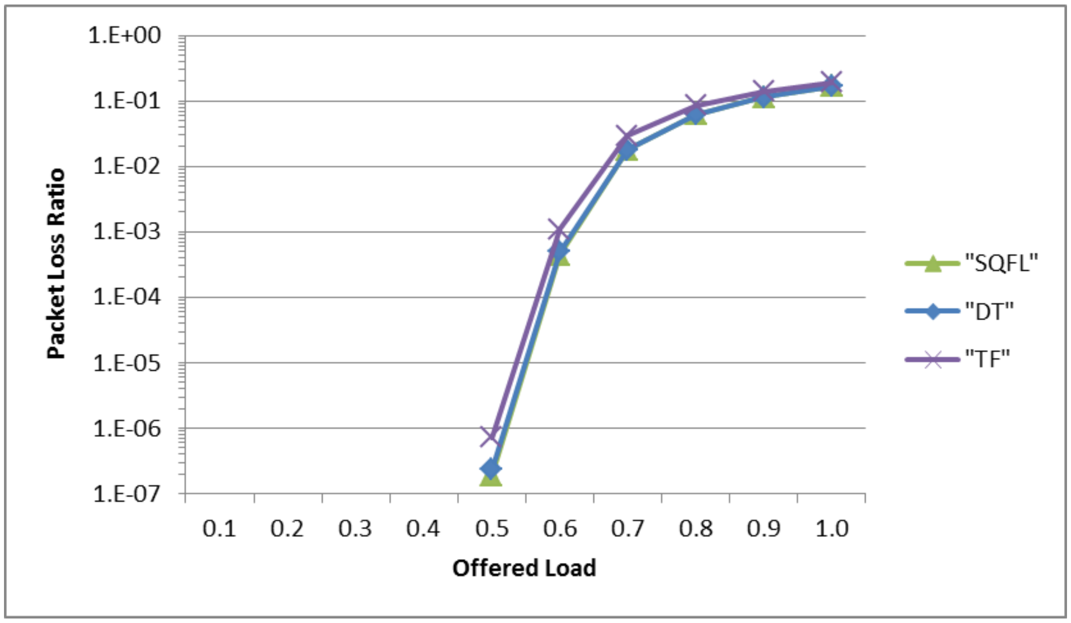


Figure 60 Comparison; Ports=48; Buffer=4096; ABL=64

Load

Looking at the data gathered from the simulation it was observed that as the load to which the switch was subjected decreased, the difference between SQFL and DT and TF increased. If we look at the data collected for the scenario in which the number of ports equals 48, the ABL of the traffic to which the switch is subjected equals 64, and the total available buffer on the switch is for 1024 cells, then we can see that the differences between SQFL and DT range from 1.36% improvement at loads of 100%, to a 36.86% improvement under loads of 10%. for the same scenario in which the number of ports equals 48, the ABL of the traffic to which the switch is subjected equals 64, and the total available buffer on the switch is for 1024 cells, we can see that the differences between SQFL and TF range from 6.37% improvement at loads of 100%, to a 53.04% improvement under loads of 10%. The results are shown in figures 61, 62 and 63.

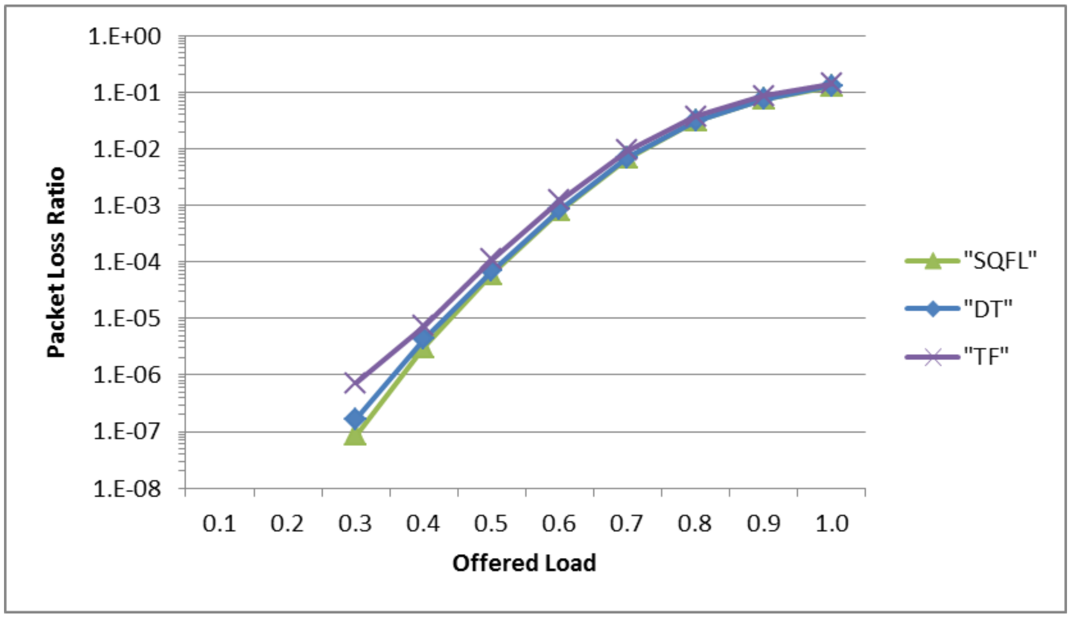


Figure 61-Comparison; Ports=8; Buffer=1024; ABL=64

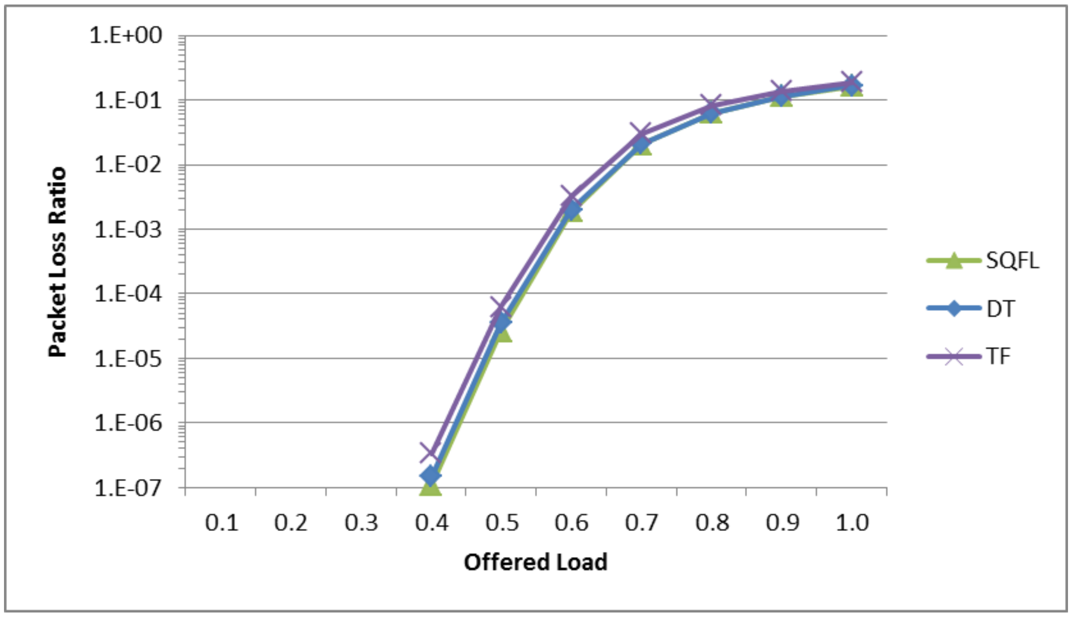


Figure 62-Comparison; Ports=24; Buffer=1024; ABL=32

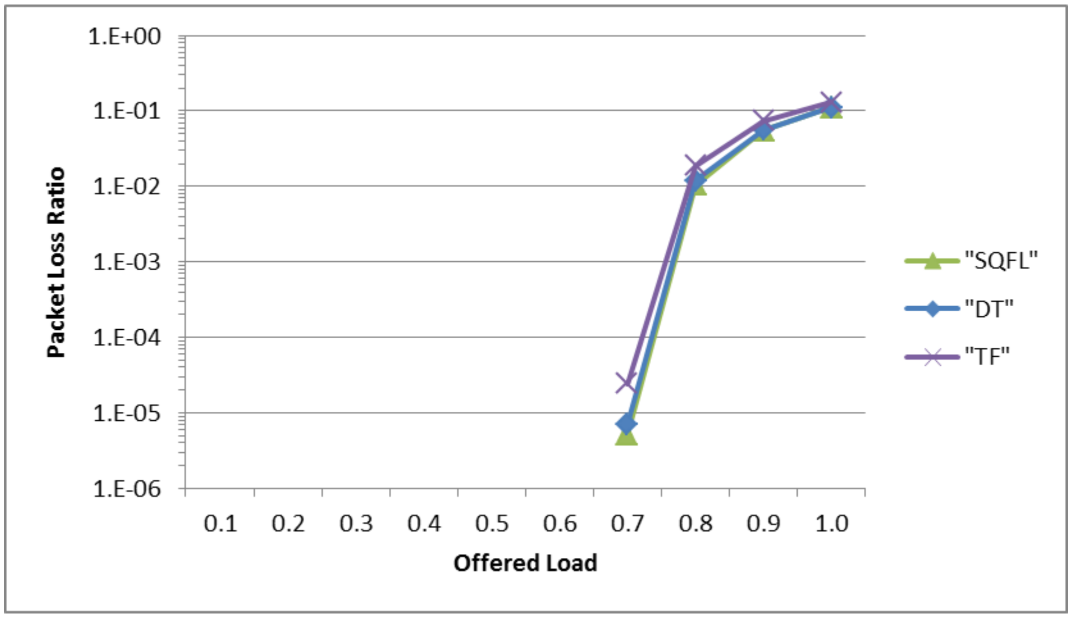


Figure 63-Comparison; Ports=48; Buffer=1024; ABL=8

Summary of Results

When the proposed scheme SQFL was compared to the optimal configurations of both conventional schemes in terms of packet loss ratio, it was found that under each and all tested scenarios SQFL presented the lowest packet loss ratio. This proves that despite having a simpler algorithm than SQF, SQFL is still capable of outperforming the conventional methods of buffering in switches. Table 4 shows the scheme with the lowest packet loss ratio under each of the tested scenarios amongst schemes DT, TF, as well as SQFL.

			L									
Ports	ABL	Bffr	10	20	30	40	50	60	70	80	90	100
8	8	1024	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
8	8	2048	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
8	8	4096	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
8	32	1024	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
8	32	2048	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
8	32	4096	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
8	64	1024	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
8	64	2048	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
8	64	4096	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
24	8	1024	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
24	8	2048	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
24	8	4096	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
24	32	1024	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
24	32	2048	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
24	32	4096	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
24	64	1024	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
24	64	2048	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
24	64	4096	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
48	8	1024	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
48	8	2048	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
48	8	4096	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
48	32	1024	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
48	32	2048	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
48	32	4096	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
48	64	1024	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
48	64	2048	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL
48	64	4096	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL	SQFL

Table 4- Best Scheme

CHAPTER VII

SQFL VERSUS SQF

In this chapter we test and evaluate the proposed scheme Shortest Queue First Lite (SQFL) versus Shortest Queue First (SQF) under different Average Burst Lengths (ABL), different buffer versus ports ratios, and different loads. Also in this chapter, we will show the number of comparison required by SQFL and SQF to sort a list using three different sorting algorithms: bubble sort, merge sort, and radix sort.

Average Burst Length (ABL)

We can see how when comparing SQFL against SQF that under all tested average burst lengths SQF outperformed SQFL, as expected. However, the degree to which SQFL sees its performance lowered versus the performance of SQF is not significant. We found that for instance in the case when the tested switch was composed of 48 ports, with a total available buffer of 1024 is subjected to a load of 100%. When the ABL of the traffic equals 8 SQFL presents a packet loss ratio only 0.11% under SQF. When we increase the ABL from 8 to 32 there is a slight spike in the performance of SQF, increasing the gap in performance to a difference of 0.14%. If the ABL is increased further, this time to 64, then we find that the gap between SQFL and SQF shrinks once more, resulting in a performance difference of 0.12%. The results are shown in figures 64, 65 and 66.

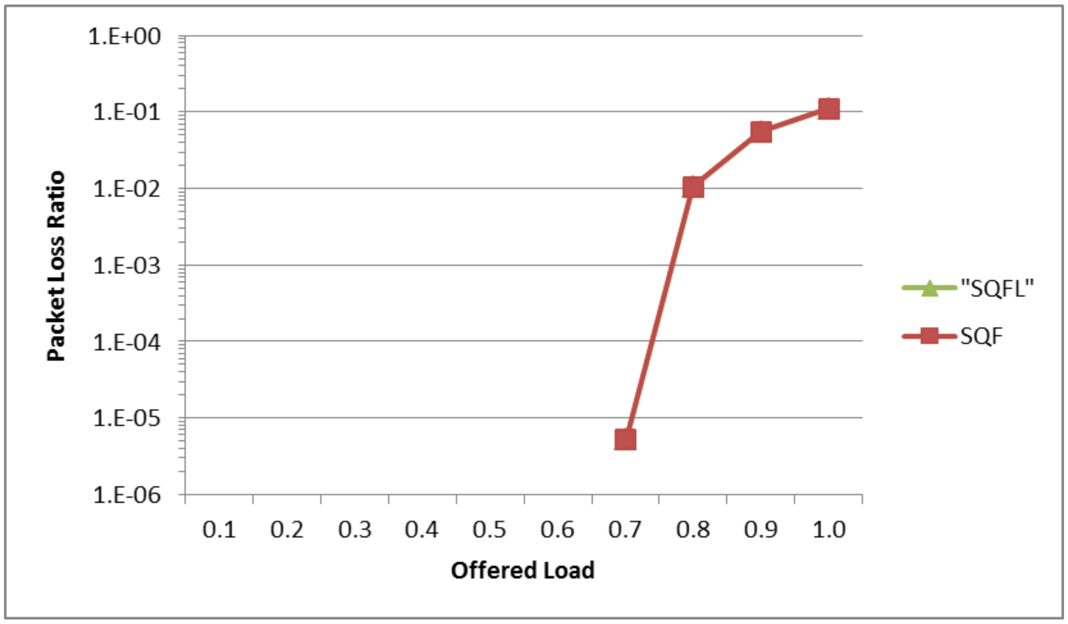


Figure 64-SQFL vs SQF; Ports=48; Buffer=1024; ABL=8

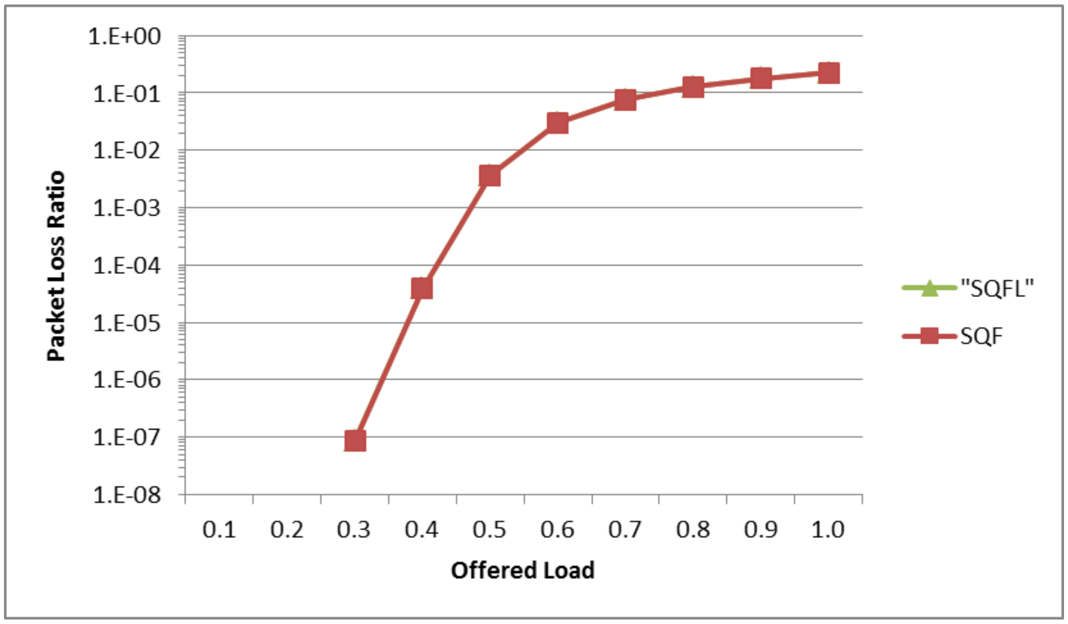


Figure 65-SQFL vs SQF; Ports=48; Buffer=1024; ABL=32

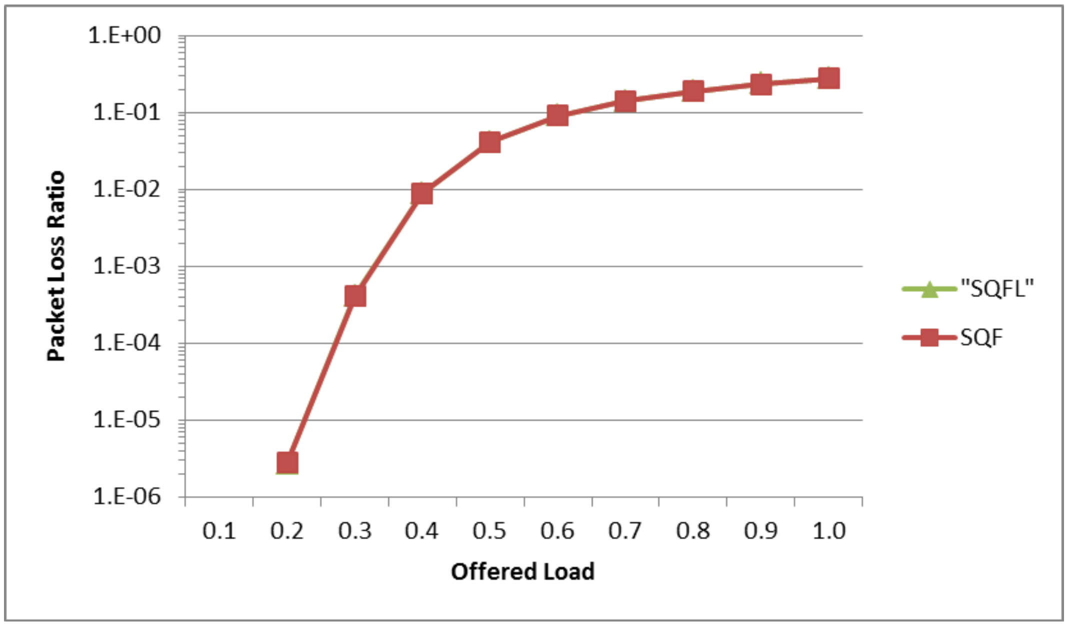


Figure 66-SQFL vs SQF; Ports=48; Buffer=1024; ABL=64

Buffer Versus Ports

We can see how when comparing SQFL against SQF under all tested buffer versus port ratios SQFL is only outperform by SQF by a very small margin. We found that for instance in the case when the tested switch was subjected to a load of 100%, and an ABL of 64: when the ratio of available buffer to ports equals 21.33 SQFL presents a packet loss difference of 0.12% with respect to SQF; when the ratio of available buffer to ports equals 42.67 the difference between SQF and SQFL is 0.07%; when the ratio of available buffer to ports equals 85.33 the difference between SQF and SQFL is 0.05%; when the ratio of available buffer to ports equals 128 the difference between SQF and SQFL is 0.04%; when the ratio of available buffer to ports equals 170.67 the difference between SQF and SQFL is 0.03%; when the ratio of available buffer to ports equals 256 the difference between SQF and SQFL is 0.07%; and when the ratio for available buffer to ports equals 512 SQF outperforms SQFL by only 0.04%. The results are shown in figures 67 to 71.

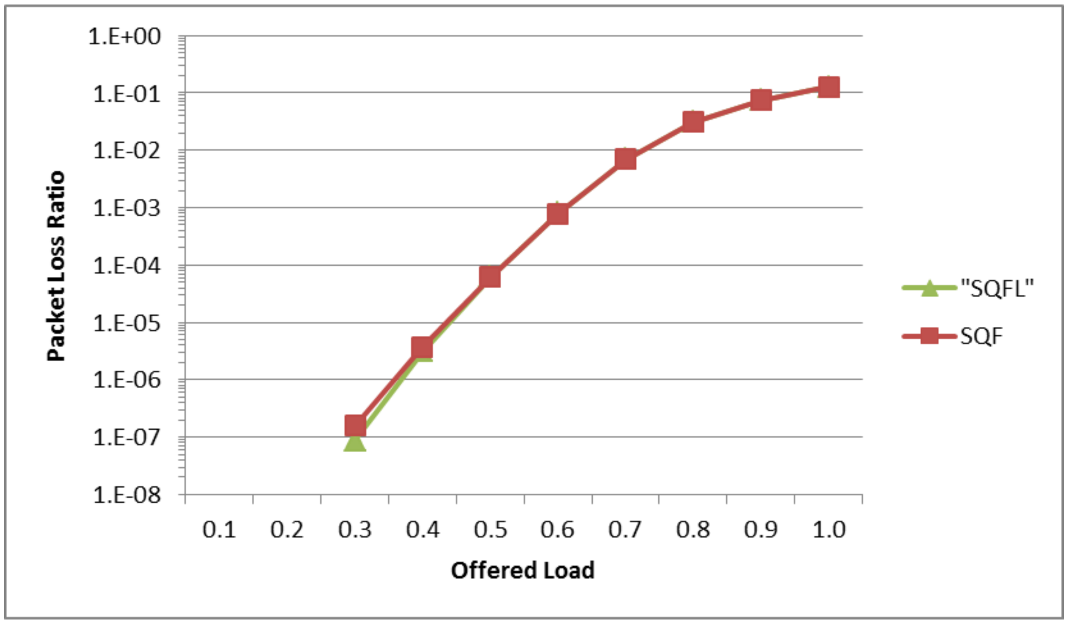


Figure 67-SQFL vs SQF; Ports=8; Buffer=1024; ABL=64

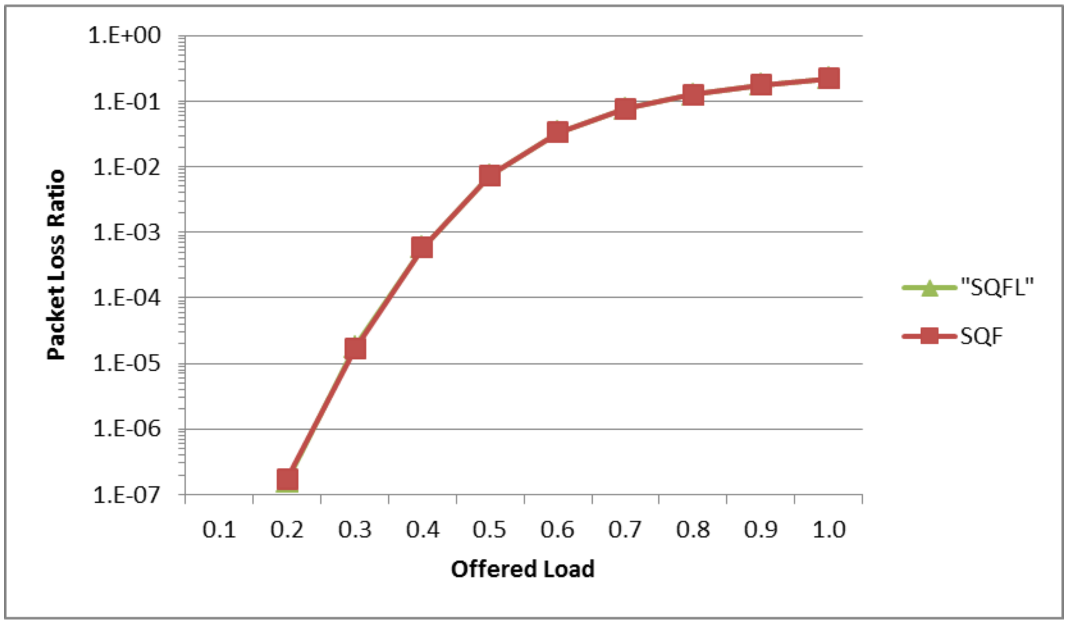


Figure 68-SQFL vs SQF; Ports=24; Buffer=1024; ABL=64

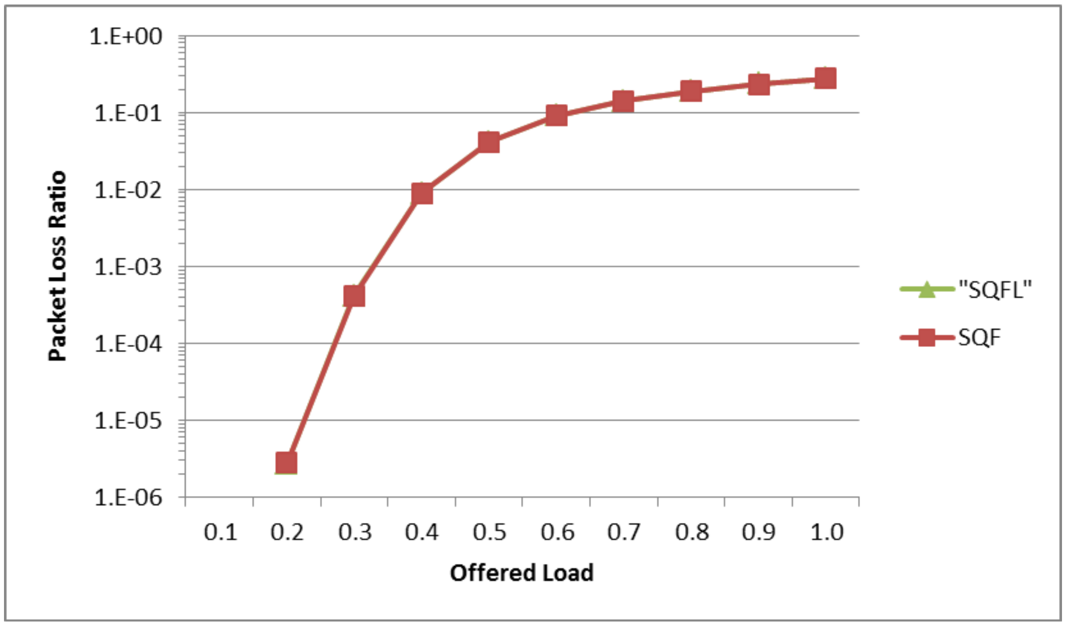


Figure 69-SQFL vs SQF; Ports=48; Buffer=1024; ABL=64

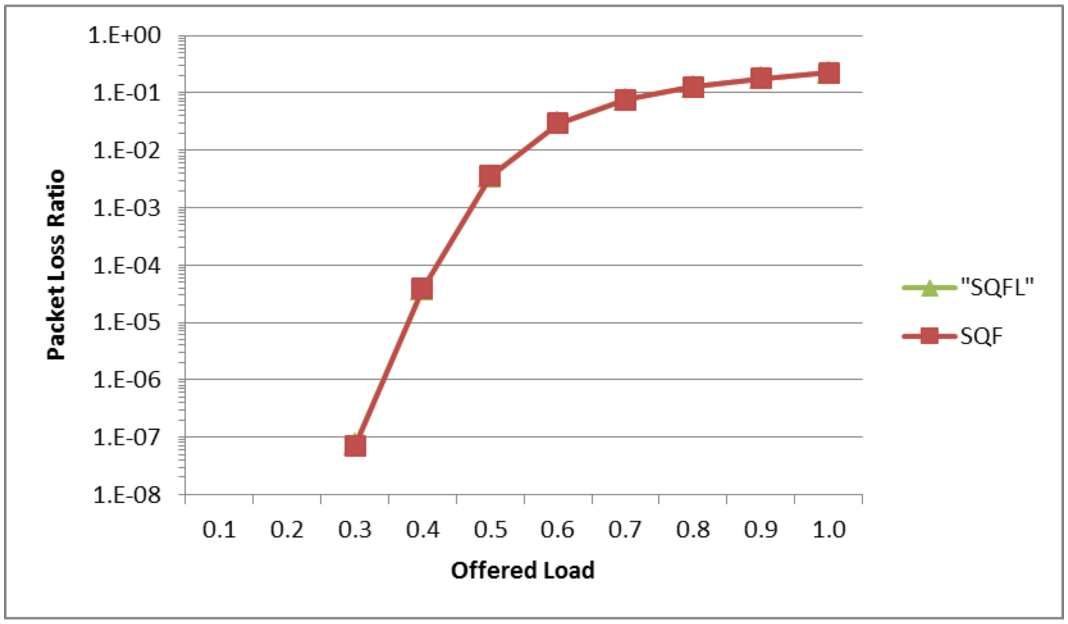


Figure 70-SQFL vs SQF; Ports=48; Buffer=2048; ABL=64

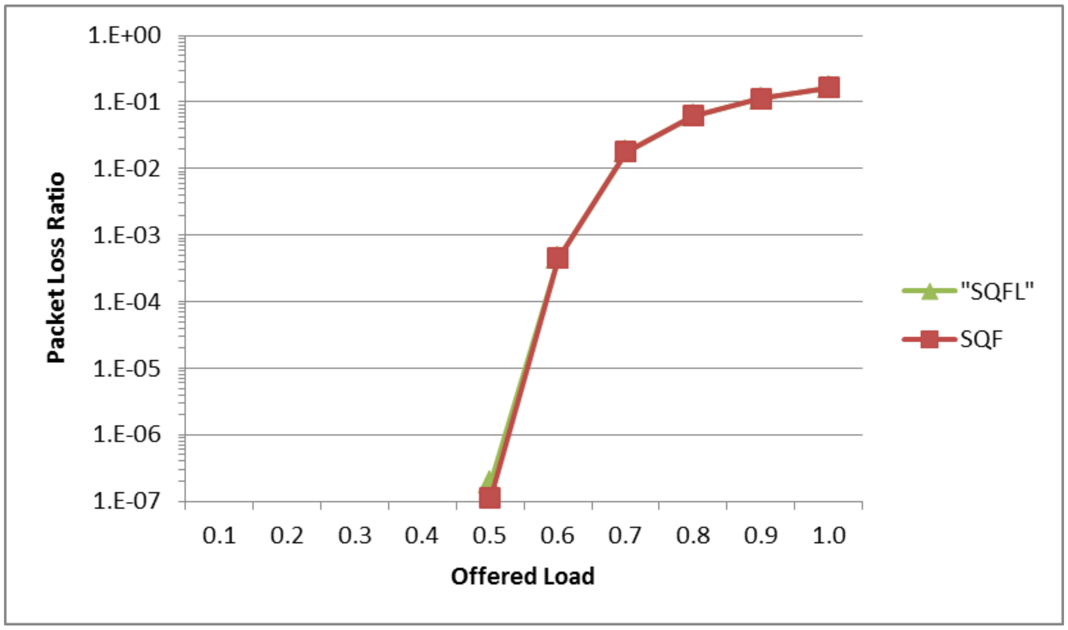


Figure 71-SQFL vs SQF; Ports=48; Buffer=4096; ABL=64

Load

Looking at the data gathered from the simulation it was observed that regardless of the load to which the switch was subjected decreased, the performance of SQFL saw only a minor decreased compared to the packet loss ratio of SQF. If we look at the data collected for the scenario in which the number of ports equals 48, the ABL of the traffic to which the switch is subjected equals 64, and the total available buffer on the switch is for 1024 cells, then we can see that the differences between SQFL and SQF range from 1.07% at loads of 20%, to 0.01 at loads of 50%. The results are shown in figures 72, 73 and 74.

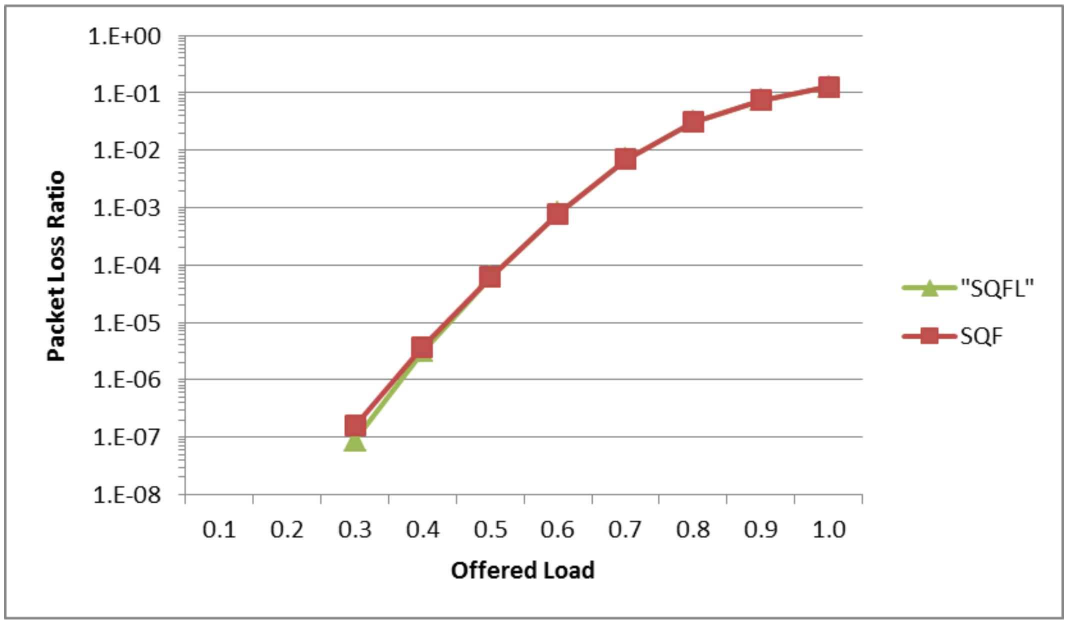


Figure 72-SQFL vs SQF; Ports=8; Buffer=1024; ABL=64

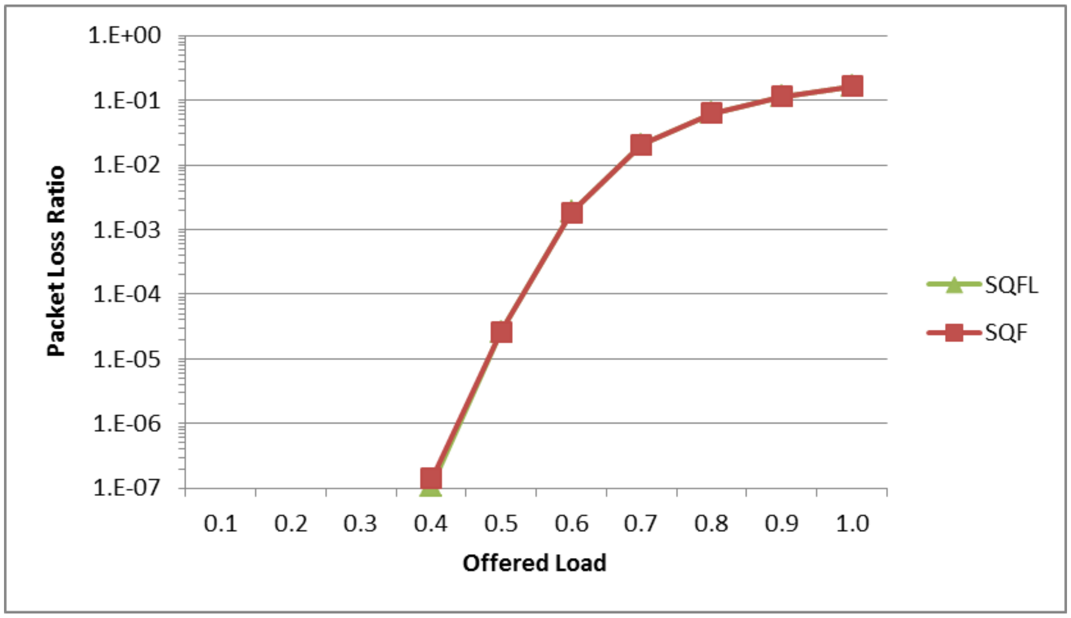


Figure 73- SQFL vs SQF; Ports=24; Buffer=1024; ABL=32

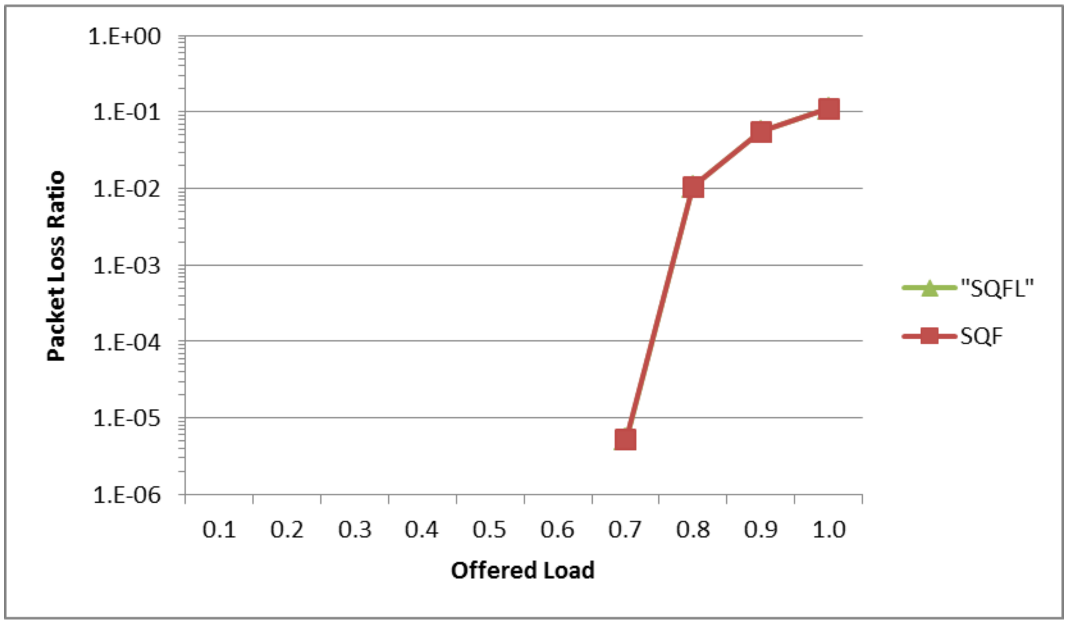


Figure 74-SQFL vs SQF; Ports=48; Buffer=1024; ABL=8

Scheme Complexity

As it was demonstrated in the first three sections of this chapter, SQFL scheme has a packet loss ratio that is comparable to that of SQF scheme. The difference between both of them is negligible, however in terms of complexity is where SQFL scheme improves upon SQF scheme, requiring SQFL to perform a significantly lower number of comparisons in order to sort the switch's priority list each time slot. For instance, using bubble sort in a switch of size 8x8 SQF has to perform 224 more comparisons, or a 87.50% more comparisons than SQFL; in a switch of size 24x24 SQF has to perform 6624 more comparisons, or a 95.83% more comparisons than SQFL; in a switch of size 48x48 SQF has to perform 54144 more comparisons, or a 97.92% more comparisons than SQFL; in a switch of size 64x64 SQF has to perform 129024 more comparisons, or a 98.44% more comparisons than SQFL; in a switch of size 128x128 SQF has to perform 1040384 more comparisons, or a 99.22% more comparisons than SQFL; in a switch of size 256x256 SQF has to perform 8355840 more comparisons, or a 99.61%

more comparisons than SQFL; in a switch of size 512x512 SQF has to perform 66977792 more comparisons, or a 99.80% more comparisons than SQFL; instance in a switch of size 1024x1024 SQF has to perform 536346624 more comparisons, or a 99.90% more comparisons than SQFL; in a switch of size 2048x2048 SQF has to perform 4292870144 more comparisons, or a 99.95% more comparisons than SQFL. Table 5 contains the number of comparisons required by SQF and SQFL to sort the switch's priority list for switches ranging from sizes 8x8 to 2048x2048 making use of the three different sorting algorithms detailed in Chapter III. Figures 75, 76 and 77 depict in a visual manner the data contained in table 5.

Algorithm	Ports	SQF	SQFL
Bubble	8	256	32
	24	6912	288
	48	55296	1152
	64	131072	2048
	128	1048576	8192
	256	8388608	32768
	512	67108864	131072
	1024	536870912	524288
	2048	4294967296	2097152
Merge	8	192	24
	24	2641	110
	48	12868	268
	64	24576	384
	128	114688	896
	256	524288	2048
	512	2359296	4608
	1024	10485760	10240
	2048	46137344	22528
Radix	8	64	8
	24	1152	48
	48	4608	96
	64	8192	128
	128	49152	384
	256	196608	768
	512	786432	1536
	1024	4194304	4096
	2048	16777216	8192

Table 5-Number of Comparisons (Worst Case Scenario)

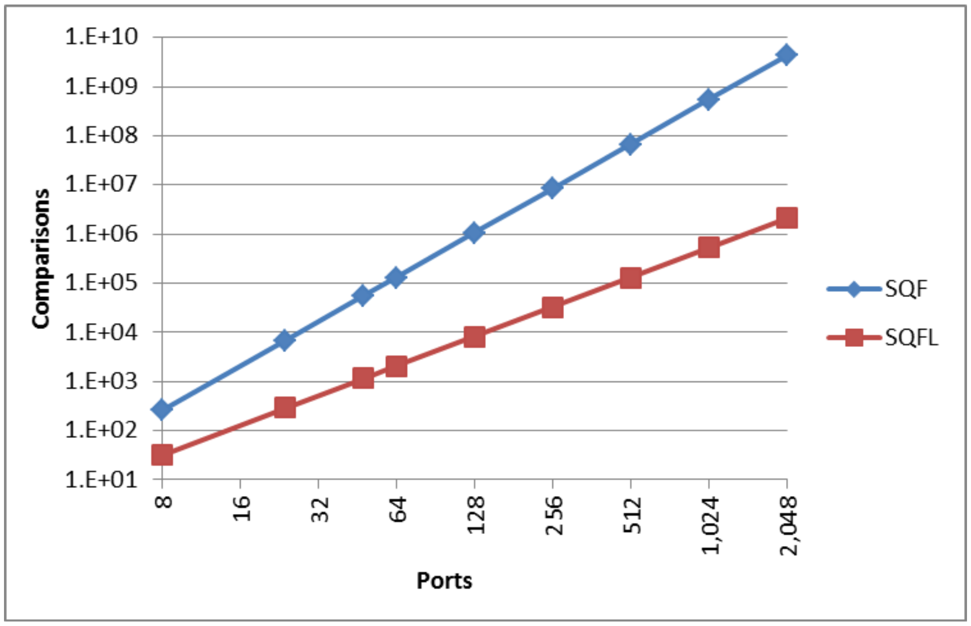


Figure 75-Number of Comparisons: Bubble Sort (Worst Case Scenario)

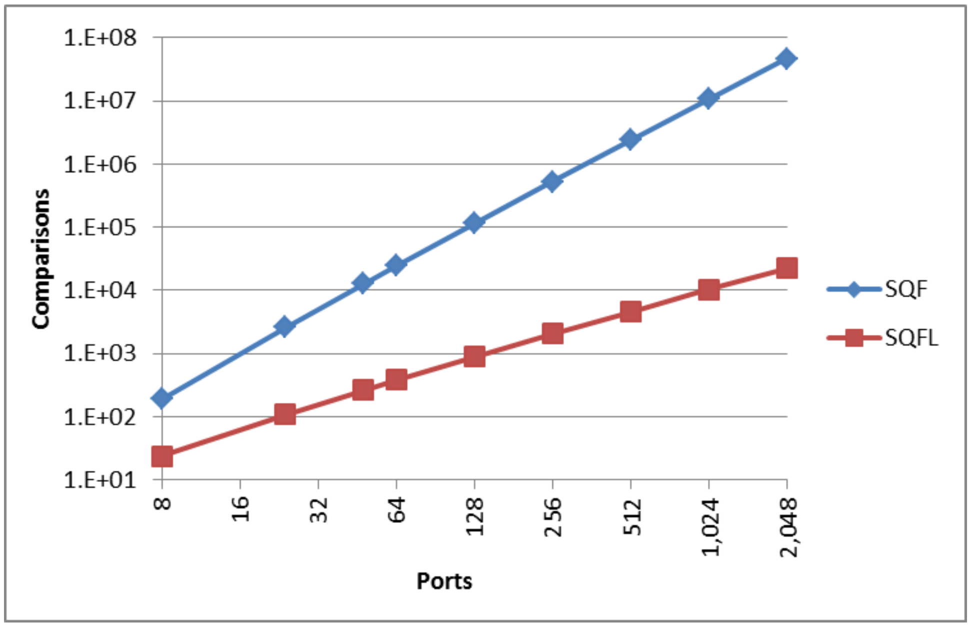


Figure 76-Number of Comparisons: Merge Sort (Worst Case Scenario)

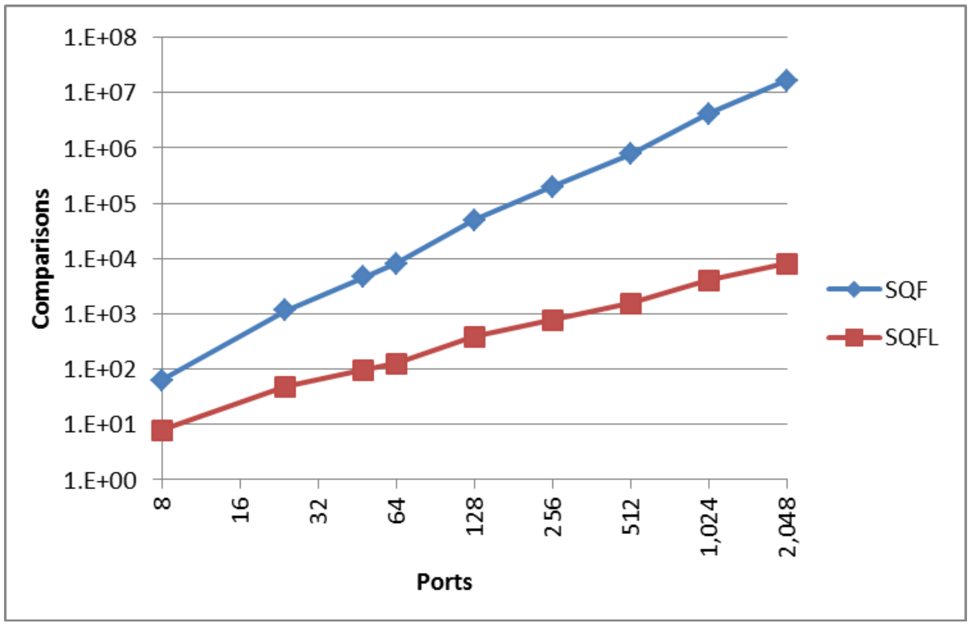


Figure 77-Number of Comparisons: Radix Sort (Worst Case Scenario)

Summary of Results

When the proposed scheme SQFL was compared to the SQF scheme it was found that the difference in packet loss ratio was almost non-existent. As it was shown in Chapter III as well as the previous section in this chapter, SQFL has a lower complexity than SQF, which allows for a lower number of comparisons required to sort the switch's priority list. These qualities make the SQFL scheme not only easier to implement, but also lower in production cost, making SQFL a better alternative compared to SQF scheme.

CHAPTER VIII

CONCLUSION

The primary goal of this thesis was to find a way to facilitate the implementation of Shortest Queue First in hardware. In Chapter III we propose a scheme which we named Shortest Queue First Lite that attempts to facilitate the hardware implementation of Shortest Queue First, simplifying the algorithm originally employed by reducing the number of sorts operations required, which not only reduces the complexity of the sharing scheme, but also reduces manufacturing costs, thus making the proposed scheme much more feasible.

In Chapter IV it was determined the optimal configurations for Dynamic Threshold and Threshold-based Filtering under each of the tested scenarios. This information was then used in Chapter V to compare the two conventional sharing memory versus Shortest Queue First Lite, where it was determined that under each of the tested scenarios the proposed scheme presented a lower packet loss ratio than both Dynamic Threshold and Threshold-based Filtering, proving that the simplified scheme outperforms the conventional schemes.

In Chapter VII the performance of the proposed scheme SQFL is compared against the performance of the SQF scheme; here it was demonstrated that despite the simplification of the scheme, the difference in packet loss ratio between SQFL and SQF were negligible. Also in this chapter it is show how the simplified SQFL scheme greatly reduces the number of comparisons required to sort the switch’s priority list, making SQFL a more viable and attractive alternative to the conventional buffer sharing schemes.

As future work we can suggest the testing and comparison of the conventional schemes, Shortest Queue First, and Shortest Queue First Lite in Multistage Interconnection Networks (MINs), as they are becoming an increasingly common way to build bigger switching fabrics. We also suggest the implementation, as well as testing of the scheme under real world traffic conditions.

REFERENCES

[1] Park, Jae-Sung; Lee, Jai-Yong; Lee, Sang-Bae, "Internet traffic measurement and analysis in a high speed network environment: Workload and flow characteristics," Communications and Networks, Journal of , vol.2, no.3, pp.287,296, Sept. 2000

[2] Ahmad, K.; Begen, AC., "IPTV and video networks in the 2015 timeframe: The evolution to medianets," Communications Magazine, IEEE , vol.47, no.12, pp.68,74, Dec. 2009

[3] "The Internet", http://www.internetsociety.org/internet, Last access 10/23/2014

[4] Park, Jae-Sung; Lee, Jai-Yong; Lee, Sang-Bae, "Internet traffic measurement and analysis in a high speed network environment: Workload and flow characteristics," Communications and Networks, Journal of , vol.2, no.3, pp.287,296, Sept. 2000

[5] Zukerman, M., "Increasing scope for circuit switching in the optical internet," Transparent Optical Networks, 2009. ICTON '09. 11th International Conference on , vol., no., pp.1,4, June 28 2009-July 2 2009

[6] "The Zettabyte Era – Trends and Analysis, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html, Last access 06/10/2014

[7] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014-2019 White Paper", http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html, Last access 02/05/2014

[8] Regennitter Frederick J, Volz John E, “An introduction to the Internet”, American Journal of Orthodontics and Dentofacial Orthopedics, Volume 107, Issue 3, March 1995, pp. 339-344

[9] Clark, Martin P. “Data Networks, IP and the Internet”. England: John Wiley and Sons Ltd. 2003

[10] Stallings William. “Data and Computer Communications”. New Jersey: Pearson Education Inc. 2007

[11] Comer, Douglas E. “Computer Networks and Internets”. New Jersey: Pearson Education Inc. 2009

[12] Blank, Andrew G. “TCP/IP Foundations” September 3, 2004

[13] Mansfield Jr, Kenneth C and Antonakos, James L. “Computer Networking from LANs to WANs: Hardware, Software, and Security” Boston: Course Technology. 2010

[14] Ramamurthy, Byrav, Rouskas, George N. and Sivalingam, Krishna M. “Next-Generation Internet Architectures and Protocols” New York: Cambridge University Press. 2011

[15] Mowery David C, Simcoe Timothy, “Is the internet a US invention? – An economic and technological history of computer networking”, Research Policy, Volume 31, Issues 8-9, December 2002, pp. 1369-1387

[16] Shizhao Li; Chen, J.-G.; Ansari, N., "Fair queueing for input-buffered switches with back pressure," ATM, 1998. ICATM-98., 1998 1st IEEE International Conference on , vol., no., pp.252,259, 22-24 Jun 1998

[17] M. J. Karol, M. G. Hluchyj and S. P. Morgan, “Input versus output queueing on a space-division packet switch,” IEEE Trans. Commun., vol. COM-35, pp. 1347-1356, Dec. 1987

[18] Duato, J.; Flich, J.; Nachiondo, T., "A cost-effective technique to reduce HOL blocking in single-stage and multistage switch fabrics," Parallel, Distributed and Network-Based

Processing, 2004. Proceedings. 12th Euromicro Conference on, vol., no., pp.48,53, 11-13 Feb. 2004

[19] T. Anderson, S. Owicki, J. Saxe, C. Thacker, High Speed Switch scheduling for local area networks, ACM Transactions on Computer Systems, Nov. 1993, pp. 319-352

[20] N. McKeown, Scheduling algorithms for input-queued cell switches, Ph.D. Thesis, University of California at Berkeley, 1995

[21] McKeown, N.; Anantharam, V.; Walrand, J., "Achieving 100% throughput in an input-queued switch," INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE , vol.1, no., pp.296,302 vol.1, 24-28 Mar 1996

[22] Nachiondo, T.; Flich, J.; Duato, J., "Efficient reduction of HOL blocking in multistage networks," Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International , vol., no., pp.8 pp.,, 4-8 April 2005

[23] Duato, J.; Flich, J.; Nachiondo, T., "A cost-effective technique to reduce HOL blocking in single-stage and multistage switch fabrics," Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on , vol., no., pp.48,53, 11-13 Feb. 2004

[24] Nachiondo, T.; Flich, J.; Duato, J., "Destination-based HoL blocking elimination," Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on , vol.1, no., pp.10 pp.,, 0-0 0

[25] Martinez, A; Garcia, P.J.; Alfaro, F.J.; Sanchez, J.L.; Flich, J.; Quiles, F.J.; Duato, J., "A Switch Architecture Guaranteeing QoS Provision and HOL Blocking Elimination," Parallel and Distributed Systems, IEEE Transactions on , vol.20, no.1, pp.13,24, Jan. 2009

[26] Escamilla, J.V.; Flich, J.; Garcia, P.J., "Head-of-Line Blocking Avoidance in Networks-on-Chip," Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International , vol., no., pp.796,805, 20-24 May 2013

[27] Tobagi, F.A., "Fast packet switch architectures for broadband integrated services digital networks," Proceedings of the IEEE , vol.78, no.1, pp.133,167, Jan 1990

[28] Suzuki, H.; Nagano, H.; Suzuki, T.; Takeuchi, T.; Iwasaki, S., "Output-buffer switch architecture for asynchronous transfer mode," Communications, 1989. ICC '89, 93 BOSTONICC/89. Conference record. 'World Prosperity Through Communications', IEEE International Conference on, vol., no., pp.99,103 vol.1, 11-14 Jun 1989

[29] Naoaki Yamanaka, GMPLS Technologies: Broadband Backbone Networks and Systems, CRC Press, 2010

[30] Hluchyj, M.G.; Karol, M.J., "Queueing in high-performance packet switching," Selected Areas in Communications, IEEE Journal on , vol.6, no.9, pp.1587,1597, Dec. 1988

[31] Kozaki, T.; Sakurai, Y.; Matsubara, O.; Mizukami, M.; Uchida, M.; Sato, Y.; Asano, K., "32×32 shared buffer type ATM switch VLSIs for B-ISDN," Communications, 1991. ICC '91, Conference Record. IEEE International Conference on , vol., no., pp.711,715 vol.2, 23-26 Jun 1991

[32] Garcia-Haro, J.; Jajszczyk, A., "ATM shared-memory switching architectures," Network, IEEE , vol.8, no.4, pp.18,26, July-Aug. 1994

[33] Castillo, Javier A. "Cluster Computer Simulation Of Buffer Sharing Schemes Under Bursty Traffic Load" August 2014

[34] Hahne, E.L.; Choudhury, A., "Dynamic queue length thresholds for multiple loss priorities, "Networking, IEEE/ACM Transactions on , vol.10, no.3, pp.368,380, Jun 2002

[35] Pustisek, M.; Kos, A.; Bester, J., "Buffer management in packet switching networks," EUROCON 2003. Computer as a Tool. The IEEE Region 8 , vol.1, no., pp.276,280 vol.1, 22-24 Sept. 2003

[36] Choudhury, A.; Hahne, E.L., "Dynamic queue length thresholds in a shared memory ATM switch", INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE , vol.2, no., pp.679,687 vol.2, 24-28 Mar 1996

[37] “Threshold-based filtering”, http://delivery.acm.org/10.1145/260000/253741/p102-leiner.pdf?ip=129.113.41.55&id=253741&acc=ACTIVE%20SERVICE&key=26DA5B5E47602328%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=560034507&CFTOKEN=29632859&__acm__=1410371613_ab27d62737f1ea3ccf9c8b711ec99575, Last access 10/12/2014

[38] Yang, Jui-Pin; Liang, Ming-Cheng; Chu, Yuan-Sun, "Threshold-based filtering buffer management scheme in a shared buffer packet switch," Communications and Networks, Journal of , vol.5, no.1, pp.82,89, March 2003

[39] Gaddis Tony, Walters Judy, Muganda Godfrey, “Starting out with C++: Early objects”, Pearson/Addison Wesley, 2008

[40] Malik D. S., “Data Structures Using C++”, February 14, 2003.

[41] “C++ Radix Sort”, http://www.sourcetricks.com/2013/03/radix-sort.html#.VTg22CFVikp, Last access 11/11/2014

[42] Kumar, S., "The sliding-window packet switch: a new class of packet switch architecture with plural memory modules and decentralized control," Selected Areas in Communications, IEEE Journal on, vol.21, no.4, pp.656,673, May 2003

[43] Jianliang Gao; Jia Hu; Geyong Min, "Performance Modelling of IEEE 802.15.4 MAC in LR-WPAN with Bursty ON-OFF Traffic," Computer and Information Technology, 2009. CIT '09. Ninth IEEE International Conference on, vol.2, no., pp.58,62, 11-14 Oct. 2009

[44] “Buffer”, http://www.techterms.com/definition/buffer, Last access 09/16/2014

[45] “High Performance Pan-American Cluster”, http://portal.utpa.edu/utpa_main/daa_home/cosm_home/hipac_home/hipac_about, Last access 09/15/2014

BIOGRAPHICAL SKETCH

Wendy Hernandez was born on September 9, 1987. She finished her undergraduate studies at The University of Texas-Pan American in May, 2012. She obtained the degree of Bachelor of Science in Computer Engineering. She finished her Masters of Science in Electrical Engineering also at The University of Texas-Pan American on May, 2015. Her current mailing address is,

1817 Cortez Dr.

Alamo TX 78516