DISSERTATION

Performance Evaluation of All-Optical Switching Architectures with Feedback or Feed-Forward Optical Buffers

Submitted by

Ayman Ghazi Fayoumi

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer, 2005

COLORADO STATE UNIVERSITY

April 11, 2005

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY AYMAN GHAZI FAYOUMI ENTITLED PERFORMANCE EVALUATION OF ALL-OPTICAL SWITCHING ARCHITECTURES WITH FEEDBACK OR FEED-FORWARD OPTICAL BUFFERS BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Adviser

Department Head

ABSTRACT OF DISSERTATION

Performance Evaluation of All-Optical Switching Architectures with Feedback or Feed-Forward Optical Buffers

Optical buffering is employed to avoid collision of optical packets or bursts in all-optical switches. Optical buffering is realized by a set of Fiber Delay Lines (FDLs), which delay packets to avoid packet collision. If two or more packets of the same wavelength arrive at the switch simultaneously and are addressed to same output fiber, all but one of these competing packets have to be switched to the optical buffers to avoid the contention.

Two main optical buffering schemes are studied and evaluated. The first scheme, output optical buffering, is realized by employing feed-forward FDLs at the output ports of an optical switch. A single output buffer is only shared by packets that are addressed to the same output fiber. The second scheme is the optical shared buffering, which is realized by employing feedback FDLs that are shared among all inputs of an optical switch.

Two different packet forwarding algorithms for switches with output buffers are evaluated: a simple forwarding algorithm (SFA) that is easier to implement, and an enhanced forwarding algorithm (EFA) that provides better performance in terms of both probability of blocking and packet average delay. Most of the proposed forwarding algorithms utilize the FDLs under FIFO discipline where the outgoing stream of packets from the buffer can have inter-packet gaps that

are not utilized. The enhanced forwarding algorithm utilizes the output buffer more effectively by filling the inter-packet gaps. Analytical models are derived to evaluate the performance of both algorithms. Simulation results are used to verify the accuracy of both of the analytical models

Considering the feasibility of implementing feed-forward optical buffers, we consider an output optical buffer realized using only a single FDL. The analytical model can be utilized with both packet and burst switching schemes to characterize the performance of the proposed architecture. Results show that the proposed architecture significantly reduces the probability of blocking in an optical switch compared to that without FDLs. The accuracy of the analytical models are verified using simulation results based on a discrete-event simulator that was built using C language. Finally, the same architecture is shown to be capable of supporting Quality of Service (QoS). The results of this work clearly suggest that employing an FDL with the proposed simple forwarding algorithm enhances the performance of the switch, in terms of the loss probability. A further significant enhancement is achievable by employing EFA.

A Surjective-Mapping based Model (SMM) is developed to evaluate the performance of an optical shared buffer switch. The resulting model is accurate, and overcomes the explosion of states that occurs with Markovian based models for moderate to large switches employing shared optical buffers. For example, a Markovian based analysis requires solving a set of 922 equations for a switch with 16×16 nodal degree and 8 feedback FDLs, while a set of only 24 equations is generated using the SMM approach to study the same switch. The SMM provides a complete characterization of the switch including the distribution of the occupancy of the delay lines. A simulator is developed to verify results of the SMM model for switches of different sizes and number of delay lines. The model enables

dimensioning the switch architecture to meet the target performance. Both analytical and simulation models present the probability of blocking as a function of the offered load and number of FDLs. The average delay of the switch, as a function of the offered load and number of FDLs, is also evaluated. Furthermore, the FDL utilization is evaluated as a function of both the offered load and the number of FDLs as well.

Ayman Ghazi Fayoumi
Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, Colorado 80523
Summer, 2005

ACKNOWLEDGEMENTS

I would like to thank my God (Allah) for providing me with all guidance and supports that enable me to finish this work. I also would like to express my appreciation and gratitude to my parents and my wife with whose supports and prayers, I have been able to progress in my graduate studies. Finally, I would like to express my deep gratitude to my adviser Dr. Anura Jayasumana for his inspiring direction and unceasing encouragement and support during the course of my graduate study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

In this chapter, an overview of all-optical networks is provided. Different switching techniques employed by all-optical packet switches are then provided. Illustration of different contention resolution is then presented. The pros and the cons of each scheme are also provided.

1.1 Introduction

Recently, the demand on high-capacity telecommunication networks has increased dramatically due to the explosion of Internet traffic, which has been doubling every four to six months [38]. Bandwidth-intensive applications, such as audio and video applications, are increasingly becoming integral parts of many Internet applications. Additionally, the spectrum of the application-specific requirements for different services is becoming wider, and therefore the underlying networks need to be capable of supporting service diversity. All-Optical Network (AON) is a promising infrastructure that can meet these challenges since it has the potential to handle diverse traffic demanding a very high bandwidth. The switching functionality as well as data transmission in this type of networks is carried out in the optical domain. As such, the bottleneck of optical-electrical-optical conversion can be partially eliminated. Wavelength Division Multiplexing (WDM) scheme seems to be an efficient and feasible approach for implementing AON. Its idea is as simple as sending many streams of data on different wavelengths simultaneously.

1.2 All-optical networks

All-optical networks have received a great deal of attention in the networking field as they provide a huge bandwidth that is data-format transparent. In all-optical WDM systems [37], the low attenuation region of the optical transmission spectrum (e.g. 1.55 micron) is divided into non-overlapping wavelength channels that coexist on a single fiber to support multiple communication channels while each channel operates at peak electronic speed. Channel spacing can be very narrow, as in Dense WDM (DWDM), which is about 0.4 nm, to increase the total number of channels per fiber. Therefore, it is possible to put more than 100 wavelengths in a single fiber to achieve a few *Tbps* aggregated bandwidth. Therefore, as the conventional electronic networks approach their full capacity, optical networks become the most promising scheme to accommodate the ever-growing Internet traffic. Utilizing the ultra-high capacity bandwidth of WDM optical networks shifts the transmission bottlenecks from the Internet infrastructure to the end point electronic machines.

There are two main WDM architectures: broadcast-and-select and wavelength-routed networks. In broadcast-and select, all nodes are connected via a star coupler. At the star coupler, a data stream coming from a node on a certain wavelength gets split equally and broadcast to all other nodes, and the nodes whose receiver is tuned to that wavelength will be able to receive that data stream. In the wavelength-routed networks, each node is an active optical switch which can route an incoming data stream encoded in a certain wavelength to an appropriate output.

1.3 Switching techniques

Optical switches represent the heart of optical networks since they are capable of switching the incoming optical data without converting the whole stream into

electronics, although the header part of the stream may be processed electronically. The typical structure of a switch can be viewed as a node that consists of internal switching elements and a controller. The controller reconfigures the switching elements according to the decision made on where to forward the incoming data stream. There are three main switching techniques in optical networks [38]: circuit switching, packet switching and burst swithcing techniques. Unless otherwise stated, the term "data" refers to the payload of the optical message, while the term "header" refers to the part that is processed in the control unit of the optical node.

1.3.1 Circuit-switched optical networks

In circuit-switched optical networks, a lightpath is established between a source-destination pair for some duration and released at the end of the established session. The source node initially sends a *set-up* request to set up a lightpath to the destination node, while the destination node sends back an acknowledgment to the source. At the end of the session, the source node sends a *release* packet to the destination through the established path to release it. The circuit-switching scheme is suitable for a constant bit rate application that requires connection duration longer than the lightpath set-up time. The main shortcoming of this scheme is that, for many Internet applications, the lightpath may not be fully utilized for the duration of the session as the fraction of the data transmission is smaller than this duration and hence results in poor bandwidth utilization.

1.3.2 Packet-switched optical networks

Optical packet switching technology is an alternative approach for implementing AON. In this scheme, the data is transmitted in an optically encoded format on a packet by packet basis without a lightpath being set up in advance. Each packet carries its own header and the switching takes place on a packet by packet

basis. Since in this case a channel is statistically multiplexed among packets from different sources, the bandwidth utilization is more efficient. One of the challenges in the optical packet switching technique is the requirement of ultra-fast routing logic that is able to concurrently process headers of the incoming optical packets at a rate at least equal to the data rate at which the optical packets are injected into the switch. Additionally, the packet processing overhead is assumed to be relatively large since the size of the control header is large when compared to the payload size. Therefore, this technology is yet to mature and needs further research to overcome a number of technological challenges [23].

1.3.3 Optical Burst Switching

Optical Burst Switching (OBS) seems a promising practical approach to implement AON. In order to reduce the control overhead, the size of the payload in this scheme is enlarged by aggregating multiple packets, intended for the same destination, at the source node to form a single *burst* with a single preceding header that gets switched as a single unit. Moreover, the bandwidth is reserved on-demand for only the duration of the burst and hence efficient bandwidth utilization is achievable. As such, the scheme enables more efficient bandwidth utilization while having low processing overhead. Burst switching is considered to have an intermediate granularity in tems of bandwidth reservation between circuit switching and packet switching optical networks [41].

1.4 Contention resolution in optical networks

In optical packet or burst-switched networks, contention occurs at a switch whenever two or more packets try to leave the switch on the same output fiber using the same wavelength at the same time. In electrical packet-switched networks, this contention is resolved with the store-and-forward technique, which requires the

packets losing the contention to be stored in a memory bank. The stored packets are be sent at a later time when the desired output fiber and wavelength becomes available. This is possible because of the availability of electronic Random Access Memory (RAM). In the optical domain, however, there is no equivalent optical RAM technology. Therefore, the optical packet switching needs to adopt different approaches for contention resolution in the optical domain. The main contention-resolution schemes proposed for optical switches are wavelength-conversion based, space based, and time based scheme.

1.4.1 Wavelength-conversion based resolution schemes

In wavelength conversion [13], the signal on each wavelength from the input fiber is demultiplexed and sent into the switch. The control logic recognizes the occurrence of contention, at which time it selects a suitable wavelength converter leading to the desired output fiber. After the conversion takes place, the converted signal gets multiplexed with other signals at the output port and then leaves the switch. The wavelength converters may either be full-range or limited-range based converters. In the full-range based conversion, the converter is capable of converting any incoming wavelength to any desired wavelength. In limited-range based conversion, the converter is only capable of converting an incoming packet to a subset of the full wavelength set [1].

1.4.2 Space based resolution schemes

Space based resolution schemes [12] are based on schemes such as multiple output fibers or deflection routing. The approach of utilizing multiple output fibers, [29, 34], is conceptually similar to that of the wavelength conversion based scheme; however, physical multiple output fibers connecting the nodes are utilized in this case instead of multiple wavelengths. In the deflection routing scheme, [18, 43], one of the contending packets is assigned the preferred output wavelength

while the others get deflected to the unused wavelengths in non-preferable output ports. The drawback of this approach is that the deflected packet may have to traverse a longer path, which may be on the order of kilometers in WAN networks, to reach the destination resulting in a significant crosstalk component in the optical signal as well as making the network more congested.

1.4.3 Time based contention resolution schemes

Time based contention resolution schemes may be implemented by utilizing fixed-length Fiber Delay Lines (FDLs). In these schemes, an optical packet propagates through a FDL on a *first-in-first-out* basis for a fixed amount of time. The propagation delay is usually chosen to be a multiple of the packet transmission time. Optical buffering schemes that use FDL for contention resolution can be divided into two categories: feed-forward [20, 21], and feedback FDL buffering [5, 10, 23, 28, 33]. In feed-forward buffering, a contending packet is delayed at the output port and leaves the node after the propagation delay of the traversed FDL. In feedback buffering, also called optical shared-buffering, the delayed packet re-enters the node and gets re-processed.

The feedback FDL approach is more costly to implement than the feed-forward FDL architecture, as the former scheme contributes to a higher complexity of input routing logic [40]. From an analytical perspective, modeling the output feed-forward architecture is simpler than that of the feedback architecture. The former can be modeled by considering an isolated output port and the analysis extended to the rest of the output ports, as packet arrivals at that port are independent of packet arrivals at the rest of the output ports [20]. However, with feedback architecture, all FDLs are shared by all packet arrivals including those arriving from the FDLs to the switch. Therefore, a shared-FDL cannot be analyzed as an isolated module due to the interdependency between arrivals from different input ports to

the switch [5]. Nevertheless, feedback architecture achieves better performance in terms of the packet loss rate than that achievable through a comparable feed-forward architecture [29].

The main goal of this dissertation is to develop a thorough understanding of the integration of optical buffering, internal switching components, and control and management mechanisms. Both feed-forward and feedback optical buffering based schemes are considered in this work. It aims at articulating the limitations of implementing optical buffering and assesses these limitations against the achievable performance gain. In addition, the work spans developing a framework of implementing service differentiation and assesses the related performance and control complexity in the optical domain when utilizing feed-forward optical buffers.

The dissertation is organized as follows. Chapter 2 surveys the literature on optical buffering. Chapter 3 presents the problem statements of this work. In Chapter 4, an overview of the simulation techniques used in this work to verify different analytical models developed to characterized the performance of different optical buffering based switch architectures. Chapter 5 provides analytical models for feed-forward-based optical buffering is presented. Chapter 6 presents an analytical model of the feedback-based optical buffering. In both Chapter 5 and 6, results of both schemes are investigated thoroughly. In addition, dimensional aspects of switch architecture are also investigated in both chapters. Finally, a conclusion and future work are provided in Chapter 7.

Chapter 2

PREVIOUS STUDIES AND THE SCOPE OF THE DISSERTATION

2.1 Introduction

All-optical switching is a promising approach to achieve a high-speed data transmission that is transparent and reconfigurable. In such a switching technique, when two or more packets or bursts are addressed to the same output wavelength simultaneously, only one is assigned that wavelength, and the remaining contenders may be dropped. In electronic packet switching, contention resolution is carried out using an electronic store-and-forward buffer, realized using random-access memory (RAM). On the other hand, the lack of an optical RAM that operates in the optical domain represents one of the main challenges in achieving effective contention resolution in all-optical switching. Optical buffering that is realized using fiber delay lines is a foreseeable approach that can partially imitate the functionality of electronic RAM. This buffering scheme, however, differs from the electronic RAM in that a discrete set of delay lines can be utilized for contention resolution, each of which provides the entering packet with a certain delay after which the packet leaves the delay line. This leads to a lower utilization of the output wavelength capacity than the output-link utilization an electronic RAM provides [28]. Nevertheless, all-optical buffering offers the optical switch a reasonable performance enhancement over a similar switch that employs no all-optical buffering.

2.2 Previous work

Optical buffering is realized by utilizing fixed-length Fiber Delay Lines (FDLs). In these schemes, optical packets propagate through each FDL on a *first-in-first-out* basis for a fixed amount of time. The propagation time delay is usually chosen to be a multiple of the packet transmission time. Optical buffering schemes that use FDL for contention resolution can be divided into two categories: feed-forward [20, 21], and feedback FDL buffering [5, 10, 23, 28, 33]. In feed-forward buffering, a contending packet is delayed at the output port and leaves the node after the propagation delay of the traversed FDL. In feedback buffering, also called optical shared-buffering, the delayed packet re-enters the node after the propagation delay of the FDL.

The feedback FDL approach is more costly to implement than the feed-forward FDL architecture, as the former scheme contributes to a higher complexity of input routing logic [40]. Additionally, modeling the behavior of the output feed-forward architecture is simpler than that of the feedback architecture. The former can be modeled by considering an isolated output port, and the analysis extended to the rest of the output ports as packet arrivals at that port are independent of packet arrivals at the rest of the output ports [20]. However, with feedback architecture, all FDLs are shared by all packet arrivals including those arriving from the FDLs to the switch. Therefore, a shared-FDL cannot be analyzed as an isolated module due to the interdependency between arrivals from different input ports to the switch [5]. Nevertheless, feedback architecture achieves better performance in terms of the packet loss rate than that achievable through a comparable feed-forward architecture [29].

Agiltron's DD Series [2] fiberoptic delay lines, shown in Figure 2.1, represents a market version of photonic time delay device. The device selectively guides a

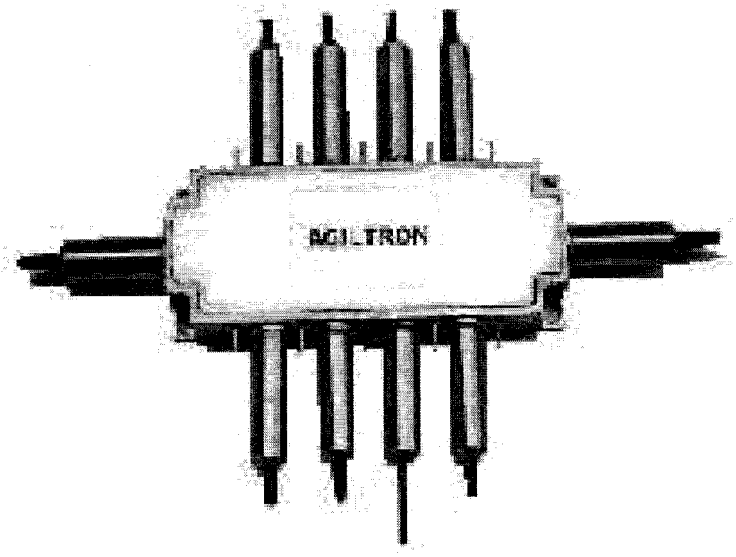

Figure 2.1: Agiltron's generic optical buffer [2]

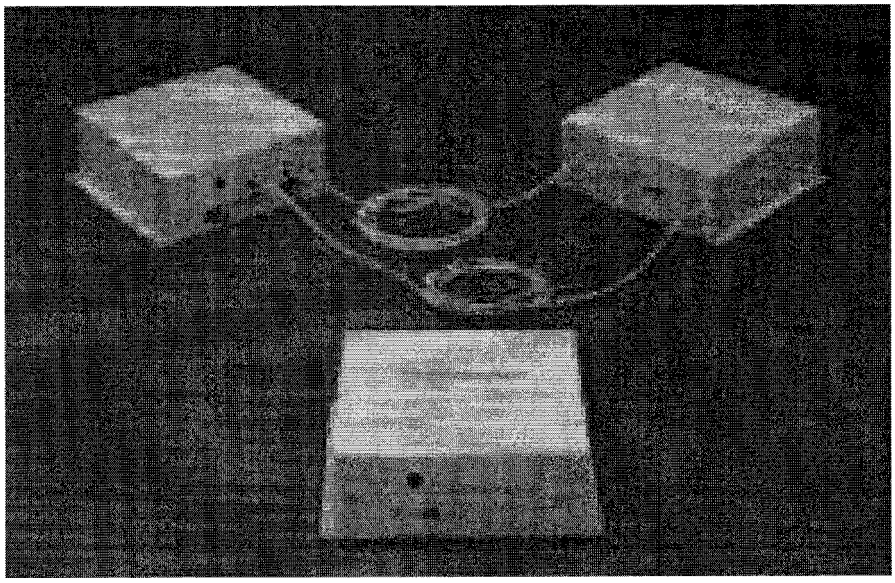

Figure 2.2: Emcore's generic optical buffer [15]

light beam among a combination of 4 fiber loops. These FDLs can provide a delay combination ranging from ΔT to $16\Delta T$, where ΔT is equal to 12 ps. Emcore [15] is a competing firm that implements FDL. Their *5000-Series* FDLs, shown in Figure 2.2, are capable to provide up to 200 ms in an increment of 0.5 ns.

Test beds for optical switches augmented with FDLs took place in projects such as KEOPS and WASPNET [22, 26]. The project in [22] aimed at defining, developing, and evaluating a transparent optical packet switch where experimental demonstration was conducted. The considered switching architecture employs shared feed-forward FDLs and operates in slotted-mode. Additionally, the project in [26] considers switch optical switches with feedback FDLs.

The authors in [28] consider a slotted switch with a feed-forward optical buffer that consists of K FDLs with propagation delays, in terms of the packet size, *1,2,* , K, respectively. A packet arriving at the buffer, while requiring a delay the buffer cannot provide, is dropped. A transform based analysis is conducted in that paper to derive an approximation for the loss probability of switch. On the other hand, our work in Chapter 5 considers an un-slotted optical switch and exponentially distributed packet length where an incoming packet can be delayed for a lower time than the already buffered packet(s) as long as it can be inserted into the packets inter-arrival time.

A simulation based performance analysis of a slotted optical packet switch both employing feedback buffering and full wavelength conversion schemes is presented in [10]. Additionally, simulation results for an asynchronous feed-forward optical switch is presented in [23] considering variable-length packet size. In [29], a hybrid optical switch architecture that combines both feed-forward and feedback FDLs is presented along with a simulation based performance analysis. While a simulation model can be an accurate, it typicall does not depict the actual relationship between the parameters that affect the performance that the analytical

models can present. Therefore, in Chapter 5 and 6, we develop analytical models for both feed-forward and feedback optical-buffering based switches and verify results with simulation-model generated results.

The study in [46] considers synchronous and asynchronous optical switching and evaluates the performance of both feed-forward and feedback optical buffering under a fixed packet size assumption. Both types of buffers have K FDLs with propagation delays, in terms of the packet size, *1,2, , K*, respectively, so that if two or more packets destined the same output enter the FDLs in the same time, no competition among them may take place once they leave. This buffering architecture, however, suggests more hardware complexity of the switch as it requires huge FDLs' lengths. Moreover, the assumption in that study is that a packet to be delayed is buffered in an FDL whose length is more than the number of packets already inside the buffer. In our study, we evaluate the performance of an asynchronous optical switch augmented with a feed-forward output buffer considering variable-size packets' lengths. Furthermore, we study a synchronous optical switch augmented with shred-optical buffer consisting of K FDLs, whose lengths are equal to reduce the switch's hardware complexity.

An analytical model for an asynchronous optical burst switch architecture employing feed-forward optical buffering is presented in [33]. Each output port is augmented with an F FDLs that is capable providing a delay up to B time units. An output wavelength encounters a direct delayed reservation as long as it will be free after the corresponding offset time of the incoming *data burst*. If none of the wavelengths will be free after this offset time, the related *burst header* reserves that wavelength on an FDL that provides the required delay after which the output wavelength becomes free. If none of the FDLs satisfies this delay requirement, the incoming burst gets discarded. The main drawback of this approach is rooted in the fundamental limitations of this type of optical burst implementation. The

delayed reservation of the burst switching suggests more control and management overhead. In addition to this overhead, implementing feed-forward optical buffers also suggests more hardware complexity. Therefore, we consider feed-forward optical buffers in switches employing immediate reservation of output wavelengths, which implies a simpler routing and control logic implementation than that of the case of burst switching.

In [3, 4] studied delay-line based buffer under FIFO. They assumes the incoming packet must be switched to a delay line that provides the minimum delay that exceeds the time needed for the previously accepted packet to entirely leave the output fiber. In contrast, the EFA algorithm implies the FDLs can be utilized differently. Since the inter-packet gaps might be large and hence can accommodate some packets' lengths, the EFA algorithm exploits this property. The algorithm suggests forwarding succeeding packet directly to the output fiber given that there is enough time to for it to leave the switch before the previously delayed packet arrives at the output fiber.

The authors in [39] proposed a similar algorithm to the EFA approach to reduce excess load at the output fiber. However, their proposed algorithm was realized by a shared optical buffer among packet arrivals at all inputs of the switch as well as wavelength converter. Nevertheless, the authors only provide a simulation results of the proposed algorithm while we provided both analytical and simulation results for the first time.

A shared-buffer switching architecture similar to that presented in Fig. 2.3 is analyzed based on a Reduced Markov Chain (RMC) in [5]. This RMC model significantly reduces the number of states compared to that of the Full Markov Chain (FMC) based model previously developed to study similar architectures [36]. While the RMC model provides results similar in accuracy to those of FMC models, in terms of packet loss probability, it is still limited to a switch with a small nodal

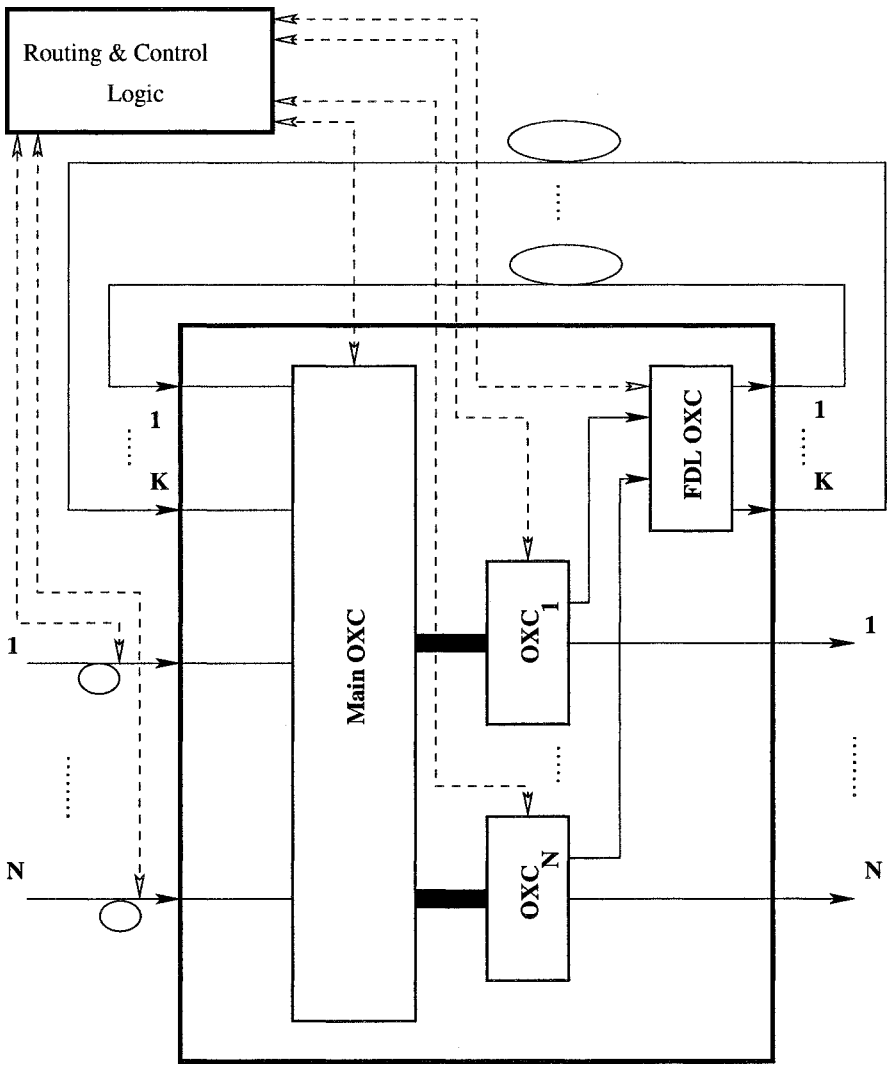

Figure 2.3: Optical packet switch with feed-back FDLs

degree and small number of FDLs. A switch with a nodal degree of $N = 16$ and $K = 8$ FDLs is represented by a 922-state RMC. The authors provide a simplified construction methodology that makes the RMC tractable for a switch of small size. However, the number of states for modeling a switch explodes and becomes untractable for higher values of N and K. Compared to the analysis presented in this work, the performance evaluation of the optical shared-buffer switch presented in [5] provides a complicated approach to characterize the performance metrics of the switch. We follow a new approach, based on a Surjective-Mapping based Model (SMM), to model the shared-feedback FDL optical switch. The model is simple, tractable, and yet provides a comprehensive insight into the performance of the optical shared-buffer switches. The model is also applicable for evaluating large switches. For instance, a set of 922 equations is generated using the RMC approach for a switch with $N = 16$ and $K = 8$, while a set of only $(N + K)$ equations is generated using the SMM approach to study the same switch. Therefore, the model provides a method to evaluate performance of switches significantly larger than what RMC based models can feasibly capture. Moreover, the SMM model provides full characteristics of the traffic flow at different stages inside the switch. Simulation based results are used below to verify the analytical results generated using the SMM based model.

The authors in [44, 45] model a shared feed-forward buffer in an asynchronous optical switch in an architecture requiring two-stage switching fabrics; one stage for switching incoming packets to the switch and another stage for switching packets coming out of the buffer. This architecture suggests a larger buffer size than that we studied in this research. We only consider a single FDL per output.

In [8, 9], the author conducts an analytical model for the feed-forward buffer with B FDLs that provide delays ranging from 0 delay to $D(B - 1)$ bit times, where D is incremental propagation delay. The author, though, approximates the

summation of non-exponentially distributed variables, each of which is the FDL propagation delay and the packet transmission time, to be a Gamma-distributed random variable. Then, for a given number of packets in the buffer, the probability of this Gamma-distributed random variable exceeding the maximum length the buffer can provides is calculated, deeming this probability to be the loss at the buffer. Clearly, considering an FDL with a large propagation delay, adding this propagation delay to an exponentially distributed packet length cannot provide an exponentially-distributed random variable, whose main property is memoryless.

2.3 This research

The main goal of this research is to overcome problems which have not been addressed for both feed-forward and feedback optical buffering schemes.

Most of the previously mentioned studies provide algorithms that try to utilize the FDLs in FIFO discipline. They suggest that, when a packet is forwarded to an FDL, the succeeding packet will be forwarded to an FDL that can provide a delay greater than the time it takes for the currently delayed packet to leave the switch. If such an FDL is not available, this succeeding packet will simply be discarded. Due to the granularity of the FDLs, there will be inter-packet gaps in the stream leaving the FDLs. These gaps represent void periods that contain no packets. As such, we proposed enhanced forwarding algorithm (EFA) enables utilizing the FDLs more effectively. Since the inter-packet gaps might be large and hence can accommodate some packets' lengths, the EFA algorithm exploits this property. The algorithm suggests forwarding succeeding packet directly to the output fiber given that there is enough time to for it to leave the switch before the previously delayed packet arrives at the output fiber. Moreover, for the first time, we provide a comprehensive study with both analytical and simulation models to analyze the EFA. We also suggested a simple forwarding algorithm that is simpler

to implement along with the related analytical and simulation performance results. Taking into account that the length of a fiber delay line (FDL) can be remarkably large when considering an FDL with a length of a multiple packet size, the optical switch considered in this work is equipped with only a single feed-forward fiber delay line per output port. As such, the complexity of the switch can be affordable and the implementing feed-forward optical buffers can be feasible. In addition, we demonstrate how QoS is achievable using a simple feed-forward optical buffering based switching architecture, and evaluate its performance.

Markovian based models were proposed to accurately analyze feedback-buffer based optical switches. For a moderate to large switch sizes, the states of these models explode rendering the study of such switches unachievable. To develop a simple and accurate, and yet overcomes the explosion of states that occurs with Markovian-based models, a Surjective-Mapping based Model (SMM) is developed to evaluate the performance of an optical shared buffer switch. The SMM provides a comprehensive and complete outlook of the performance characterization of the switch including the distribution of the occupancy of the delay lines and the delay encountered in the switch. Results from the model, for switches of different sizes and number of delay lines, are verified using simulation based results. The SMM approach requires significantly fewer computations, $O(N+K)$, compared to Markovian based models. The model enables dimensioning the switch architecture to meet the target performance. The SMM technique is a new approach that can further be utilized to evaluate different slotted communication systems.

Chapter 3

PROBLEM STATEMENT

Driven by the tremendous increase of the number of Internet users and applications, data networks have been growing exponentially in the last few years. To support the current and foreseeable growth of bandwidth demand, all-optical network is a promising solution for deployment in telecommunications backbone networks. In all-optical WDM networks, each fiber can provide multiple communication channels through different non-overlapping wavelengths, where each channel is capable of providing T bps date rate.

Many critical issues are yet to be solved in the context of all-optical networks. From the network performance perspective, one major challenge is how to resolve packet contention that occurs when two or more packets or bursts are forwarded to the same output simultaneously. Another major challenge from QoS perspective is how to differentiate services for different classes of traffic in the optical domain itself. In a traditional electronic switch, contention is generally resolved through random access memory (RAM). In the optical domain, RAM-like buffers are not available.

Several contention resolution schemes, such as wavelength conversion, deflection routing, and optical buffering, have been proposed. Performance- and cost-wise, optical buffering realized by fiber delay lines (FDLs) is deemed to be an efficient contention resolution approach as its is a passive device, as opposed to

wavelength converters that are active device, and it does not contribute to network congestion as deflection routing does.

Design of optical switches as well as all-optical networks based on these switches requires performance models that relate designated performance parameters. Analytical models are needed that relate different parameters, such as the number of FDLs, their lengths, etc. to the performance of the switch. Optimizing the switch performance metrics with a given set of constraints is another important aspect of having a complete analytical model. Some switching configurations are very complex and involve tremendous interdependencies between the functionalities of internal switching elements. As such, an analytical model may be based on many approximations. In this case, simulation is an alternative tool that can be utilized to provide an accurate picture of how the switch behaves. Simulation code provides flexibility in changing the switch configuration and relates different parameters together according to whatever control and management mechanism is chosen. Additionally, simulation provides an efficient way to validate the accuracy of the analytical models. This work involves both analytical and simulation models to evaluate the performance of optical buffering based switches.

The scope of this work is to develop a thorough understanding of the integration of optical buffering, internal switching components, and control and management mechanism. This work also aims at articulating the limitations of implementing optical buffering, and assesses these limitations against the achievable performance gain. Therefore, the goal of this work was to implement a complete framework that provides a comprehensive understanding of different performance metrics of optical buffering schemes when implemented in an isolated optical switch. Performance parameters generated by such models can in turn be used to evaluate multi-hop networks. The work also addresses service differentiation schemes and assesses the related performance and control complexity. The

techniques developed in the process can be used for feasibility and performance evaluation of more complex architectures employing optical buffering scheme in conjunction with other contention resolution schemes.

Chapter 4

ALL-OPTICAL BUFFERING BASED SWITCHES: SIMULATION BASED MODELS

In this chapter, an overview on the simulation techniques used in this work is highlighted. We begin with an introduction on the effectiveness of using simulation. Then, a discrete-event simulation technique is discussed in details. Slotted-time based simulation is then illustrated. The different ways of verifying the simulation is then provided.

4.1 Introduction

Simulation is one of the most powerful, yet easy and convenient techniques to evaluate and characterize the performance of a system that is generally not available via a code of program. Owing to its extreme flexibility on the investigated system, simulation allows varying any parameters of the system simply by modifying the simulator code. Normally, a simulation model is developed to verify the accuracy of an analytical model. Occasionally, it is not feasible to have an analytical model for a very sophisticated system. In this case, a system test-bed may be built in the lab. However, for a complex system, it is very expensive to have such a physical model of the system. Thesefore, the simulation approach is an easy and flexible way to provide an insight into the expected behavior of the real system.

This chapter provides an overview on the simulations conducted of this work. A detailed illustration of the discrete-event simulator developed for the asynchronous based feed-forward optical buffering is presented. The chapter also provides an overview of the slotted based simulator developed for the feedback optical buffer.

4.2 Discrete-event simulator

In this work, for modeling switches augmented with feed-forward optical buffers working on an asynchronous mode, a discrete-event simulator was written in C and then was used to simulate and evaluate the switch with different configurations and parameters. Based on descriptions extracted from [19], this section describes the main components of the simulator and the mean by which it was verified. The simulated model is considered to be a *discrete state/time model* since the buffer is described by some discrete states in a discrete time manner. The simulation results consist of collected average performance parameters corresponding to a steady state of the switch. Appendix B shows the source code, which is written in C, for the simulator discussed in this section.

4.2.1 Main components of the simulator

The simulator is composed of several major components that interact while the program is running to simulate the real network so that statistical metrics can be estimated.

4.2.1.1 Event

An event describes the activity to be carried out at the time specified. It implicitly describes the state of each packet in terms of the location of the packet in the buffer. For example, a NEW ARRIVAL event demonstrates that a new

packet has just arrived and needs to be processed by forwarding it to either port A or B.

The main components of an event are as follows:

- Event type: describe the type of action the related packet is to perform. It can be classified as follows:
  - NEW ARRIVAL: describes a packet that has newly arrived at the buffer. Upon processing this event, TAKE DECISION event will be generated.
  - TAKE DECISION: describes the packet that has already been processed and needs to be routed. As a result, either the desired output port is set BUSY according to the routing algorithm used or the packet gets dropped if the routing is not applicable.
  - SET OUTPUT PORT BUSY: describes the packet that has passed to the output port of a node. As a result, that port is kept busy for the *Transmission Time* of the packet, after which a generated SET OUTPUT PORT FREE event will be processed.
  - SET OUTPUT PORT FREE: describes the packet that has just left the output port. As a result, that port is released and becomes ready to accept a new packet.
- Size of the packet: describes the size of the packet in bit time. In the simulator, each bit in the packet is assumed to last for one bit time, so the size of the packet is defined by its transmission time.
- Output port: describes the port that the packet will be routed to according to the routing decision made.
- Came from port: describes the port the packet came from.

- Arrival time: represents the time the packet arrives at the destination.
- Current time: represents the current time of the simulator.
- Time to be processed: represents the time at which the event will be processed.
- Port state: represents the state of the ports of a buffer and has a state of either *BUSY* or *FREE*.

4.2.2 Event list

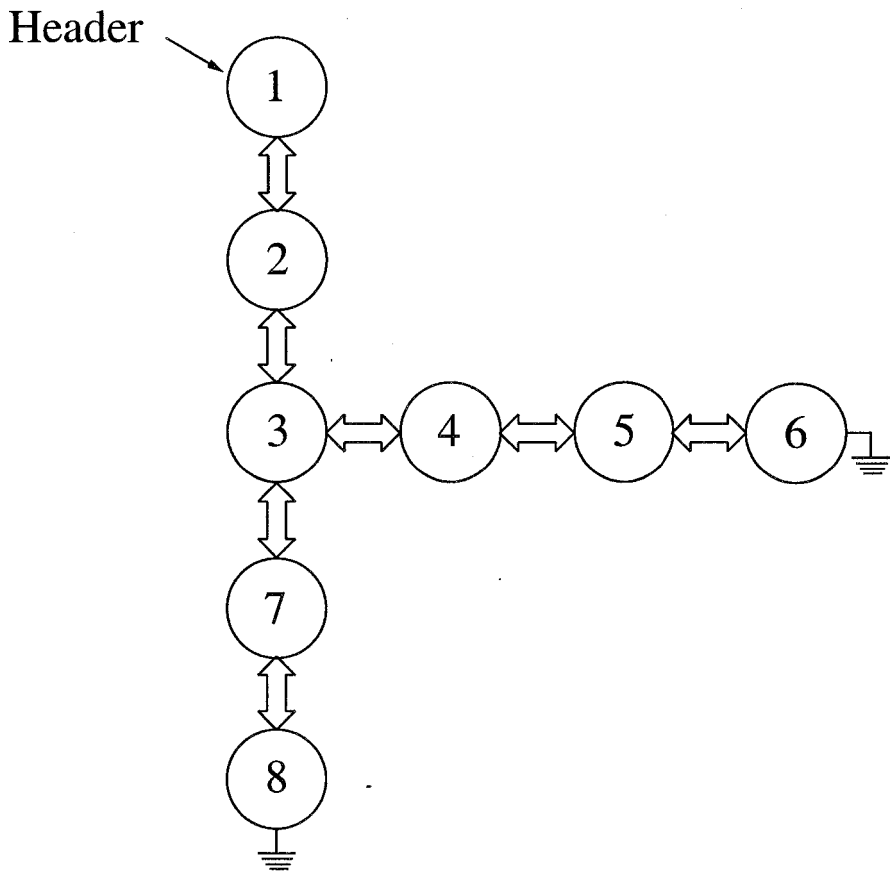

Figure 4.1: Schematic representation of the event linked list

A schematic representation of a typical event list is shown in Figure 4.1. Circles in the Figure represents events linked together to form the event list. The

Picked event	Generated event(s)	Increment value
NEW ARRIVAL	TAKE DECISION	*processing time*
TAKE DECISION	1) SET OUTPUT PORT BUSY	0
	2) SET OUTPUT PORT FREE	packet *transmission time*
	3) NEW ARRIVAL	actual arrival time

Table 4.1: The generated events resulting from the related processed events

next event to be processed is the one that is pointed to by the header pointer. Initially, the list is empty and then each event is inserted into the list and sorted according to the "time to be processed" field in the event. The header pointer points to the top most (number 1 in the Figure) event to be processed first. If two events have the same value of "time to be processed", they will be inserted into the list in a simultaneous manner, such as events number 4, 5, and 6 in the figure, and randomly one of them will be picked and processed.

When an event is picked from the list and gets processed, one or more of the related new events shown in Table 4.1 is generated, with its time to be processed set to the picked event's *time to be processed* incremented by the value given in the Table. Note that the (NEW ARRIVAL) event is generated as a result of picking a TAKE DECISION event when a packet is lost. As an example, let's consider the situation when a packet arrives at the buffer, i.e. the NEW ARRIVAL event has just been processed. A TAKE ECISION event will be generated and inserted into the event list as a result of processing the NEW ARRIVAL event to simulate the time it takes to process the header of the packet and to set up the internal switches according to the decision made. Obviously, the *time to be processed* of the newly generated event, TAKE DECISION, will equal the *time to be processed* of the NEW ARRIVAL event plus the *processing time*. In our simulator, *processing time* is a negligible value. After picking the TAKE DECISION event from the event list and processing it, and if the decision is made to route the packet to an

output port, three events will certainly be generated and inserted into the event list. The first event is SET OUTPUT PORT BUSY, which results in setting that output port *BUSY* so no other packet can be routed to this port until it gets released. This event will be inserted at the top of the event list so it can be processed immediately. The second event is SET OUTPUT PORT FREE whose *time to be processed* equals that of SET OUTPUT PORT BUSY event plus the *Transmission Time* of the packet. Upon picking and processing this event, that port gets released and set to a *FREE* state so that another packet can be routed to this output port.

4.2.3 Random generator

The simulator uses its own random generator function rather than the system built-in generator routine [27]. The cycle life (period) of the random generator used is a very large number and successive values of this generator are independent and uniformly distributed ranging between 0 and 1. This generator is called *Multiplicative Linear-Congruential Generator (LCG)* and is shown in equation 4.1

$$x_n = 7^5 x_{n-1} mod(2^{31} - 1) \tag{4.1}$$

The period of this generator is of a value equal to 2147483646. The inverse transformation method is used to generate the truncated exponential variate that represents the time between the successive packets' arrivals as well as the packet size. The generator is calculated as,

$$x = -\mu \log x_n \tag{4.2}$$

where μ represents the *Mean Interarrival Time or Mean Packet Size.* Each time an interarrival time or a packet is generated, the generator is fed with a new non-correlated non-zero odd-value seed. That is, each time a new variate is generated, the related seed is updated.

4.2.4 Configuration parameters

The basic time unit of the simulator is called *bit time*. It represents the transmission time of a single bit. It is used to express different time parameters such as *Propagation Time Delay*, *Interarrival Time*, *Transmission Time*, and *Processing Time*. The *Mean Transmission Time* for each node is defined by the *Mean Packet Length*; so 1 kbit packet corresponds to 1024 bit times. A truncated exponential distribution is used for the message length. The *Mean Interarrival Time* defines the time between successive packet arrivals that arrive at the buffer. The load is defined as the ratio of the *Mean Transmission Time* to the *Mean Interarrival Time*. The length of the FDL defines its propagation delay. By considering the speed of the light in the fiber to be 2×10^5 km/sec, and assuming the network operating at a rate of 1×10^9 bit/sec, the propagation delay for an FDL length of 1 km is 5000 bit time.

4.2.5 Simulator algorithms

The flow chart shown in Figure 4.2 illustrates the general flow of the simulator. Initially, all random generators, including seeds' generator, packet initialization time generator, and packet size generator are initialized. Additionally, all port variables are set to *FREE* state and all statistical variables are initialized to zero. Seeds are fed into the related generators so that the first packet gets generated and inserted into the event list. Upon building the event list, the actual simulation is started. The first event then gets picked and processed and its *time to be processed* is the starting time of the simulation, so it is used as a reference for calculating the simulation time. Each time a NEW ARRIVAL event is processed, the delay and the loss are recorded. After a fairly large simulation time, the average delay and the loss probability are calculated. New average delay and loss probability are calculated for a similar window of simulation time. If the variation of both the loss

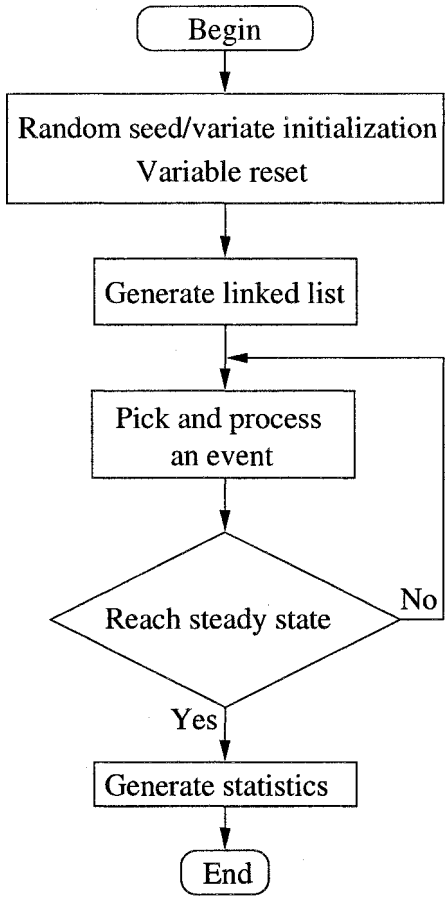

Figure 4.2: The flow chart of the simulator behavior

probability and the delay from the previously calculated values are greater than 1%, this duration is considered to be a transient state where the statistical data is discarded. As such, this scenario gets repeated until both metrics reach a variation of less than or equal to 1%, at which point it is considered to be the starting point of the steady state of the buffer, so it indicates the beginning of the collection of the statistical data in terms of the average delay and the loss probability for a window of simulation time.

4.3 Slotted based simulator

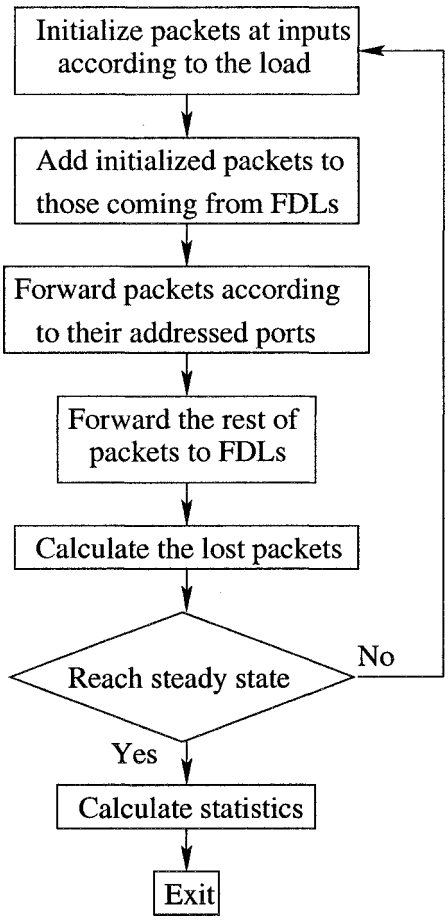

Figure 4.3: The flow chart of the slotted simulator behavior of the feedback optical buffering

For the feedback optical buffering case, a slotted based simulator was built. This type of simulator is easier that the asynchronous based discrete-event simulator built for the feed-forward buffering case. In the slotted based switch, where a packet size is fixed, all output ports are in a *FREE* state at the beginning of each slot and the loss might take place only due to two or more packets being addressed to the same output port. Therefore, a simple code *loop* routine handles this type of simulation. Figure 4.3 illustrates the simple flow the slotted simulator follows

via a simple *loop* routine. Appendix D shows the source code, which is written in Matlab, for the simulator discussed in this section.

4.4 Simulator verification

In this subsection, the techniques that were used to verify the simulators are discussed.

4.4.1 Tracing

For the discrete-event simulator, the following functions were developed for tracing purpose:

- PrintEvent: to print the content of a picked event.
- PrintList: to print the content of the event list.
- PrintPortStatus: to print the status of the ports.
- PrintStatistics: to print the overall statistics of the simulated buffer.

The simulator operates in a discrete manner and each time an event is about to be processed, the simulator is suspended until the "ENTER" key is hit. Each time an event is picked from the event list, it is printed and then the event list is checked to see whether it has been modified accordingly and consistently. The port status is also checked to ensure that it is being modified correctly according to what is expected. Moreover, the supposed-to-be-generated event is monitored by monitoring the event list for the next cycle to ensure that this event has been inserted properly into the list according to the time to be processed parameter. Each time a packet is dropped, an indication is generated. By monitoring the port status, if all ports are BUSY, a drop indication is observed for incoming packets. Finally, buffer statistics are printed and compared with the manually recorded

data to see if total consistency has been achieved. One of the observations made is that the number of arrived packets is equal to the number of processed packets plus dropped packets plus those residing in the linked list.

For the slotted based simulator, at the beginning of each time slot, addresses of the packets coming from the FDLs as well as those arriving at the switch are observed. At the next time slot, their consistency is the observed with packet arrivals from FDLs.

4.4.2 Degeneracy test

The simulator is observed under the following extreme boundaries:

1. Manually, all the ports are set to the BUSY state and thereby, all packet arrivals are dropped.

2. At a low and high load point, the loss is observed to check its consistency with these load points.

4.4.3 Similar works

Considering the case of the feed-forward optical buffering, the studied routing algorithms, to the best of our knowledge, are new and hence no previous work has been used as a benchmark to the achieved results. On the other hand, for the feedback optical buffering, results were compared to those obtained form a similar buffer presented in [5] and proved to be similar.

Chapter 5

FEED-FORWARD-FDLS BASED OPTICAL SWITCH

5.1 Introduction

In this work, a comprehensive model is developed to predict the performance of an all-optical network switch considering exponentially distributed packet or burst lengths. To keep the complexity of the switch affordable and the implementing feed-forward optical buffers feasible, the optical switch is equipped with only a single feed-forward fiber delay line per output port. Since the algorithm for forwarding the packets to the different feed-forward FDLs of the optical buffer has a great impact on the performance of the optical switch, the models presented here evaluate the switch with different forwarding algorithms.

5.2 Optical switch with output buffers

Figure 5.1 shows the architecture of an optical switch augmented with output feed-forward FDLs. The FDL indicated by the shaded area will be referred to as an output buffer module. Figure 5.2 shows a simplified logical view of the architecture with a nodal degree of $N \times N$. The output buffer module consists of two ports; port A represents the port with no FDL and hence its propagation delay is negligible. Port B denotes the beginning of the FDL whose propagation delay is D. Note that port F represents the end of the FDL and is virtually the same as port A.

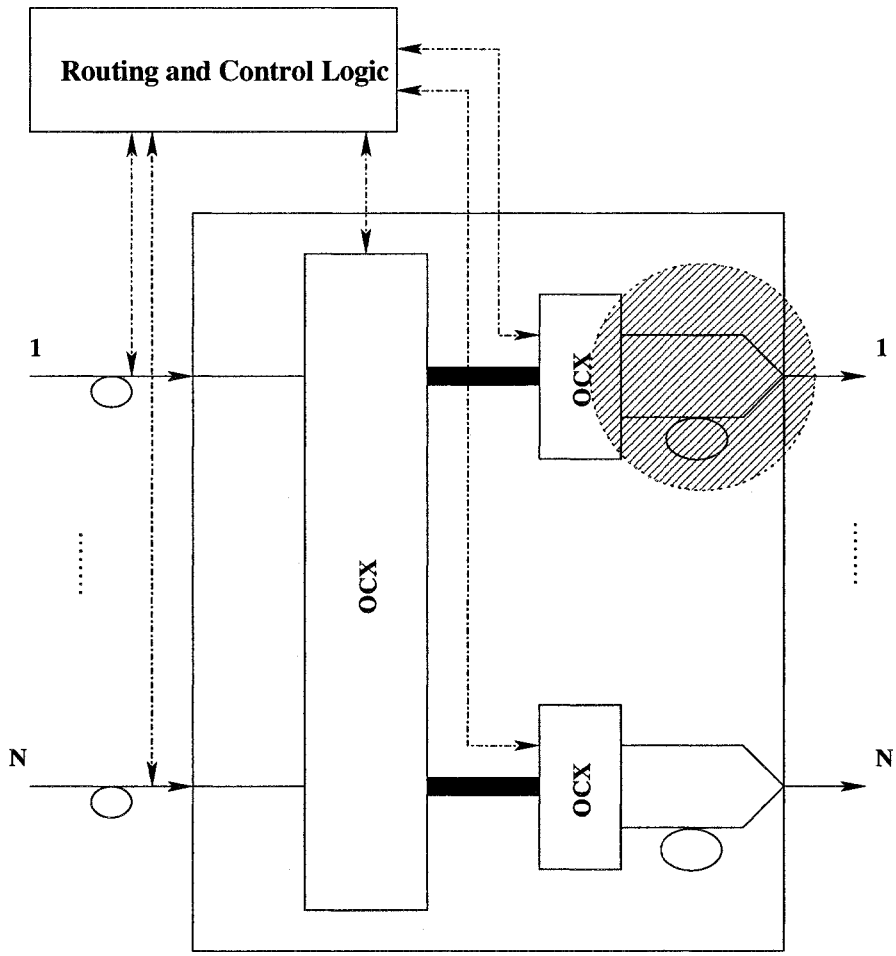

Figure 5.1: Optical packet switch with an output feed-forward FDL

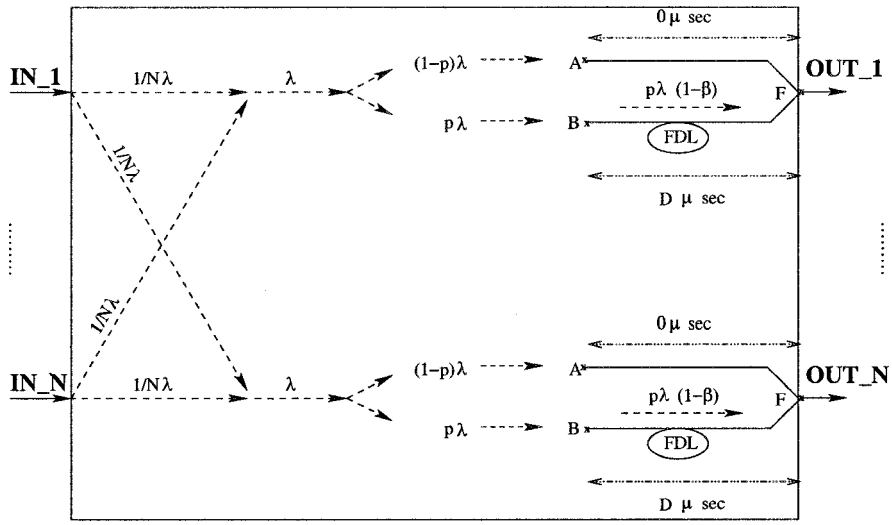

Figure 5.2: A logical representation depicting arrival rates from different inputs at output FDLs

```
if port A is FREE and no burst exists in the FDL
    forward the incoming burst to port A
else if port B is FREE
        forward the incoming burst to port B
    else drop the burst
```

Figure 5.3: The simple forwarding algorithm (SFA)

Figure 5.3 shows the forwarding scheduler algorithm, *the Simple Forwarding Algorithm* (SFA), that determines the port to which an incoming packet is forwarded. In Section 5.5, an enhanced forwarding algorithm (EFA) will be presented that requires more control overhead while providing a better performance. The SFA requires a straight forwarding control mechanism in which an incoming packet is forwarded to port A as long as the FDL contains no packet; otherwise, the packet is forwarded to port B. If a packet is forwarded to port A, it will leave port F before any packet that is forwarded to port B reaches port F since the propagation delay of the FDL is considered to be large enough to accommodate the transmission time of the packet that was forwarded to port A. In the analytical model presented in this work, where the packet length is exponentially distributed, this potential collision is ignored. The length of the FDL is selected, as discussed in Section 5.3.4, such that the probability of this collision is negligible. Appendix A shows the source code, which is written in Matlab, for implementing the analytical model discussed in this chapter.

5.3 Analytical model

The model presented in this section characterizes an $N \times N$ optical switch in terms of the blocking probability. Each output link is augmented with a single feed-forward optical buffer to resolve the contention that may take place. Due to the symmetry of the optical switch, the analytical model only focuses on one output buffer and the rest of the output ports can be modeled similarly. A Poisson packet arrival process is considered with an arrival rate of λ per port, and the packet destinations are assumed to be uniformly distributed among all the output modules of the switch; i.e., each packet is addressed to every output port with a probability of $1/N$. Since the aggregated traffic at an output port is represented by randomly multiplexed Poisson arrival, this aggregated traffic is also a Poisson

arrival with a rate of λ. Therefore, after multiplexing the data from all input links to all output modules, each module receives data at a rate of λ. The length of a packet is exponentially distributed, with a mean transmission time of $1/\mu$. The load, ρ, is considered to be the ratio of the arrival rate to the service rate of the packets, i.e. $\rho = \lambda/\mu$.

Lets consider the *SFA* shown in Figure 5.3. Let $(1-p)$ be the probability of forwarding an incoming data packet to port A, and hence, as shown in Figure 5.2, the data rate of packets forwarded to port A is $(1-p)\lambda$. The remaining stream, of rate $p\lambda$, will attempt port B, and according to the *SFA* some of this data will get blocked at port B. If a packet is forwarded to port A, it encounters no delay, whereas it encounters a delay of D, the propagation delay of the FDL, if it is forwarded to port B. The forwarding of a packet to a port is determined by many different random parameters, and therefore we approximate the streams forwarded to ports A and B as independent Poisson streams with rates $(1-p)\lambda$ and $p\lambda$ respectively for the purpose of evaluating the interaction between the two streams at port F. Thus, we model the behavior of ports A and B as independent M/M/1/1 queues for this purpose. The approximation made for the independence between the stream forwarded to the FDL and that is directly forwarded to the output fiber can be reasonable. The stream that is forwarded to the FDL is partially consists of the overflow from the stream that is directly forwarded to the output fiber. However, that stream is a factor of both packet length and inter-arrival time of the incoming stream. To clarify, though port A is free, an incoming packet can be forwarded to port B since its length, which is independent of the arrival rate, is longer that the residual time of the delayed packet. This approximation allows us to develop this analytical model. The simulation results presented in Section 5.4 evaluate the accuracy of the model, and indicate that it is a reasonable approximation.

5.3.1 Packet forwarding and port busy probabilities

Let α be the probability that port A is busy and β be the probability that port B is busy. From the M/M/1/1 queuing model, let $(1-\beta)$ and β represent the probability of port B is being in a free and busy states, respectively.

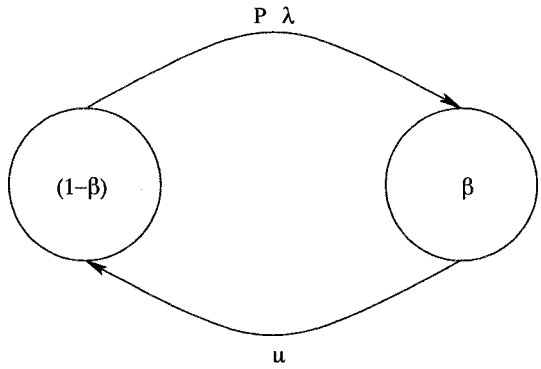

Figure 5.4: Port B states

As shown in Figure 5.4, β can be simply solved by solving the two-state Markov chain. Therefore, β is evaluated as

$$\begin{aligned} \beta &= \frac{p\lambda}{p\lambda+\mu} \\ &= \frac{p\rho}{p\rho+1} \end{aligned} \tag{5.1}$$

There are two packet streams that affect the state of port F. One of these streams is the *direct stream* that is forwarded to port A without traversing the FDL. This stream affects the state of port A, and hence port F, according to the queuing model. Let α' be the probability that port F is busy due to this stream. From Figure 5.5, α' is evaluated as,

$$\begin{aligned} \alpha' &= \frac{(1-p)\lambda}{(1-p)\lambda+\mu} \\ &= \frac{(1-p)\rho}{(1-p)\rho+1} \end{aligned} \tag{5.2}$$

The other stream that affects the state of port F is a *delayed stream* that corresponds to the data packets coming from port B. Let α'' be the probability of

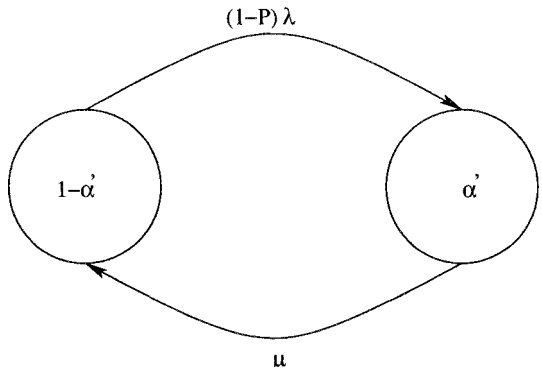

Figure 5.5: Port F states due to *direct stream*

port F being busy as a result the *delayed stream*. In fact, α'' is the load due to the *delayed stream* at port F and hence $\alpha'' = \beta$. According to the SFA shown in Figure 5.3, there are two events that are mutually exclusive:

- Port F being busy due to the *direct stream*, and
- Port F being busy due to the *delayed stream*.

Therefore, the probability that port F is busy, either by the *direct stream* or by the *delayed stream*, is given by α, where

$$\begin{aligned} \alpha &= \alpha' + \alpha'' \\ &= \frac{(1-p)\rho}{(1-p)\rho + 1} + \frac{p\rho}{p\rho + 1} \end{aligned} \tag{5.3}$$

According to the SFA shown in Figure 5.3, a packet is forwarded to port B if either the FDL has at least one packet or port F is busy. The probability of forwarding an incoming packet to port B is p. Let Γ be the event that at least one packet is present in the FDL, and γ be its probability. Then,

$$p = \gamma + \alpha - \eta \tag{5.4}$$

where η is the probability that port F is busy while there are one or more packets present in the FDL, i.e.,

$$\eta = P[port\ F\ is\ busy\ \cap\ \Gamma) \tag{5.5}$$

5.3.2 The time (X) during which the FDL is free of packets

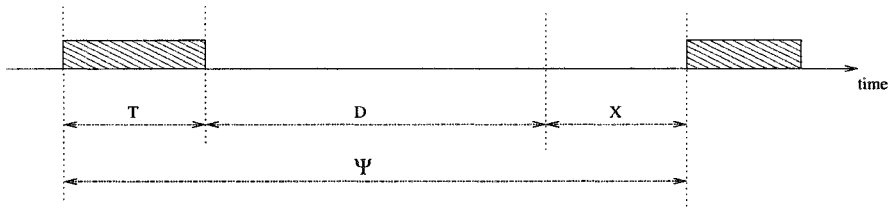

Figure 5.6: Timing diagram of packet arrival at the FDL when the inter-arrival time is greater than D

To evaluate the probability p, it is necessary to first evaluate γ, which is the probability that at least one packet exists in the FDL,

$$\gamma = 1 - \omega \tag{5.6}$$

where ω represents the probability that the FDL contains no packet, i.e. it is empty. To evaluate ω, the case of interest here is when the inter-arrival time of two packets entering the FDL is greater than the transmission time of the first packet plus the delay D. When the inter-arrival time of two packets is less than the transmission time of the packet plus D, it is not possible for the FDL to be empty, i.e. it would contain at least a part of a packet. Figure 5.6 shows the relationship between different parameters that constitute the inter-arrival time between a pair of packets entering the FDL, where the inter-arrival time is represented by the random variable Ψ ($\Psi > D$). The random variable T represents the transmission time of a packet, which is assumed to be exponentially distributed with a rate of μ, and hence,

$$f_T(t) = \mu e^{-\mu t} \tag{5.7}$$

The random variable X represents the FDL idle time between the packet arrival. For the case under consideration, it corresponds to the time during which the FDL

is empty. Let the rate of packet arrival entering the FDL be $\widehat{\lambda}$ (i.e. $\widehat{\lambda} = p\lambda$). This rate represents the rate of exponentially distributed random variable Ψ. Thus,

$$f_{\Psi}(\psi) = \widehat{\lambda} e^{-\widehat{\lambda}\psi} \tag{5.8}$$

where ψ is the range of Ψ. Let $\Theta = T + D$, then

$$\begin{aligned} f_{\Theta}(\theta) &= f_T(\theta - D) \\ &= \mu e^{-\mu(\theta - D)} \qquad \theta \in [D, \infty) \end{aligned} \tag{5.9}$$

$$\begin{aligned} P[\Psi < \Theta] &= \int_D^{\infty} \int_0^{\theta} f_{\Theta}(\theta) f_{\Psi}(\psi) \quad d_{\psi}\, d_{\theta} \\ &= \int_D^{\infty} \int_0^{\theta} \mu e^{-\mu(\theta - D)} \widehat{\lambda} e^{-\widehat{\lambda}\psi} \quad d_{\psi}\, d_{\theta} \\ &= 1 - \frac{1}{1 + p\rho} e^{-\widehat{\lambda} D} \end{aligned} \tag{5.10}$$

Therefore,

$$P[\Psi > \Theta] = \frac{1}{1 + p\rho} e^{-\widehat{\lambda} D} \tag{5.11}$$

The values associated with the random variable X satisfies,

$$X = \begin{cases} 0 & if\ \Psi \leq \Theta \\ > 0 & if\ \Psi > \Theta \end{cases} \tag{5.12}$$

The PDF for X can be written as

$$f_X(x) = g_X(x) + h_X(x) \tag{5.13}$$

where $g_X(x)$ and $h_X(x)$ are contributions corresponding to the cases where $\Psi < \Theta$ and $\Psi > \Theta$ respectively. Consider the case when $\Psi < \Theta$. In this case, $X = 0$ as, from Figure 5.6, there will be one or more packets inside the FDL. Therefore,

$$\begin{aligned} g_X(x) &= \delta(x) P[\Psi < \Theta] \\ &= \delta(x)(1 - \frac{1}{1 + p\rho} e^{-\widehat{\lambda} D}) \end{aligned} \tag{5.14}$$

where $\delta(x)$ is the delta function. Next, consider the case when $\Psi > \Theta$. Here, $X = \Psi - \Theta$. The conditional CDF of X given that $\Psi > \Theta$ is given by

$$\begin{aligned} H_X(x \mid_{\Psi > \Theta}) &= P[X \leq x \mid \Psi > \Theta] \\ &= P[\Psi - \Theta \leq x \mid \Psi > \Theta] \\ &= \frac{P[\Psi \leq x + \Theta, \Psi > \Theta]}{P[\Psi > \Theta} \\ &= \frac{P[\Theta \leq \Psi \leq x + \Theta]}{P[\Psi > \Theta]} \end{aligned} \tag{5.15}$$

Now,

$$\begin{aligned} P[\Theta \leq \Psi \leq x + \Theta] &= \int_D^\infty \int_\theta^{x+\theta} f_\Theta(\theta) f_\Psi(\psi) d_\psi d_\theta \\ &= \int_D^\infty \int_\theta^{x+\theta} \mu e^{-\mu(\theta - D)} \widehat{\lambda} e^{-\widehat{\lambda}\psi} d_\psi d_\theta \\ &= (1 - e^{-x\widehat{\lambda}}) \frac{1}{1 + p\rho} e^{-\widehat{\lambda} D} \end{aligned} \tag{5.16}$$

Therefore,

$$\begin{aligned} H_X(x \mid_{\Psi > \Theta}) &= \frac{(1 - e^{-x\widehat{\lambda}}) \frac{1}{1+p\rho} e^{-\widehat{\lambda} D}}{\frac{1}{1+p\rho} e^{-\widehat{\lambda} D}} \\ &= 1 - e^{-\widehat{\lambda} x} \end{aligned} \tag{5.17}$$

which implies that, when $\Psi > \Theta$, $X \sim EXP(\widehat{\lambda})$. Consequently,

$$\begin{aligned} h_X(x) &= \widehat{\lambda} e^{-\widehat{\lambda} x} P[\Psi > \Theta] \\ &= \widehat{\lambda} e^{-\widehat{\lambda} x} \frac{1}{1 + p\rho} e^{-\widehat{\lambda} D} \end{aligned} \tag{5.18}$$

Using equation (5.13), the PDF of X in the whole domain; i.e., when $\Psi < \Theta$ and $\Psi > \Theta$, is given by,

$$\begin{aligned} f_X(x) &= \delta(x)(1 - \frac{1}{1 + p\rho} e^{-\widehat{\lambda} D}) + \\ &\quad \widehat{\lambda} e^{-\widehat{\lambda} x} \frac{1}{1 + p\rho} e^{-\widehat{\lambda} D} \end{aligned} \tag{5.19}$$

From $f_X(x)$, the expected value of X is found to be

$$\begin{aligned} E(X) &= \int_0^\infty x f_X(x) d_x \\ &= \frac{1}{1 + p\rho} \frac{e^{-\widehat{\lambda} D}}{\widehat{\lambda}} \end{aligned} \tag{5.20}$$

5.3.3 The forwarding probabilities to the ports of output buffer

From Figure 5.6, assuming the system to be stable [6], the probability that no packet exists in the FDL, ω, can be approximated by

$$\begin{aligned}\omega &= lim_{n\to\infty}\frac{\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} \psi_i} \\ &= lim_{n\to\infty}\frac{(\sum_{i=1}^{n} x_i)/n}{(\sum_{i=1}^{n} \psi_i)/n} \\ &= \frac{E(X)}{E(\Psi)} \\ &= \frac{1}{1+p\rho}e^{-\widehat{\lambda}D} \end{aligned} \tag{5.21}$$

and

$$\gamma = 1 - \frac{1}{1+p\rho}e^{-\widehat{\lambda}D} \tag{5.22}$$

Here, x_i and ψ_i correspond to the i^{th} idle period and the corresponding inter-arrival time respectively. By definition,

$$\eta = P[\Gamma|F \textit{ is busy}]\alpha \tag{5.23}$$

The probability $P[\Gamma|F \textit{ is busy}]$ defines the probability of Γ given that port F is busy. When port F is known to be busy, all the packet stream will certainly be forwarded to port B, i.e., $p = 1$. In this situation, γ is derived similarly to the previous case, but with $p = 1$. Therefore,

$$P[\Gamma|F \textit{ is busy}] = 1 - \frac{1}{1+\rho}e^{-\lambda D} \tag{5.24}$$

Therefore,

$$\begin{aligned}\eta &= (1 - \tfrac{1}{1+\rho}e^{-\lambda D}) \\ &\quad (\tfrac{p\rho}{p\rho+1} + \tfrac{(1-p)\rho}{(1-p)\rho+1}) \end{aligned} \tag{5.25}$$

Now, from Equations (5.3), (5.22), and (5.24), by the mean of iteration, p can be evaluated for different values of λ, μ, and D.

In the SFA, an incoming packet will be blocked only when port B is busy. Therefore, the probability of blocking is given by

$$P_B = \beta \tag{5.26}$$

5.3.4 Determining the propagation delay of the FDL

In this section we characterize the propagation delay of the FDL, D. If two packets arrive simultaneously and both are addressed to the same output link, then one of them will be forwarded to port A, which is called *direct packet*, while the other packet will be forwarded to port B, which is called *delayed packet*. Losses at port F will take place if the *delayed packet* reaches port F while the *direct packet* is still leaving that port. To avoid this situation, the propagation delay of the FDL, D, should be large enough so that the *direct packet* leaves port F before the *delayed packet* reaches this port. Since the transmission time of the *direct packet* is assumed to be exponentially distributed, D is chosen such that the probability that the transmission time of the *direct packet* is greater that D is negligible. In other words, given that the length of the *direct packet* is T,

$$P[T > D] = \epsilon \tag{5.27}$$

where ϵ is very small value. Therefore,

$$D = \frac{\ln(\epsilon)}{\mu} \tag{5.28}$$

For example, if ϵ is chosen to be 0.05, then $D \approx \frac{3}{\mu}$. This relation can be used to determine the approximate length of the FDL. Note that, given that the length of a packet is exponentially distributed, there will be some minor drop due to the described conflict at port F, even after setting D according to (5.28). Since this drop is negligible, we did not take it into account in either the analytical or simulation models. In practice, the lengths of packets are upper-bounded, so D may be selected such that there will be no drop at port F due to collision between both the *direct packet* and the *delayed packet*.

5.4 Performance of the Simple Forwarding Algorithm

To verify the analytical results for the SFA, as discussed in Chapter 3, a discrete-event simulator was developed using C language to evaluate the performance in terms of the blocking probability at each of the output buffers of the $N \times N$ optical switch shown in Figure 5.1. The load, ρ, is considered to be the ratio of the arrival rate to the service rate of the packets, i.e. λ/μ. A Poisson packet generator was used in the simulation for generating packets to the output buffer module. The inter-arrival times and the packet transmission times of the packets were generated independently.

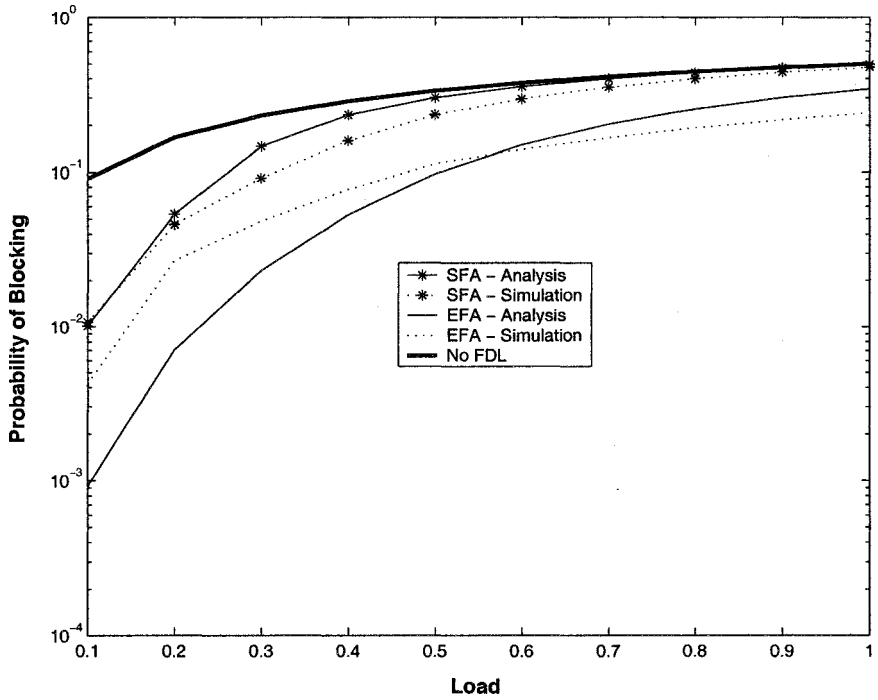

Figure 5.7: The blocking probabilities for SFA, EFA, and no FDL cases for an $N \times N$ optical switch

The results shown in Figure 5.7 corresponding to the simulation and the analytical value of blocking probabilities for an $N \times N$ optical switch of the SFA forwarding algorithm. The mean transmission time of a packet is assumed to be

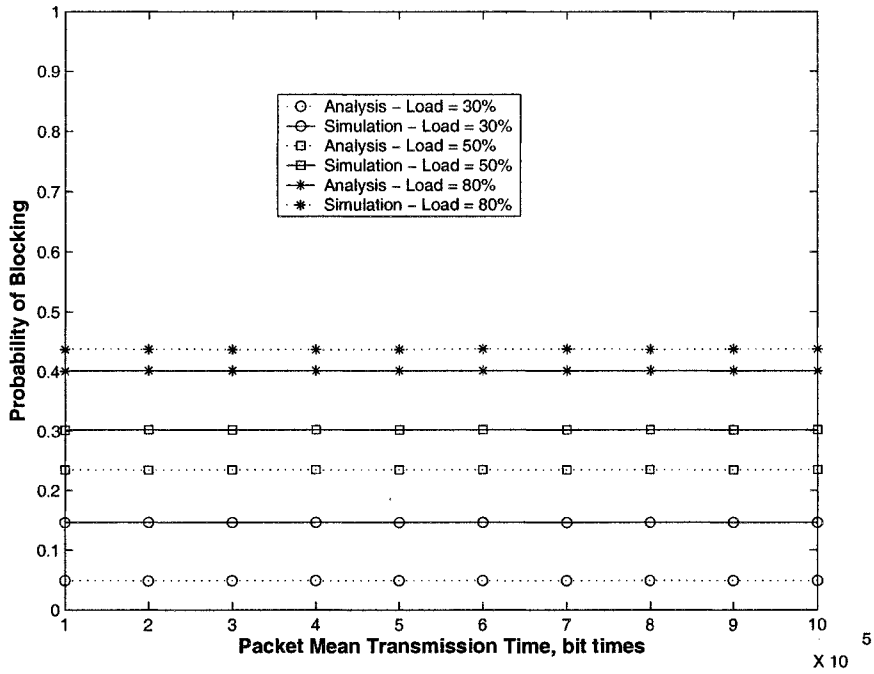

Figure 5.8: The blocking probability of the SFA for different values of μ at a 30%, 50%, and 80% load

100×10^3 *bit times*, where a *bit times* is considered to be the transmission time of a single bit. There is a close match between the results from the analytical model and the simulation model. The difference shown between the two results is partly due to the approximation used in developing the model.

Note that Figure 5.7 also shows the probability of blocking that corresponds to an output module with no fiber delay line derived by an M/M/1/1 queuing model. Comparing this probability of blocking with that for the SFA, the latter represents a significantly lower probability of blocking at lower loads. The difference between the two curves becomes marginal at higher loads due to the fact that, at higher loads, the SFA forwards almost all incoming packets to port B as the FDL contains packets most of the time, and hence the FDL has negligible impact. Note that Figure 5.7 also shows results corresponding to EFA, which will be discussed in the next section.

As explained in Section 5.3.4, the FDL propagation delay has to be selected relative to the mean transmission time of packets. As a result, despite the mean transmission time of the packet, the simulation and analytical models show that, for a given load, the optical switch performs similarly in terms of the blocking probability as long as the length of the FDL satisfies Equation 5.28. For example, Figure 5.8 shows a constant blocking probability for an $N \times N$ optical switch at an 80% load for different values of μ. We conducted further analysis and simulations of

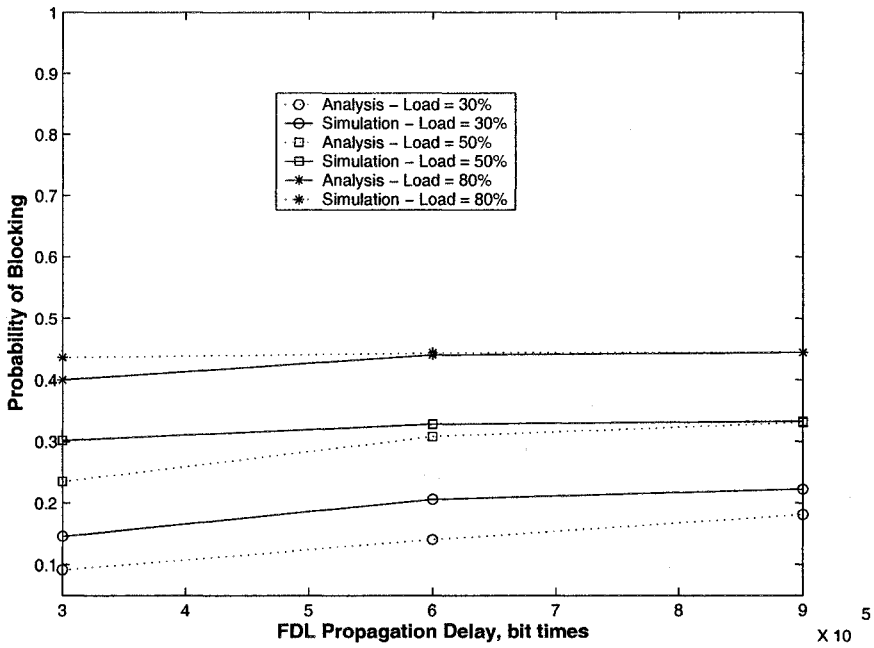

Figure 5.9: The blocking probability of the SFA for different values of D at a 30%, 50%, and 80% load

an $N \times N$ optical switch for different values of FDL propagation delays considering a mean packet transmission time of 100×10^3 *bit times*. As shown in Figure 5.9, the results confirm a close match between the results from the analytical model and those from the simulation model.

5.5 Enhanced forwarding algorithm (EFA)

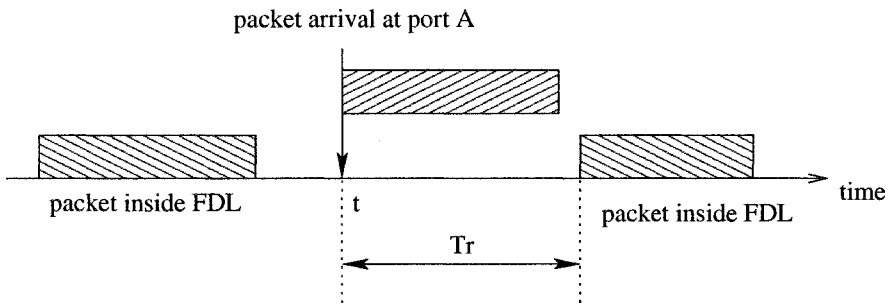

Figure 5.10: The residual time of a *delayed packet* inside the FDL

In this section, we present an *enhanced forwarding algorithm* (EFA) to achieve better performance. The idea behind the SFA is to provide a simple algorithm and implementation of the forwarding scheduler. As long as at least one packet exists inside the FDL, no packet will be forwarded to port A [20]. On the other hand, the EFA achieves better performance by utilizing port F more effectively. The time it takes for the *delayed packet* nearest to port F to reach this port is called the *residual time* (T_r). This is illustrated in Figure 5.10, where a new packet arrives at port A at time t, but the packet in the FDL will arrive at port F only at time $t + T_r$. If an incoming packet arrives at the output module while port A is $FREE$ and the packet transmission (T) time is less than the T_r, allowing it to be inserted into T_r, then this packet becomes a *direct packet* where it leaves the switch through port A. As such the EFA utilizes port F more effectively. The EFA is described in Figure 5.11. Note that this version of forwarding algorithm requires keeping track of T_r to decide whether an incoming packet can be inserted during this time. Therefore, the EFA requires a more complex output buffer module, but provides better performance in terms of the probability of blocking and the delay, as will be shown shortly. The analysis conducted in Section 5.3 holds in the case of the EFA except for the definition of the probability p as well as the probability

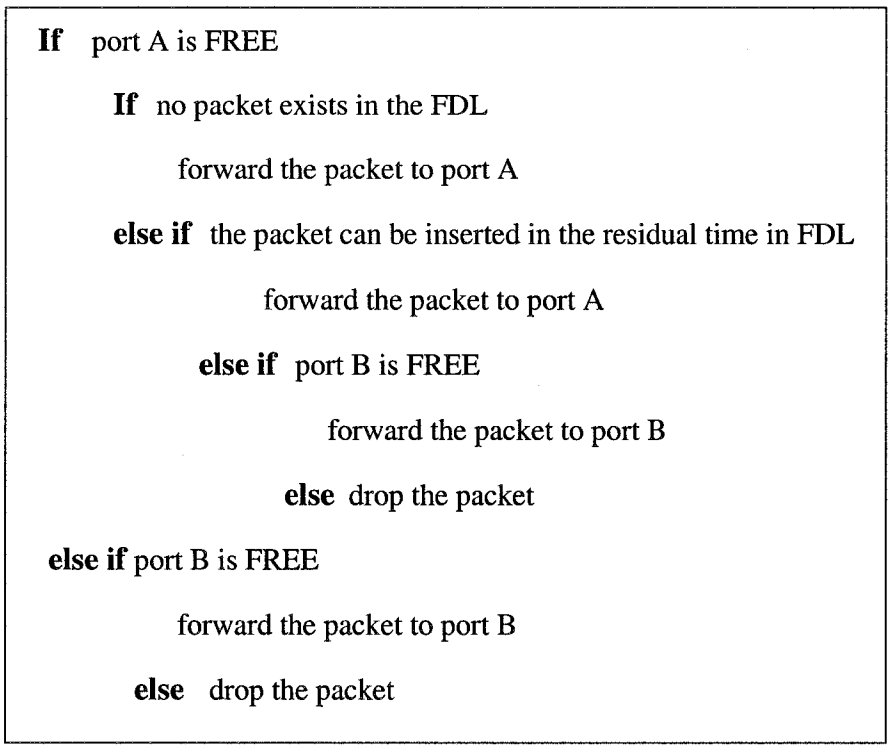

Figure 5.11: The enhanced forwarding algorithm (EFA)

of blocking, P_B. Since EFA suggests that an incoming packet is forwarded to port B whenever:

- Port A is $BUSY$ or
- Port A is $FREE$ but the packet does not fit into the residual time in the FDL

Therefore, the probability p for the EFA algorithm shown in Figure 5.11 is given by,

$$p = \alpha + (1-\alpha)\gamma P[T > T_r] \tag{5.29}$$

where $P[T > T_r]$ represents the probability that the *direct packet* arriving at port A cannot be inserted during T_r. Assuming the stream entering FDL follows a Poisson process,

$$\begin{aligned} P[T > T_r] &= 1 - \int_0^\infty \int_0^t f_\Psi(\psi) f_T(t) d_\psi d_\mu \\ &= 1 - \int_0^\infty \int_0^t \widehat{\lambda} e^{-\widehat{\lambda}\psi} \mu e^{-\mu t} d_\psi d_\mu \\ &= \frac{p\rho}{1+p\rho} \end{aligned} \tag{5.30}$$

Now, using Equations (5.3), (5.22), (5.24), and (5.30), Equation (5.29) can be solved for different values of λ, μ, and D. Now, P_B for EFA is given by,

$$P_B = \alpha\beta + (1-\alpha)\beta P[T > T_r] \tag{5.31}$$

Figure 5.7 shows both analytical and simulation results comparing SFA and EFA for the same switching configuration. For example, at 50% load, the blocking probability of the EFA is 67% lower than that of the SFA. The delay encountered by a packet is proportional to the probability of forwarding it to the FDL, as only those forwarded to FDL undergo a delay, D. Therefore, the mean delay of packets is

$$Delay = p(1-\beta)D \tag{5.32}$$

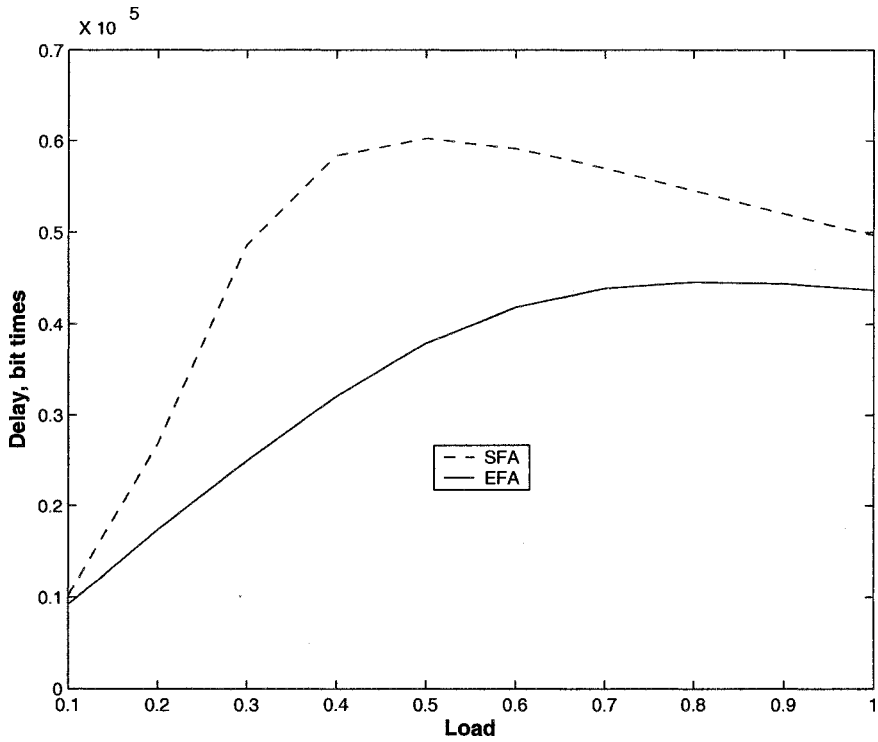

Figure 5.12: The delays for SFA and EFA algorithms

Figure 5.12 shows the delay a packet may undergo for both SFA and EFA. The figure shows the enhancement achievable using the EFA over the SFA. For example, at a load of 50%, when the SFA is utilized, the delay a packet may encounter is 59% higher than that which may be encountered when the EFA is employed.

5.6 Supporting QoS

A QoS mechanism is introduced here that allows service differentiation between different streams arriving at each node. The architecture presented here

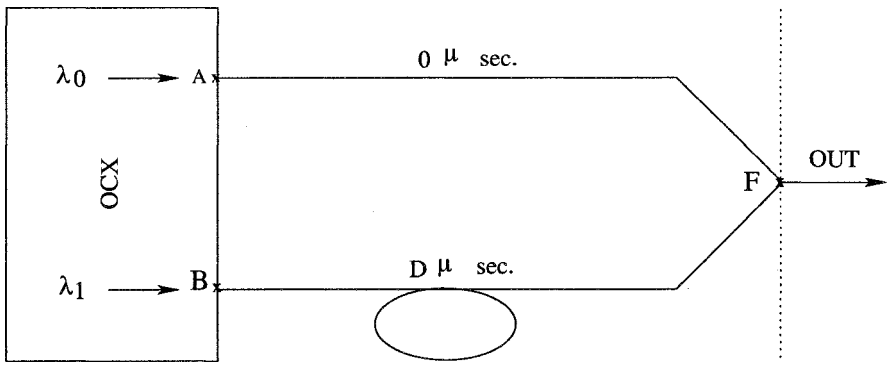

Figure 5.13: An output module of an optical switch showing the flow of the data streams belonging to two classes

provides two classes of services; namely class 0 (C_0) for lower priority data packets, and class 1 (C_1) for higher priority data packets. The algorithm can be generalized to support n different priority classes by considering n FDLs with lengths of $D, 2D, , nD$ respectively. In this case, a packet with a lower priority class is allowed to leave the switch through a port only during the time that port is $FREE$ of packets given that there is sufficient time to transmit that packet before higher priority packets in the longer FDLs arrive at port F. Packets that belong to C_0 and C_1 are assumed to have arrival rates of λ_0 and λ_1 respectively and mean transmission times of μ_0 and μ_1 respectively. Let the offered load corresponding to the C_i be ρ_i, where $\rho_i = \lambda_i/\mu_i$. The proposed forwarding algorithm is implemented

such that C_1 packets are forwarded to port B, where they encounter a delay of D due to traversing the FDL, as shown in Figure 5.13. Let the time it takes for the C_1 packet nearest to port F to reach this port be called the *residual time* (T_r). As shown in Figure 5.14, the C_0 packet arrives at port A at time t, but the C_1 packet in the FDL will arrive at port F only at time $t + T_r$. C_0 packets are forwarded to port A, where no delay is encountered, as long as they can be inserted into T_r. The FDL enables packets of the lower priority class to leave the output port during the residual times of packets belonging to C_1 types of service. For C_1

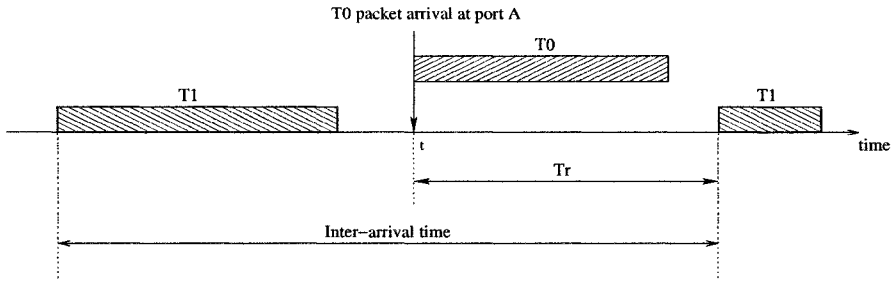

Figure 5.14: The insertion of a C_0 packet into T_r of the arrival of C_1 packets

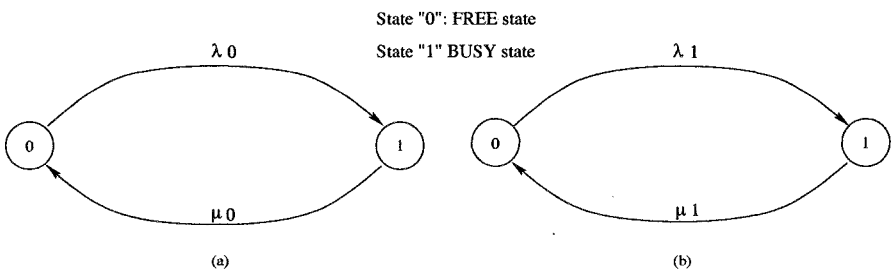

Figure 5.15: The transition probabilities for (a) port A being busy by a C_0 stream and for (b) port B being busy by a C_1 stream

packets, the only contention that may cause blocking is the *inter-class* contention among packets of the same class, i.e. between arrival at port B of C_1 packets from different input ports. Ports A and B being busy due to C_0 and C_1 streams, respectively, can be modeled as shown in Figure 5.15 (b), where state "0" represents the *FREE* state and state "1" represents the *BUSY* state. Considering the fact

that $P[0] + P[1] = 1$, where $P[i]$ is the probability of being at state i, and solving the two-state transition probabilities, the probability of blocking for a C_1 stream, $P_B[C_1]$, is the probability of port B being $BUSY$ due to a packet belonging to C_1 stream, $P[F\ is\ BUSY\ by\ C_1]$, and is given by

$$P_B[C_1] = \frac{\rho_1}{\rho_1 + 1} \tag{5.33}$$

The flow chart shown in Figure 5.16 shows three different cases in which a C_0 packet is blocked.

Case 1 represents the situation when a C_0 packet arrives at port F while the port is $BUSY$ by a C_1 packet. The probability of port F being $BUSY$ due to a C_1 packet is simply the load due to C_1 stream at port F, which equals $P_B[C_1]$.

Case 2 represents the *inter-class* contention among the C_0 packets at port A. The behavior of port B due to the C_0 stream can be modeled as shown in Figure 5.15 (a), and hence the probability of port A being $BUSY$ due to this contention is given by,

$$P[A\ is\ BUSY\ by\ c_0] = \frac{\rho_0}{\rho_0 + 1} \tag{5.34}$$

Case 3 represents the situation when a C_0 packet arrives at port A while port A as well as F are $FREE$ of packets of both classes. In the case that the C_0 packet's transmission time (T_0) is greater than T_r, the C_0 packet will be blocked; otherwise, it leaves the switch through port A. T_0 is assumed to be exponentially distributed with a rate of μ_0. The random variable T_r is exponentially distributed with a rate of λ_1 since it represents the time for the next arrival of a C_1 packet to reach port F. Therefore,

$$f_{T_r}(t) = \lambda_1\ e^{-\lambda_1 t} \tag{5.35}$$

The random variables T_0 and T_r are independent random variables, and their joint probability distribution is obtained by

$$f_{T_r,T_0}(t, t_0) = \lambda_1 \mu_0\ e^{-\lambda_1 t}\ e^{-\mu_0 t_0} \tag{5.36}$$

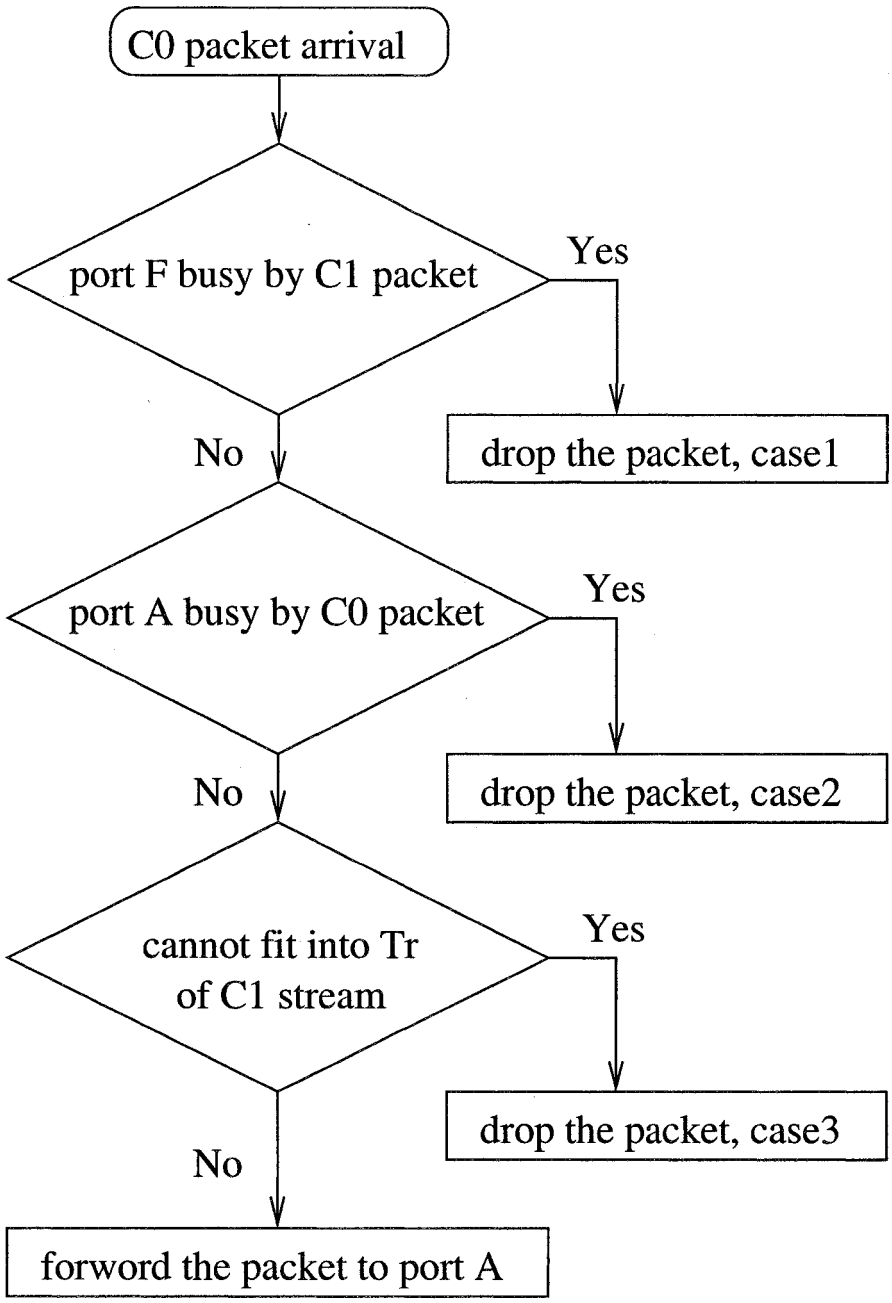

Figure 5.16: A flow chart presenting the cases in which a C_0 stream is blocked

where $\mu_0\ e^{-\mu_0 t_0}$ is the PDF of the random variable T_0. The probability $P[T_0 > T_r]$ is obtained as

$$\begin{aligned} P[T_0 > T_r] &= \int\int_{T_0 > T_r} f_{T_r,T_0}(t, t_0)\ d_t\ d_{t_0} \\ &\vdots \\ &= \frac{\rho_1}{\rho_1 + \frac{\mu_0}{\mu_1}} \end{aligned} \tag{5.37}$$

Therefore, in **case 3**, the probability of blocking of C_0 packets, $P_B[C_0]$, is a result the transmission time of a C_0 packet being larger than T_r. Considering all three cases,

$$\begin{aligned} P_B[C_0] &= P[F\ is\ BUSY\ by\ C_1] + P[A\ is\ BUSY\ by\ C_0] \\ &\quad + \Big[1 - (P[F\ is\ BUSY\ by\ C_1] + \\ &\quad (P[A\ is\ BUSY\ by\ C_0])\Big] P_B[T_0 > T_r] \\ &= \frac{\rho_1}{\rho_1 + 1} + \frac{\rho_0}{\rho_0 + 1} + \Big[1 - (\frac{\rho_1}{\rho_1 + 1} + \\ &\quad \frac{\rho_0}{\rho_0 + 1})\Big] \frac{\rho_1}{\rho_1 + \frac{\mu_0}{\mu_1}} \end{aligned} \tag{5.38}$$

Note that the probability of the blocking of a C_0 stream is a function of the offered loads of both the C_0 and C_1 streams. Figure 5.17 shows the blocking probabilities for both the classes as a function of different loads. The stream of class C_1 is assumed to have a mean packet transmission time of 10000 *bit time* units, while the C_0 stream is assumed to have mean packet transmission times of 1000, 10000, 50000, and 100000 *bit time* units. Note that as the mean packet transmission time of a C_0 stream decreases, the probability of the blocking of this stream decreases since the packets belonging to this class have a better chance to fit T_rs of C_1 arrival. At higher loads, the probability of the blocking of the C_0 stream is twice that of the C_1 stream. Even though the C_1 stream experiences lower loss probability, it also encounters a fixed delay, D, of the FDL which it always traverses.

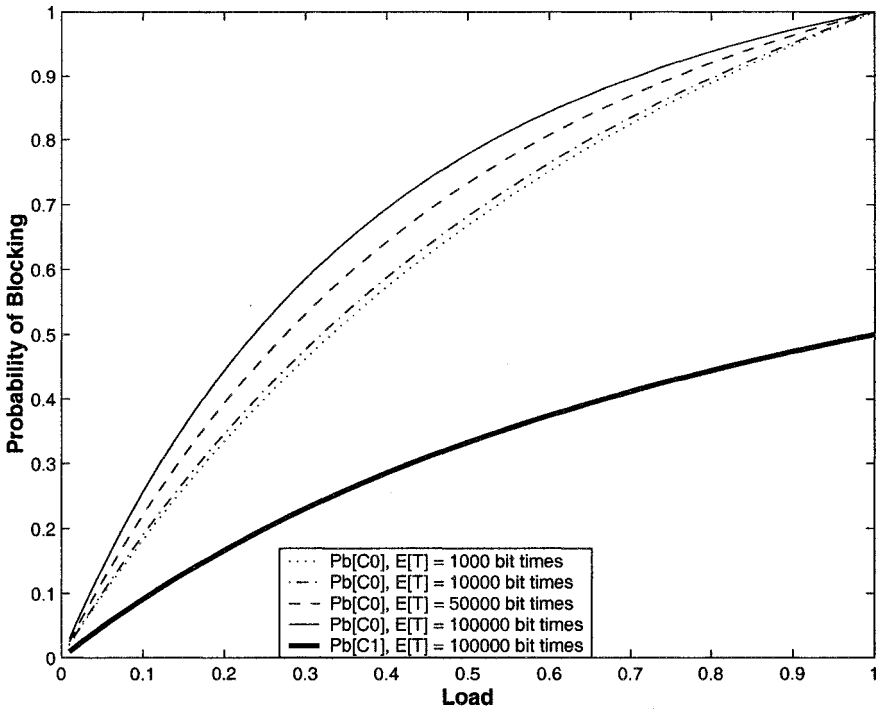

Figure 5.17: The blocking probabilities for different loads for the C_1 and C_0 streams with different mean transmission times

5.7 Conclusion

We evaluated the performance of optical switches with feed-forward output fiber delay lines using an analytical model, which in turn was verified via simulation. Two different forwarding algorithms for the switch were presented and evaluated: a simple forwarding algorithm that is easier to implement, and an enhanced forwarding algorithm with better performance in terms of both probability of blocking and packet delay, but with increased complexity. The models can be utilized to evaluate the effects of the different parameters on the performance of the considered optical switch. Finally, we demonstrated how QoS may be achieved using the same switching architecture, and evaluated its performance.

Chapter 6

SHARED-FEEDBACK-FDLS BASED OPTICAL SWITCH

6.1 Introduction

Previous works on performance evaluation of the optical shared-buffer switch present a limited characterization of the performance metrics of the switch compared to what is presented in this work. We follow a new approach, based on the Surjective-Mapping based Model (SMM), to model the shared-feedback FDL optical switch. The model is simple, tractable, and yet provides a comprehensive insight into the performance of the optical shared-buffer switches. The model is also applicable for evaluating large switches. For instance, a set of 922 equations is generated using the RMC approach presented in [5] for a switch with $N = 16$ and $K = 8$, while a set of only $(N + K)$ equations is generated using the SMM approach to study the same switch. Therefore, the model provides a method to evaluate performance of switches significantly larger than what RMC based models can feasibly capture. Moreover, the SMM model provides full characteristics of the traffic flow at different stages inside the switch. Simulation based results are used to verify the analytical results generated using the SMM based model.

6.2 Optical switch with feedback shared-buffer

Figure 6.1 shows the optical shared-buffer switching architecture that is evaluated in this work. The $N \times N$ switch consists of N primary inputs and N primary

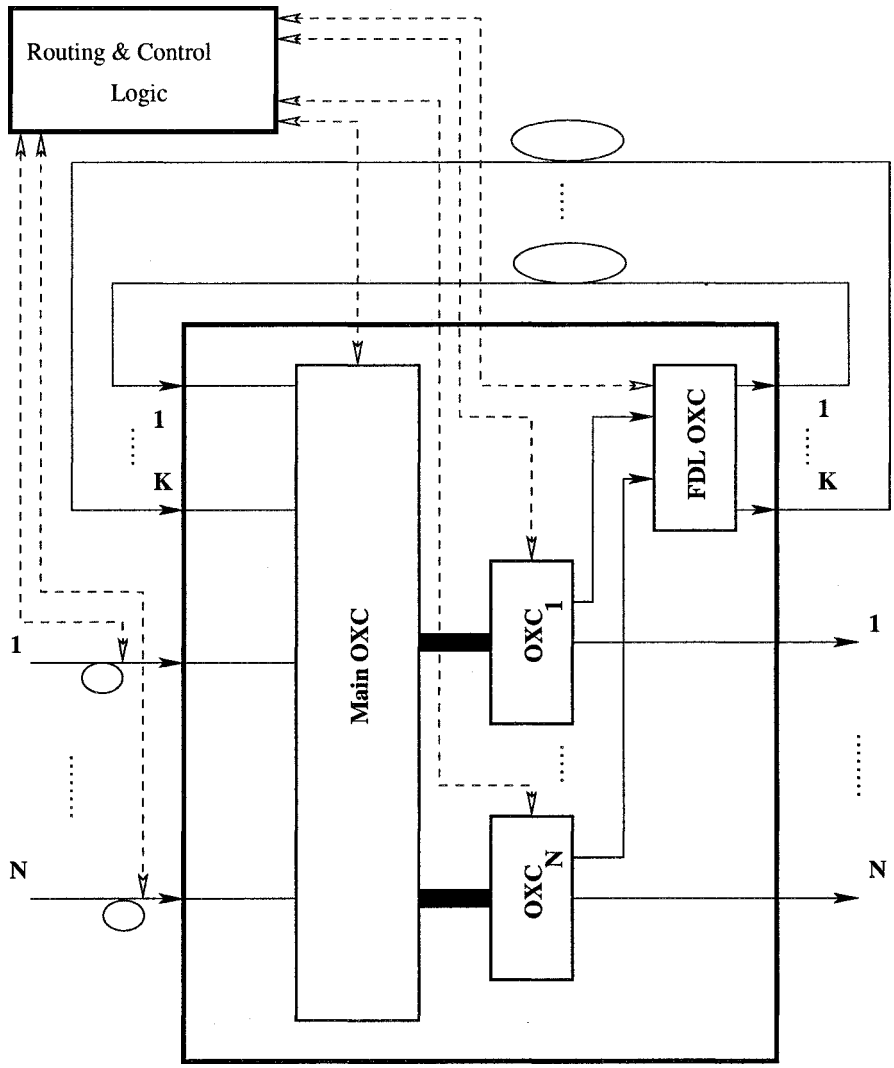

Figure 6.1: Optical packet switch with feed-back FDLs

outputs. The switch is augmented with K FDLs. Each fiber line can either be passive or augmented with an amplifier [5]. The inputs to the switch from the K FDLs will be referred to as secondary inputs. The propagation delay of each line is equal to the time period of a single slot. A slotted system is considered where the duration of the slot is equal to the transmission time of packets of fixed length. After propagating in a delay line, a packet presents itself back at the inputs of the switch and competes for the required output port in the following time slot. The main optical cross-connect ($Main\ OXC$) switches packets arriving on primary and secondary inputs to the output OXC corresponding to the destination address of the packet. If only one packet is forwarded to an OXC_i, it is forwarded to the output port i. When multiple packets compete for the same port, the related OXC forwards one of the packets, chosen randomly, to that output port while forwarding the rest of the competing packets to the FDLs' cross-connect ($FDL\ OXC$) which forwards the packets to the FDLs. If the number of incoming packets to $FDL\ OXC$ is greater than K, the total number of FDLs, K packets, randomly chosen, are forwarded to the K available FDLs and the rest of the packets are dropped. Note that the criteria for selecting a packet to be dropped can be selecting the packet that has encountered the longest delay among the competing packets. Some other criteria for selecting a packet to be dropped can also be considered. In this work, the packets to be dropped are selected randomly in deriving the model; however, the values for throughput, average delay, loss probability, and delay line utilization are still valid regardless of the criteria used to select a packet to be dropped.

6.3 Performance Analysis

This section presents the SMM based performance model. Appendix C shows the source code, which is written in Maple, for implementing the analytical model

discussed in this section. This model addresses the symmetric case where the load is uniformly distributed among the primary input ports, and destinations of packets are uniformly distributed among the primary output ports. The switch operates in a slotted mode, in which incoming packets are aligned at the inputs, and all are processed and switched during a time slot. Let X and Y be the random variables representing the number of packets present in a given time slot at the secondary and primary inputs, respectively. X takes values from 0 to K while Y takes values from 0 to N. The probability that a packet arrives at one of the primary input lines in a given time slot is denoted by ρ, which corresponds to the normalized offered load. Since packet arrivals at primary inputs are assumed to be independent of each other, the number of packet arrivals at the primary inputs in a given time slot follows Binomial distribution, i.e.,

$$P(Y = y) = \binom{N}{y} \rho^y \, (1-\rho)^{N-y} \tag{6.1}$$

Let Z be the random variable representing the total number of packets arriving at both primary and secondary inputs, i.e.,

$$Z = X + Y \tag{6.2}$$

The packet arrivals during a given time slot at the primary inputs, represented by Y, are assumed to be independent of the packet arrivals during the previous time slot, on which the random variable X depends. Therefore, the distribution of Z is derived by convoluting both X and Y, and hence,

$$P(Z = z) = \sum_{x=0}^{z} P(X = x) P(Y = z - x) \tag{6.3}$$

Note that Z can take values from 0 to $(N + K)$. Let T be the random variable representing the total number of packets that are forwarded from all OXC_is to the $FDL\ OXC$. The minimum value of T is 0, corresponding to the case where either all Z packets leave the switch through the primary outputs with no competition

among themselves or there is no packet arrival at the primary outputs, i.e., $z = 0$. The maximum value of T is $(N + K - 1)$, corresponding to the case where $Z = N + K$, and all these $(N + K)$ packets are addressed to the same output port. As only one packet leaves the switch through that port, the remaining $(N + K - 1)$ packets, represented by T, are forwarded to the FDLs. The distribution of T, $P(T = t)$, can be determined conditioned on Z as follows:

$$P(T = t) = \sum_{z=t+1}^{Z_L} P(T = t|Z = z)P(Z = z) + \delta[t]P(Z = 0) \tag{6.4}$$

Note that the second term of the equation, $\delta[t]P(Z = 0)$, corresponds to the situation where there is no packet arrival at the primary outputs, an event with probability $P(Z = 0)$, in which case no packets will be forwarded to the FDLs, i.e., $t = 0$. The upper limit of the summation, Z_L, represents the maximum value of z that can result in no more than t packets to be forwarded to the FDLs. Since at most N packets can be forwarded to the primary outputs, for a given t, Z_L cannot exceed $(N + t)$. On the other hand, the maximum possible value of Z is $(N + K)$, corresponding to the case where packets arrive on all primary and secondary inputs. Thus, when $t > K$, Z_L is $(N + K)$. Therefore,

$$Z_L = N + min(t, K) \tag{6.5}$$

The event $T = t$ implies that out of z packet arrivals at the output ports, $(z - t)$ packets leave the switch though $(z-t)$ output ports while t packets are forwarded to the FDLs due to their contention with these $(z-t)$ packets. The $(z-t)$ output ports through which the $(z - t)$ packets leave the switch are randomly chosen by these packets, as the output ports desired by different packets are independent of each other and uniformly distributed among the N primary output ports. Therefore, these $(z - t)$ ports are selected randomly from the N total output ports in $\binom{N}{z-t}$ different ways, each of which is equally likely. The total number of combinations

through which z packets may attempt to leave the switch through N output ports is given by N^z. A subset of these total combinations results in having $(z-t)$ packets leave the switch while t packets are forwarded to the FDLs. To identify the cardinality of this subset, Surjective-mapping technique [20] can be utilized. Surjective mapping provides a number of different ways, $S_{m,n}$, to map an m source

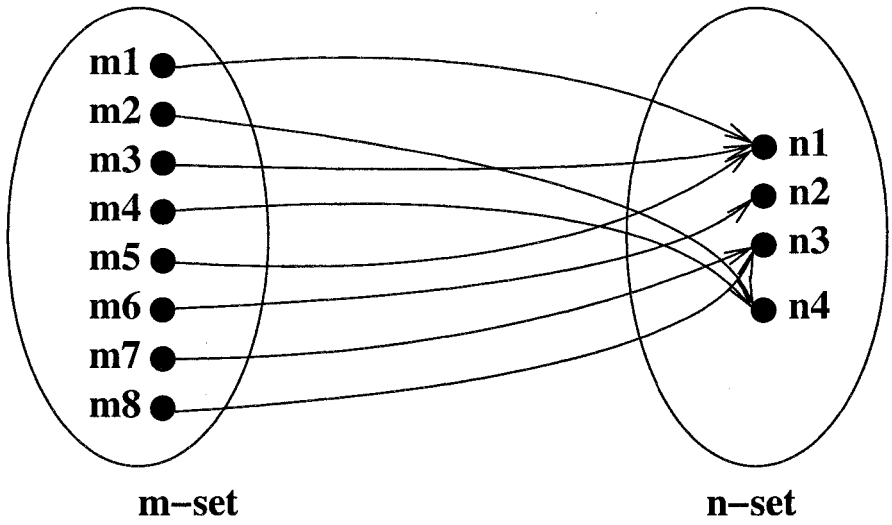

Figure 6.2: Surjective mapping from m-set to n-set

set to an n destination set such that every element of the destination n-set elements is mapped to by at least one element of the source m-set elements. For example, Figure 6.2 shows a source m-set and a destination n-set with $m = 8$ and $n = 4$. Each element of the four elemets in the n-set is mapped to by at least one element of the eight elements from the m-set. In the optical switch under consideration, the $(z-t)$ output ports correspond to the destination n-set, while the z packet arrivals correspond to the source m-set. The idea here is to surjectively map z packet arrivals to $(z-t)$ output ports such that each of the $(z-t)$ ports receives at least one packet from the z packet arrivals. Since each output port can only serve one packet in a time slot, in case more than one packet is addressed to an output port of these $(z-t)$ ports, one packet will be served by that port while

the rest will be forwarded to the FDLs. This mapping implies that there will be competition among the z packets since $z \geq z - t$. As such, $(z - t)$ packets leave the switch through the $(z - t)$ output ports while t packets are forwarded to the FDLs. The formula to identify the number of ways to surjectively map z packets to $(z - t)$ output ports, $S_{z,z-t}$, is given by [20],

$$S_{z,z-t} = \sum_{i=0}^{z-t} (-1)^i \binom{z-t}{i} (z - t - i)^z \tag{6.6}$$

The cardinality of the subset that results in t packets getting forwarded to the FDLs can therefore be identified by multiplying the number of $(z-t)$ output ports that can be randomly selected from the N total output ports, $\binom{N}{z-t}$, by the number of ways to surjectively map z packets to $(z - t)$ output ports, $S_{z,z-t}$. Therefore, for a given number of packet arrivals at the primary outputs, z, the probability of having t packets getting forwarded to the FDLs can be found by dividing the cardinality of the subset that results in t packets getting forwarded to the FDLs, which is $\binom{N}{z-t} S_{z,z-t}$, by the total number of ways z packets attempt to leave the switch through the N output ports, which is equal to N^z. Therefore,

$$P(T = t | Z = z) = \frac{\binom{N}{z-t} S_{z,z-t}}{N^z} \tag{6.7}$$

The random variable X corresponding to the number of packet arrivals at the secondary inputs is equal to T except in the case when $T \geq K$, for which X is equal to K. The ditribution of X is given by,

$$P(X = x) = \begin{cases} P(T = x) & if\ x < K \\ \sum_{j=K}^{N+K-1} P(T = j) & if\ x = K \end{cases} \tag{6.8}$$

Using Equations (6.3), (6.7), and (6.8), the distribution of T, Equation (6.4), can be evaluated. The mean number of packets that enter the switch in a given time

slot, $E[Y]$, is the mean of the binomial distribution associated to the random variable, Y, and hence,

$$\begin{aligned} E[Y] &= \sum_{y=0}^{N} yP(Y=y) \\ &= N\rho \end{aligned} \tag{6.9}$$

The mean number of packets that are dropped in a time slot, $E[\Phi]$, is the mean number of packets represented by T that exceeds the number of FDLs, K, and hence,

$$E[\Phi] = \sum_{t=K+1}^{N+K-1} (t-K)P(T=t) \tag{6.10}$$

Thus, the probability of blocking of the switch, P_B, is the ratio of the mean number of packets that are dropped to the mean number of packets that enter the switch in a given time slot. Thus,

$$\begin{aligned} P_B &= \frac{E[\Phi]}{E[Y]} \\ &= \frac{1}{N\rho} \sum_{t=K+1}^{N+K-1} (t-K)P(T=t) \end{aligned} \tag{6.11}$$

The utilization of the FDLs is load dependent. This parameter helps identify the suitable number of FDLs when designing optical shared-buffer switch. Let the fraction of FDLs that is utilized in a given time slot be U: U is $\frac{t}{K}$ when $t \leq K$ and 1 when $t > K$. The FDL utilization is therefore,

$$U = \frac{1}{K} \sum_{t=0}^{K} tP(T=t) + \sum_{t=K+1}^{N+K-1} P(T=t) \tag{6.12}$$

Let the probability that a packet re-circulates in the switch during a given time slot be ξ. It is the ratio of the number of packets that re-circulate in the switch to the total number of packets attempting to leave the switch, which is represented

by the random variable Z. The number of packets that re-circulate in the switch is equal to t when $t \leq K$, and equal to K when $t > K$. Therefore,

$$\xi = \sum_{t=1}^{K} \sum_{z=t+1}^{N+t} \frac{t}{z} P(T = t|Z = z)P(Z = z) + \sum_{t=K+1}^{N+K-1} \sum_{z=t+1}^{N+K} \frac{K}{z} P(T = t|Z = z)P(Z = z) \quad (6.13)$$

Let D be the random variable representing the delay in terms of the number of time slots a packet spends due to re-circulating in the switch. Note that for delay evaluation, we consider only the packets that eventually leave the switch, as opposed to the packets that are dropped. Therefore, the delay distribution is to be conditioned on the fact that the packets under consideration eventually leave the switch.

Let Ψ represent the event that a packet eventually leaves the switch. Thus, the distribution of the delay the packets encounter may be expressed as $P(D = d|\Psi)$, and hence,

$$P(D = d|\Psi) = \frac{P(\{D = d\} \cap \Psi)}{P(\Psi)} \quad (6.14)$$

Let the probability that a packet among z packets leaves the switch be σ. This probability is the ratio of the number of packets that leave the switch, $(z - t)$, to the total number of packets attempting to leave the switch, z. Therefore,

$$\sigma = \sum_{t=0}^{N+K-1} \sum_{z=t+1}^{N+t} \frac{(z-t)}{z} P(T = t|Z = z)P(Z = z) \quad (6.15)$$

In order to identify the probability $P(\Psi)$, all possible numbers of re-circulations a packet might experience after which it leaves the switch should be considered. Therefore,

$$\begin{aligned} P(\Psi) &= \sum_{d=0}^{\infty} \xi^d \sigma \\ &= \frac{\sigma}{1-\xi} \end{aligned} \quad (6.16)$$

Now, $P(\{D = d\} \cap \Psi)$ is the probability that a packet encounters d re-circulations after which it leaves the switch. Therefore, this probability can be written as,

$$P(\{D = d\} \cap \Psi) = \xi^d \sigma \tag{6.17}$$

Therefore, Equation (6.14), can be written as,

$$\begin{aligned} P(D = d|\Psi) &= \frac{\xi^d \sigma}{P(\Psi)} \\ &= \xi^d (1 - \xi) \end{aligned} \tag{6.18}$$

Note that, the number of re-circulations a packet experiences in the switch follows a geometric distribution whose parameter is ξ. Each packet among the z packet arrivals is equally likely to be forwarded to the FDLs, i.e., independent of how many re-circulations a packet has experienced in the switch so far. Therefore, the number of encountered re-circulations a packet experiences follows geometric distribution. The average delay a packet encounters in the switch is therefore,

$$\bar{D} = \frac{\xi}{1 - \xi} \tag{6.19}$$

For a given switch configuration, it would be useful to identify the delay, in terms of the number of re-circulations, most of the packets encounter in the switch. Let D_U be the number of re-circulations that makes $P(D > D_U) < \epsilon$. From the CDF of the delay distribution,

$$D_U = \lceil \frac{Ln(\epsilon)}{Ln(\xi)} - 1 \rceil \tag{6.20}$$

6.4 Results

A simulator was developed to evaluated the performance of optical shared-buffer switches with feedback FDLs. The results from analytical and simulation models for the slotted optical shared-buffer switch are presented in this section. The destination of a packet arriving at input ports of the switch are uniformly

distributed among all the output ports. In case of competition, the preferred output port is given randomly to one of the competing packets from either the primary or secondary inputs, while the rest of the competing packets are forwarded to the FDLs. Figure 6.3 compares the blocking probability values for 4×4 switches

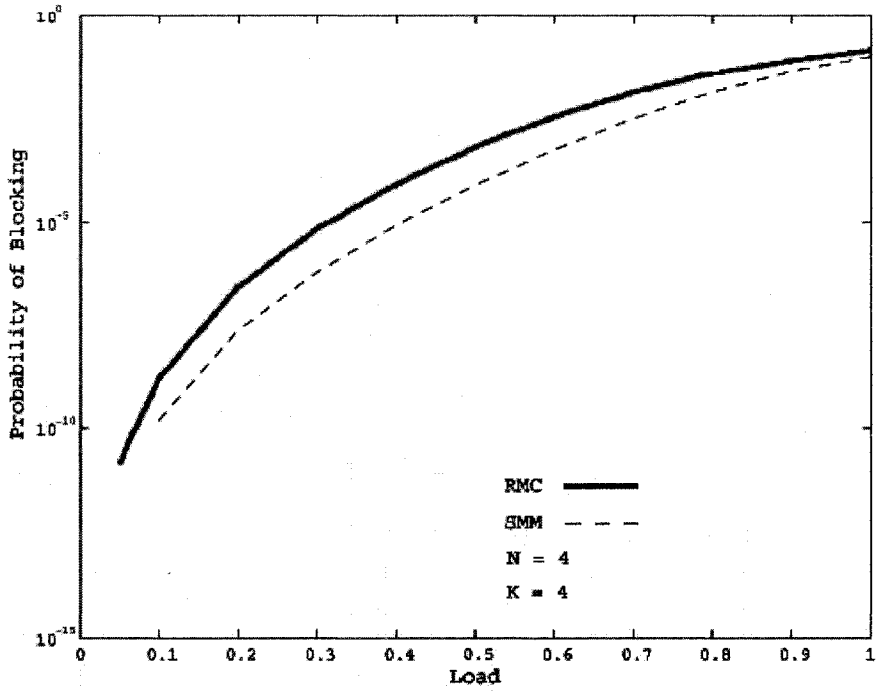

Figure 6.3: The loss probabilities from the RMC based model [5] and from SMM based model for $N = K = 4$

with 4 FDLs based on the RMC model, presented in [5], with those from the SMM based model. The SMM based results show a good match to those of RMC based model. Yet, the latter approach is limited to switches with small nodal degrees, whereas the proposed SMM based model provides a feasible method to model optical switches with high nodal degree augmented with any number of FDLs. Figure 6.4 shows analytical results for probability of blocking, calculated using Equation (6.11) as well as simulation based results, for 8×8, 16×16, 32×32, and 64×64, optical shared-buffer switches, each of which is augmented with 8 FDLs.

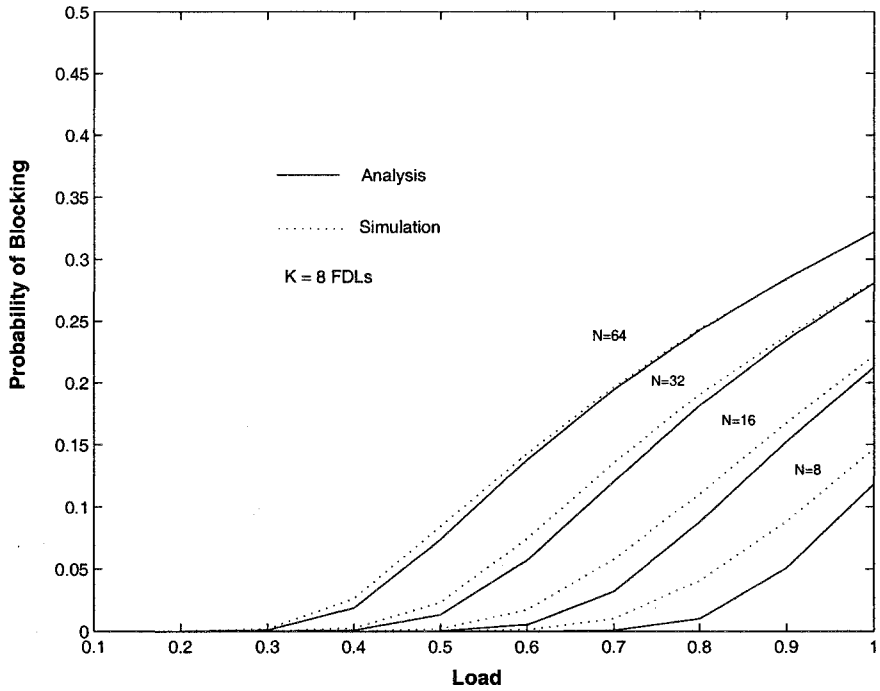

Figure 6.4: The loss probability of optical shared-buffer switch with different nodal degrees at different loads

Note that as the number of FDLs (K) approaches the number of primary inputs N, as in the case of 8×8 optical shared-buffer switch with 8 FDLs, the analytical results differ slightly from those of the simulation. This is due to the approximation that, at each time slot, the numbers of packets arriving at the secondary inputs are assumed to be independent and identically distributed (*iid*) random variables, each of which is thus represented by the random variable X. In fact, in a given time slot, the number of packets arriving at the secondary inputs depends on the number of packet arrivals at these inputs in the previous time slot. However, this dependency becomes marginal as N moderately exceeds K. That is due to the fact that as the number of primary inputs (N) becomes moderately higher than the number of the secondary inputs (K), the number of packets entering the FDLs is dominated by packet arrivals from the primary inputs and hence the dependency between packet arrivals from the FDLs in successive time slots is marginal. From a practical perspective, it is reasonable to assume K to be significantly less than N due to the high hardware complexity required for the switch implementation as more FDLs are employed. Furthermore, as shown in Figure 6.5, the performance enhancement, in terms of the probability of blocking, achievable by employing more FDLs diminishes with K for a value of K significantly less than N. The case where $N = 64$ and $K = 8$ in Figure 6.4 reinforces the fact of the marginal dependency between packet arrivals from the FDLs in successive time slots as N becomes larger than K. Therefore, in such cases, when N is larger than K, the SMM based model provides extremely accurate results as these results perfectly match the simulation results. Figure 6.6 shows the delay distributions of both the analytical model, generated using Equations (6.13) and (6.18), and the simulation model for 16×16, 32×32, and 64×64 optical shared-buffer switches, each of which is augmented with 8 FDLs, at loads of 100%, 50%, and 10% respectively. The close agreement between analytical and simulation results also indicates that there is no

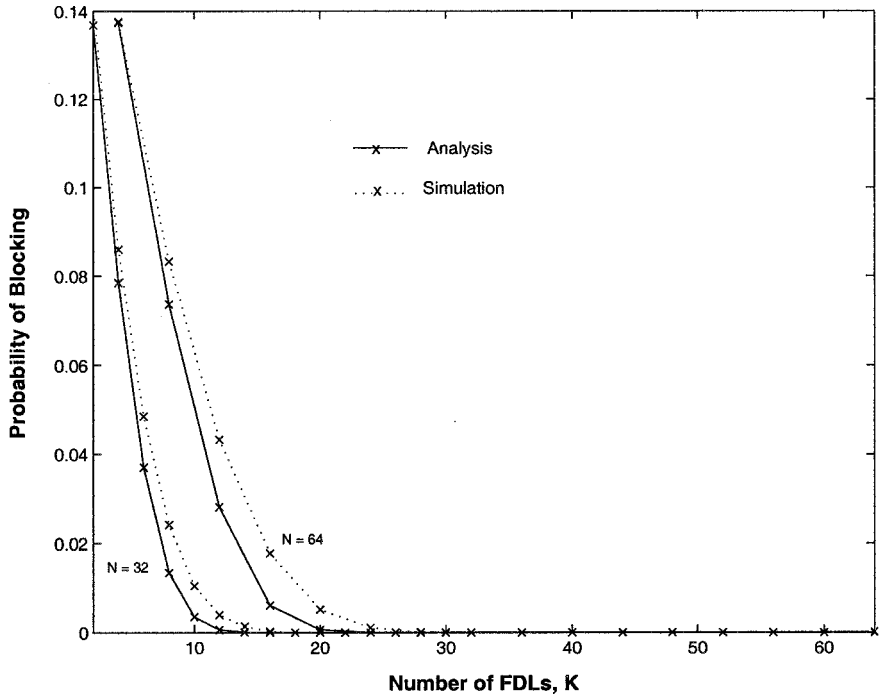

Figure 6.5: Variation of the loss probability of optical shared-buffer switch with the number of FDLs at 50% load

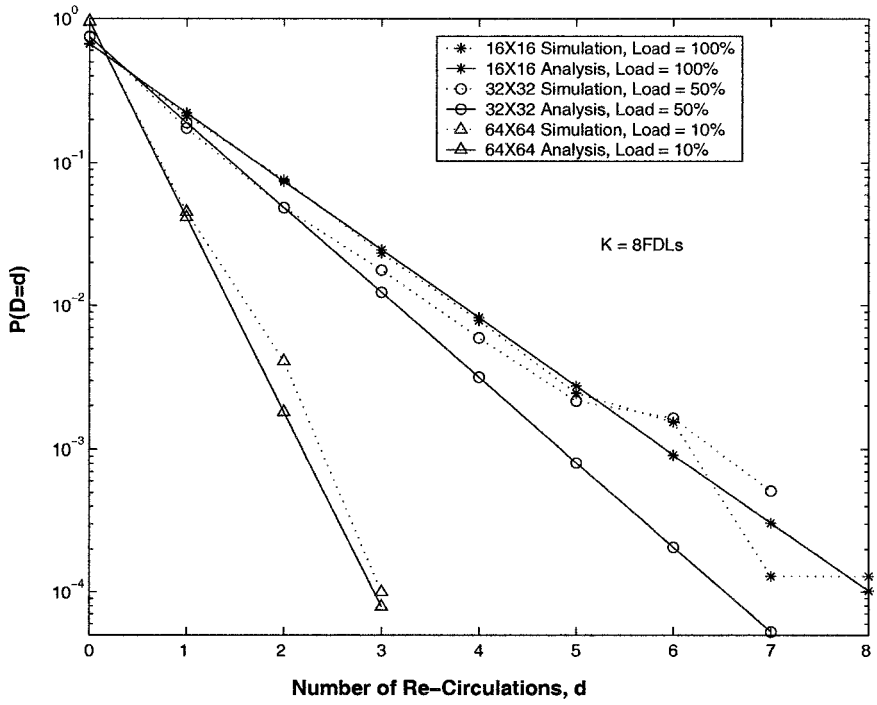

Figure 6.6: The delay distribution of delivered packets

significant dependency between packets coming from the FDLs in successive time slots on the model. Figure 6.7 shows the average delays a packet encounters, calculated using Equation (6.19), in 16×16, 32×32, and 64×64 optical shared-buffer switches, each of which is augmented with 8 FDLs. The average delay reveals an increasing trend up to a certain load after which it decreases. This decrease in the average delay is due to the higher probability of blocking that takes place at higher load values when the chance of a packet re-circulating in the switch becomes smaller. Figure 6.8 shows the average delay a packet encounters in 16×16, 32×32, and 64×64 optical shared-buffer switches, each of which is augmented with a different number of FDLs at 50% load. The average delay curves saturate after certain values of the number of FDLs. This behavior is due to the fact that, for a given load, a certain number of FDLs is enough to handle 100% of the packet arrivals. Beyond this number of FDLs, employing more FDLs does not

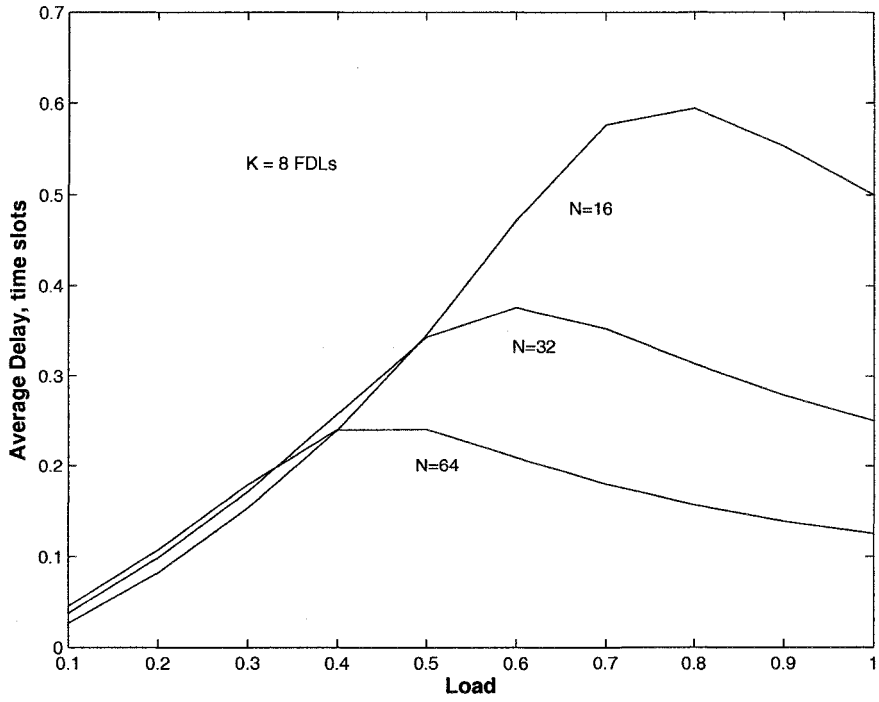

Figure 6.7: The average delay delivered packets for different loads and switch sizes

improve the performance in terms of the average delay. Figure 6.9 shows the bound of the delay encountered by $(1-\epsilon) \times 100\%$ of the packets, where ϵ is set to 0.0001, in 16×16, 32×32, and 64×64, optical shared-buffer switches, each of which is augmented with 8 FDLs. Figure 6.10 shows a similar trend to that revealed in Figure 6.7. The figure shows the probability of a packet re-circulating in the switch, generated using Equation (6.13). The decreasing trend is attributed to the higher probability of blocking associated with higher loads. Figure 6.11 shows the probability of a packet re-circulate in optical cross-connects with different nodal degrees at load 50%. The figure shows that more packets get chance to re-circulate in the switch as more FDLs is considered up to a certain number. Beyond that number, considering more FDLs implies an overcapacity that is wasted as 100% of the packets have already been served by a fewer FDLs. Figure 6.12 shows the FDL utilization calculated using Equation (6.12) for the optical switch with different

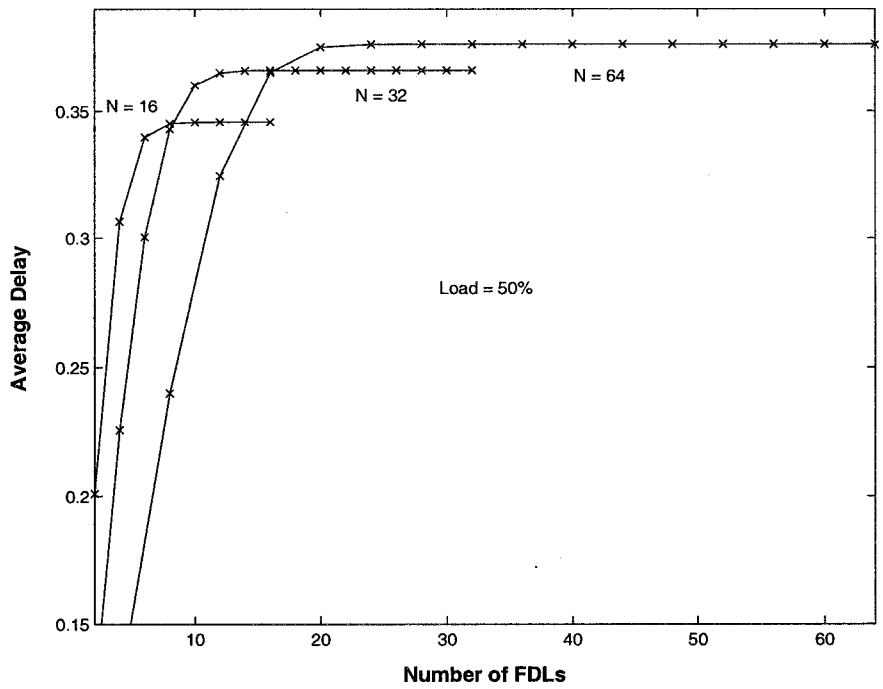

Figure 6.8: The average delay a packet encounters vs. the number of FDLs for different nodal degrees at a load of 50%

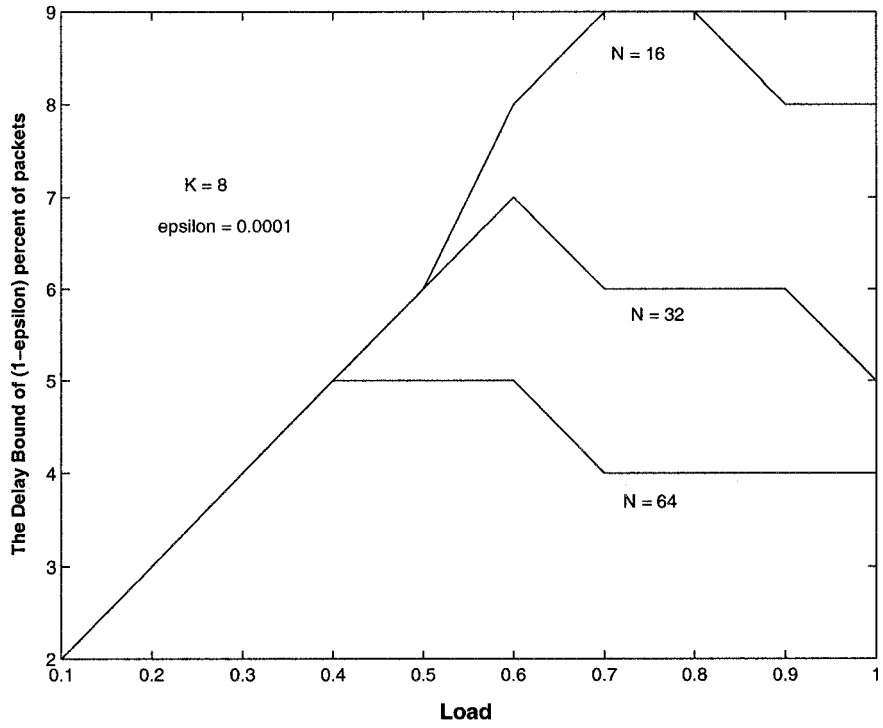

Figure 6.9: The bound of the delay $(1-\epsilon)$ percent of the packets encounter with $\epsilon = 0.0001$

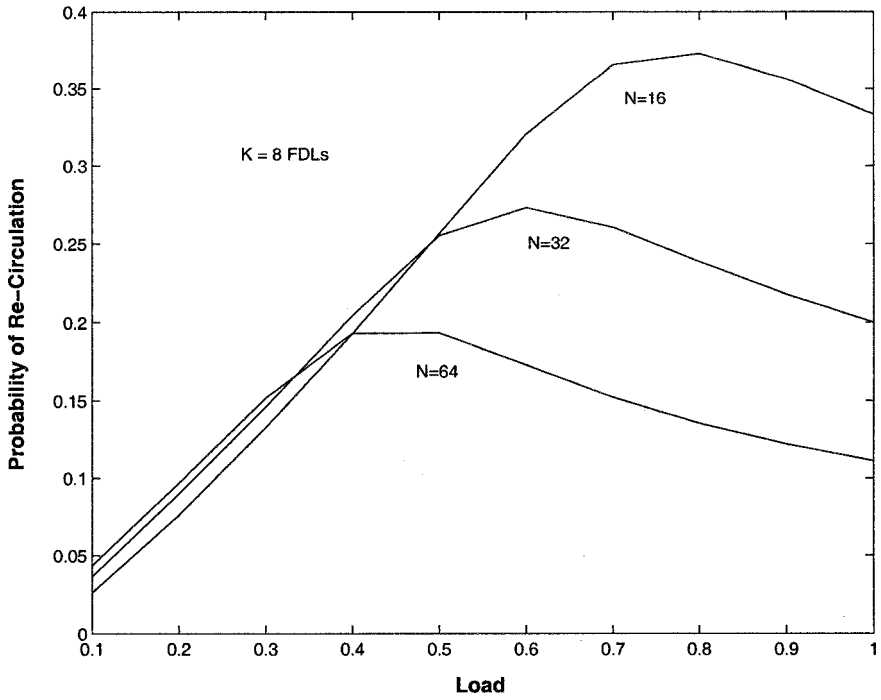

Figure 6.10: The probability of packet re-circulations vs. load for different nodal degrees and 8 FDLs

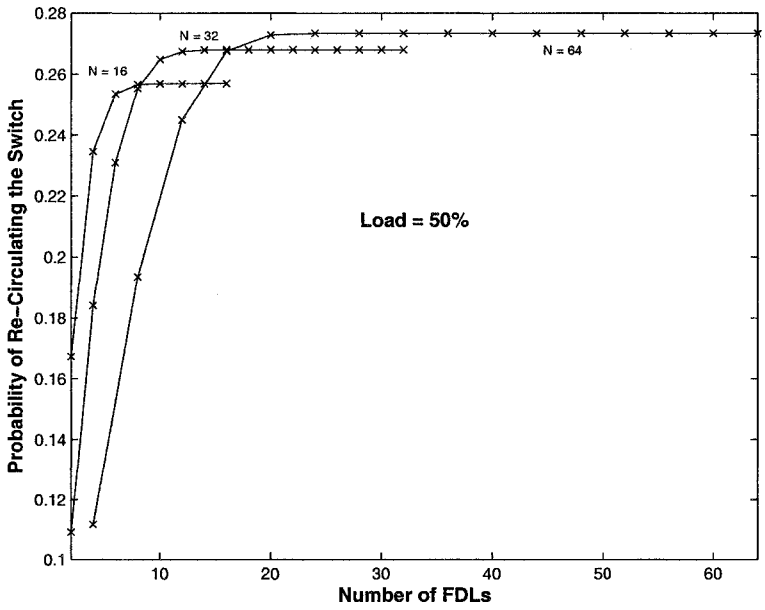

Figure 6.11: The probability of a packet re-circulate in optical cross-connects with different nodal degrees at load 50%

nodal degrees, and $K = 8$ FDLs. The figure shows that for a given number of FDLs, the higher the nodal degree, the higher the utilization of FDLs. As the nodal degree of the switch increases, utilization reaches 100% at lower loads.

6.5 Conclusion

We developed a Surjective-Mapping based Model (SMM) to evaluate the performance of slotted optical shared-buffer switches utilizing feedback fiber delay lines for optical buffering. The model is especially useful for performance evaluation of switches for which Markovian based models become very complicated. The SMM approach requires significantly fower computations, $O(N + K)$, compared to Markovian based models. The model can thus be used to solve large switches for which the RMC based approach becomes untractable. Moreover, the model presented in this work provides a comprehensive insight into the performance

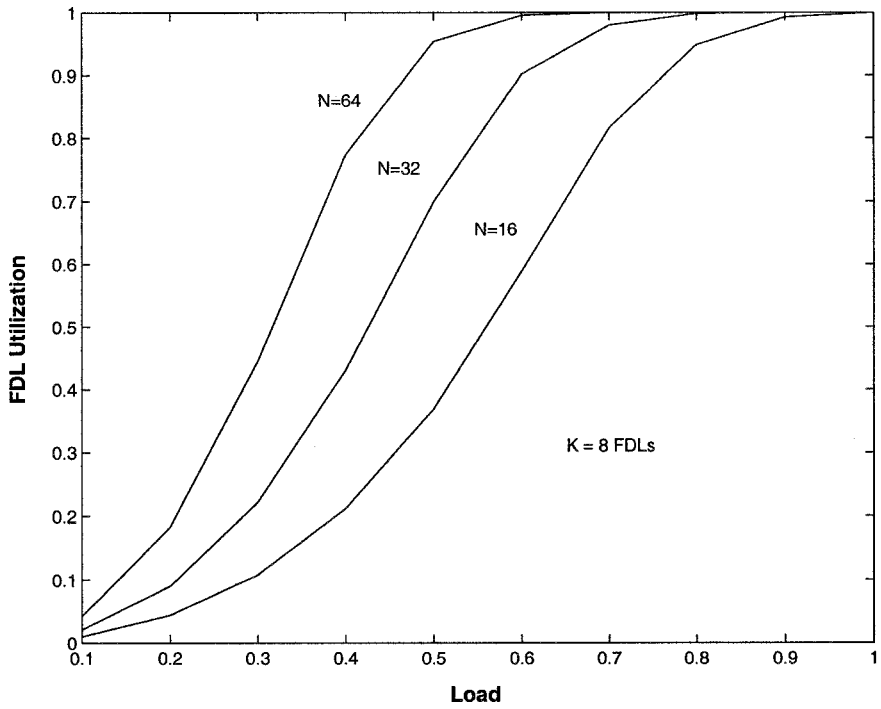

Figure 6.12: The FDL utilization of the 8 FDLs for optical shared-buffer switch with different nodal degrees

characteristics of the switch. The SMM technique realized in this work is a new approach that can further be utilized to evaluate different slotted communication systems.

Chapter 7

CONCLUSION

The dissertation investigated the use of both feed-forward and feedback fiber delay lines in all-optical switches as optical buffering schemes for contention resolution. In this chapter, we first outline the significance of buffering scheme as well as the work undertaken and reported in this dissertation and the main outcomes of this research are highlighted. Next, the many possible areas to which this work can be extended are discussed.

7.1 Significance of this work

Despite the efforts of many researchers in the area of optical buffering, the practical realization of these switches remains challenging. One main reason behind this is the fact that FDLs can be utilized at different stages of the switch at different levels of hierarchies through different forwarding algorithms. These alternatives suggest trade-offs between the cost and the performance. The main goal of this research was to understand the optimal combinations among configuration parameters, forwarding algorithms and the cost in all-optical switches.

The work proposed algorithms for feed-forward optical buffering that utilize the least number of FDLs, and developed a comprehensive and yet simple model for feedback optical buffering. For feed-forward optical buffering, the proposed algorithms aim at utilizing the optical buffers more effectively. The performance

of these algorithms is evaluated considering optical switches with a single feed-forward fiber delay line per output port. The model is verified using simulation results. Two forwarding algorithms were considered: a simple forwarding algorithm that requires simple control and management, and an enhanced forwarding algorithm with better performance in terms of both probability of blocking and packet delay, but requiring more control and management overhead. The models developed consider a single FDL per output and can be utilized to evaluate the effects of the different parameters on the performance of the considered optical switch. Finally, we demonstrated that different QoS for different classes of packets is achievable using the same switching architecture, and we evaluated the corresponding performance.

For the case of feedback optical buffering, we developed a Surjective-Mapping based Model (SMM) to evaluate the performance of optical shared-buffer switches utilizing feedback fiber delay lines for optical buffering. This model is especially useful for performance evaluation of switches for which Markovian based models become very complicated. The SMM approach requires significantly fewer computations, $O(N + K)$, where N is the nodal degree and K is the number of FDLs, compared to the computation Markovian based models require. The model can thus be used to solve large switches for which a Markovian based model becomes untractable. Moreover, the model presented in this work provides a comprehensive insight into the performance characteristics of the switch. The SMM technique realized in this work is a novel approach that can further be utilized to evaluate different slotted communication systems.

7.2 Areas for future research

Several issues related to optical switches can be further investigated as an extension of this work so as to complete the framework for studying the optical

buffering mechanisms. While some of these aspects are related to direct extension of this work, others cover different challenges.

For example, extension of the analysis conducted for the single-FDL based feed-forward optical buffer to analyze feed-forward optical buffer with multiple FDLs is possible. A model is needed for wavelength-convertible based optical switch employing feed-forward optical buffers with multiple FDLs. A further extension may cover the analysis of an optical switch employing both feedback and feed-forward optical buffers under both synchronous and asynchronous operation modes. Moreover, the analysis of both optical buffering schemes may be extended to incorporate the effect of wavelength converters. Further progress in this arena is required to develop a thorough understanding of the integration of optical buffering, internal switching components, and control and management mechanisms.

Bibliography

[1] F. A. Al-Zahrani, A. A. Habiballa, A. G. Fayoumi, and A. P. Jayasumana, "Performance Tradeoffs of Shared Limited Range Wavelength Conversion Schemes in Optical WDM Networks," IEEE and IFIP International Conference on wireless and Optical Communications Networks (WOCN 2005), Dubai, UAE, March 2005.

[2] Agiltron Incorporation for Optical Solutions: http://www.agiltron.com/fiberdelayline.html

[3] R. Almeida, J. Pelegrini, H. Waldman, "A generic-traffic optical buffer modeling for asynchronous optical switching networks," IEEE Communications Letters, Vol. 9 , No. 2 , pp. 175 - 177, February 2005

[4] R. Almeida, J. Pelegrini, H. Waldman, "Optical buffer modelling for performance evaluation considering any packet inter-arrival time distribution," IEEE International Conference on Communications, Vol. 3 , pp. 1771-1775, June 2004

[5] P. Bergstrom, M. Ingram, A. Vernon, J. Hughes, P. Tetali, "A Markov chain model for an optical shared-memory packet switch," IEEE Transactions on Communications, Vol. 47, No. 10, 1593-1603, October 1999

[6] D. Bertsekas, and R. Gallager, *Data Networks*, Prentice Hall, New Jersey, Second Edition, 1992

[7] G. Bianchi, J. Turner, "Improved queuing analysis of shared buffer switching networks," IEEE/ACM Transactions on Networking, Vol. 1, pp. 482-490, August 1993

[8] F. Callegati, "On the design of optical buffers for variable length packet traffic," Proceedings of IEEE Ninth Conference on Computer Communications and Networks, pp. 448 - 452, October 2000

[9] F. Callegati, "Optical buffers for variable length packets," IEEE Communications Letters, Vol. 4, No. 9, pp. 292 - 294, September 2000

[10] F. Callegati, D. Careglio, W. Cerroni, J. Sole-Pareta, "Assessment of packet loss for an optical feedback buffer node using slotted variable length packet and heavy tailed traffic," in Proc. 4th IEEE International Conference on Transparent Optical Networks (ICTON2002), Poland, April 2002

[11] P. Cameron, *Combinatorics: Topics, Techniques, Algorithms* Cambridge University Press, 1994

[12] G. Castanon, "Design-dimensioning model for transparent WDM packet-Switched irregular networks," Journal of Lightwave Technology, Vol. 20, No. 1, pp. 1-9, January 2002

[13] S. Danielsen, B. Mikkelsen, C. Joergensen, T. Durhuus, and K. Stubkjaer, "WDM packet switch architectures and analysis of the influence of tunable wavelength converters on the performance," Journal of Lightwave Technology, Vol. 15, No. 2, pp. 219-27, February 1997

[14] J. Elmirghani, H. Mouftah, "All-optical wavelength conversion: technologies and applications in DWDM networks," IEEE Communication Magazine, Vol. 38, No. 3, pp. 86-92, March 2000

[15] EMCORE Corporation: http://www.emcore.com/assets/fiber/ds00-306_EM.pdf

[16] A. Fayoumi and A. Jayasumana, " An analysis of an all-optical cross-connect using feed-forward optical buffers with and without QoS," sent to IEEE Journal of Lightwave Technology, sent for publication

[17] A. Fayoumi and A. Jayasumana, " A Surjective-mapping based model for optical shared-buffer cross-connect," sent to IEEE/ACM Transaction on Networking.

[18] A. Fayoumi and A. Jayasumana, "Performance evaluation of an all-optical shuffleNet using optical buffering and deflection routing," Photonic Network Communication, Vol. 6, No. 1, pp. 43-50, July 2003

[19] A. Fayoumi, "Performance evaluation of multihop networks using optical buffering", MS Thesis, Colorado State University, 2001

[20] A. Fayoumi and A. P. Jayasumana, "Performance model of an optical switch using fiber delay lines for resolving contention" Proceedings of IEEE International Conference on Local Computer Networks, Bonn, Germany, pp. 178-186 October 2003

[21] R. Geldenhuys, F. Leuschner, Y. Liu, G. Khoe, N. Calabretta, H. Dorren, "Selecting fiber delay line distributions for traveling buffers in an all-optical packet switched cross-connect," Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering, Vol. 2 , pp. 889-892, May 2003

[22] C. Guillemot, etc. al., "Transparent optical packet switching: the European ACTS KEOPS project approach," Journal of Lightwave Technology No. 16, pp. 2117-2134, 1998

[23] P. Hansen, S. Danielsen, K. Stubkjaer,"Optical packet switching without packet alignment," Proceedings of European Conference on Optical Communications, Vol. 1, pp. 591-592, Madrid, Spain, September 1998

[24] H. Harai, N. Wada, F. Kubota, and W. Chujo, "Contention resolution using multi-stage fiber delay line buffer in a photonic packet switch," Proceedings of IEEE International Conference on Communications (ICC 2002), Vol. 5, pp. 2843-2847, 2002

[25] D. Hunter, M. Chia, and I. Andonovic, "Buffering in optical packet switches," Journal of Lightwave Technology, Vol. 16, No. 12, pp. 2081-2094, December 1998

[26] D. Hunter, etc. al., "WASPNET: a wavelength switched packet network," IEEE Communications Magazine, Vol. 37, No. 3, pp. 120 - 129, March 1999

[27] R. Jain, The Art of Computer Systems Performance Analysis, John Wiley & Sons, Inc., New York, 1991

[28] K. Laevens, H. Bruneel, "Analysis of a single-wavelength optical buffer," Proceedings of IEEE Information on Communications (INFOCOM 2003), Vol. 3, pp. 2262-2267, March 2003

[29] L. Li, S. Scott, J. Deogun, "A novel fiber delay line buffering architecture for optical packet switching," Proceedings of IEEE Global Telecommunication Conference (GLOBECOM 2003), Vol. 5, pp. 2809-2813, December 2003

[30] Y. Liu, M. Hill, N. Calabretta, H. deWaardt, G. Khoe, and H.Dorren, "Three-state all-optical memory based on coupled ring lasers," IEEE Photonics Technology Letters, Vol. 15, No. 10, October 2003

[31] D. Liu, and M. Liu, "Differentiated services and scheduling scheme in optical burst-switched WDM networks," Proceedings of IEEE International Conference on Networks (ICON 2002), Singapore, pp. 23-27, August 2002

[32] X. Lu, and B. Mark, "Analytical modeling of optical burst switching with fiber delay lines," Proceedings of IEEE Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002), pp. 501-506, 2002

[33] X. Lu and B. Mark, "A new performance model of optical burst switching with fiber delay lines," Proceedings IEEE International Conference on Communications, Vo. 2, pp 1365-1369, May 2003

[34] Y. Luo, N. Ansari, "A computational model for estimating blocking probabilities of multifiber WDM optical networks," IEEE Communication Letters, Vol. 8 , No. 1 , pp. 60-62, January 2004

[35] F. Masatti, P. Morin, D. Chiaroni, and G. Loura, "Fiber delay lines optical buffer for ATM photonic switching applications," Proceedings of IEEE Information on Communications (INFOCOM 93), Vol. 3, pp. 935-942, 1993

[36] A. Monterosso, A. Pattavina, "Performance analysis of multistage interconnection networks with shared-buffered switching elements for ATM switching," Proceedings of IEEE Conferenc on Computer Communications, pp. 124-131, 1992

[37] C. Murthy and M. Gurusamy, *WDM Optical Networks: Concepts, Design, and Algorithms*, Prentice-Hall, Inc., New Jersey, 2002

[38] R. Ramaswami and K. Sivarajan, *Optical Networks*, Morgan Kaufmann Publishers, San Francisco, California, Second Edition, 2002

[39] Tancevski, et al. "Optical routing of asynchronous, variable length packets," IEEE Journal on Selected Areas in Communications, Vol. 18, No. 10, pp. 2084- 2093, October 2000

[40] J. Turner, "Queuing analysis of buffered switching networks," IEEE Transactions on Communications, Vol. 41, pp. 412-420, February 1993

[41] S. Verma, H. Chaskar, R. Ravikanth, "Optical burst switching: a viable solution for terabit IP backbone," IEEE Network, Vol. 14, No. 6, pp. 48-53, November 2000

[42] M. Yoo, C. Qiao, and S. Dixit, "Optical burst switching for service differentiation in the next-generation optical internet," IEEE Communication Magazine, Vol. 39, No. 2, pp. 98-104, February 2001

[43] A. Zalesky, H. Vu, M. Zukerman, Z. Rosberg, E. Wong, "Evaluation of limited wavelength conversion and deflection routing as methods to reduce blocking probability in optical burst switched networks," Proceedings of IEEE International Conference on Communications, Vol. 3 , pp. 1543-1547, June 2004

[44] T. Zhang, K. Lu, and J. Jue,"An Analytical Model for Shared Fiber Delay Line Buffers in Asynchronous Optical Packet and Burst Switches," to appear in the Proceedings of IEEE International Conference on Communications (ICC) 2005

[45] T. Zhang, K. Lu, and J. Jue, "Architectures and Performance of Fiber Delay Line Buffers in Packet-Based Multifiber Optical Networks," to appear in the Proceedings of Optical Fiber Communication Conference and Exposition (OFC) 2005

[46] X. Zhu, J. Kahn, "Queuing models of optical delay lines in synchronous and asynchronous optical packet-switching networks," Optical Engineering, Vol. 42, No. 6, pp. 1741-1748, June 2003

Appendix A

ANALYTICAL MODEL SOURCE CODE: FEED-FORWARD OPTICAL BUFFERS

The source code of the analytical model, which is written in Matlab, to evaluate the performance of optical switches with feed-forward optical buffers under different configurations and parameters is shown below. The code is used for both simple and enhanced forwarding algorithms. The probability of forwarding an incoming packet to the FDL (p) is derived via the technique of interaction using a for-loop to test if the incremented value of p satisfies the probability of forwarding an incoming packet to the FDL equations explained in Chapter 5. From this probability, the blocking probability of the switch is derived for both SFA and EFA algorithms.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%File name:
%- Analysis.m
%
%Purpose:
%- To identify the probability of blocking as well as the average delay for an optical
%switch with feed-forward optical buffers employing SFA and EFA.
%- To identify the probability of blocking for an optical switch with no FDL.
%
%Input parameters:
%- Load: the offered load at the output buffer, defined as Lambda/Mu
%- Mu: the rate at which the packets get transmitted into the buffer
%- Lambda: the service rate at which the packets get serviced at the buffer
%- D: the length of the FDL in terms of the mean transmission time of the packets
%- maxInter: The number of iterations takes place to identify the probability of forwarding
%a packet to the FDL.
%
%Output parameters:
%- PB1: the probability of blocking corresponding to the SFA case
%- Pd1: the average delay corresponding to the SFA case
%- PB2: the probability of blocking corresponding to the EFA case
%- Pd2: the average delay corresponding to the EFA case
```

```
%- PB3: the probability of blocking corresponding to no FDL case
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
clc
clear all

Load = 0.1:0.1:1; % The offered load at the output
maxIter = 10000; % The maximum iteration
mu = 0.0001;       % Packet transmission rate
D = 3/mu;            % FDL length

for j = 1:1:10

    lambda= Load(j) * mu;

    for i = 1 :1:maxIter    % For SFA
        p = i/maxIter;
        alfa = (((1-p)*lambda)/((1-p)*lambda+mu))+((p*lambda)/(p*lambda+mu));
        beta = (p*lambda)/(p*lambda+mu);
        gama = 1 - (mu/(mu+p*lambda))*exp(-p*lambda*D);
        given = 1 - (mu/(mu+lambda))*exp(-lambda*D);
        eta = given * alfa;
        aa = gama + alfa - eta;
        t=abs(aa-p);
        if (t<0.0001)
            break
        end
    end


    PB1(j) = beta; % Probability of Blocking (SFA)

    pd1(j) = (1-beta)*p; % Calculate the average delay

    for i = 1 :1:maxIter    % For EFA
        p = i/maxIter;
        alfa = (((1-p)*lambda)/((1-p)*lambda+mu))+((p*lambda)/(p*lambda+mu));
        beta = (p*lambda)/(p*lambda+mu);
        gama = 1 - (mu/(mu+p*lambda))*exp(-p*lambda*D);
        given = 1 - (mu/(mu+lambda))*exp(-lambda*D);
        eta = given * alfa;
        w = 1-eta;
        greaterThan = (p*lambda)/(mu+p*lambda);
        a = alfa+(1-alfa)*((gama)*(greaterThan));
        t=abs(a-p);
        if (t<0.0001)
            break
        end
    end


    PB2(j) = alfa * beta + (1-alfa)*beta*(greaterThan); % Probability of Blocking (EFA)

    pd2(j) = (1-beta)*p; % Calculate the average delay


    PB3(j) = lambda/(lambda+mu);    % Probability of Blocking (No FDL)
end
```

Appendix B

SIMULATOR SOURCE CODE: FEED-FORWARD OPTICAL BUFFERS

The source code of the simulator, which is written in C, used for evaluating the performance of optical switches with feed-forward optical buffers under different configurations and parameters is shown below. The code is used for both simple and enhanced forwarding algorithms. The configuration file that contains different parameters of the switch is used as an input file to the simulator and shown in the header file *Constants*. The main routines used in this simulator are shown in the header file *Functions*. Finally, definitions of different data structures used in the simulator is shown in the header file *Definitions*. To generate different random numbers, Multiplicative Linear Congruential generator is used in the simulator. The simulator was built in such a way that the switch performance is evaluated after the system reaches the steady state.

```
////////////////////////////////////////////////////////////////////////////////////////
// File Name:
// - Simulator.cpp
//
// Purpose:
// - To identify the probability of blocking for an optical switch with feed-forward optical
// buffers employing SFA and EFA.
//
// Input parameters:
// - Algorithm: To identify which algorithm is employed, set to SFA for simple forwarding
// algorithm or EFA to enhanced forwarding algorithm
// - Load: the offered load at the output buffer used by the Poisson generator
// - D: the propagation delay of FDL
// - PKTmean: the mean transmission time of a packet at the output buffer
//
// Output parameters:
// - Prob(blocking): probability of blocking =  //((double)DropPKTs/(double)ForwardedPKTs)
```

```
// - Beta: probability that port B is busy = //(((double)ForwardedToB)/(double)ForwardedPKTs)
// - Alfa = probability that port A is busy =
// (((double)ForwardedToA)/(double)ForwardedPKTs)
////////////////////////////////////////////////////////////////////////////////////////////

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include "Definitions.h"
#include "Constants.h"
#include "Functions.h"
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{

 int i;

 Algorithm = EFA; // either SFA (simple forwarding algorithm) or EFA (enhanced forwarding algorithm)
 Load = 0.1; // set the load at the output buffer
 D = 3*PKTmean; // set the propagation delay of FDL
 PKTmean=100000; // set the mean transmission time of a packet

 Initialization();
 BuildLL();

 for(i=0;i<TotalSIM;i++)
 {
 Current = PickEvent();

 if(i == (TotalSIM-1))
   EndTime = Current->time_to_be_processed;

// Each time an event is picket, it type is checked and then the event is forwarded to the
// corresponding routine.
// Event type are
// NEW_ARRIVAL: representing a newly arrival packet to be processed
// SET_A_BUSY: to set port A to a BUSY state
// SET_A_FREE: to set port A to a FREE state
// SET_B_BUSY: to set port B to a BUSY state
// SET_B_FREE: to set port B to a FREE state

 switch(Current->event_type)
    {
     case NEW_ARRIVAL :NEWARRIVAL(Current);
                       if(DEBUG)
                       {
                        PrintEvent(Current);
                        getchar();
                       }
                       free(Current);
                       break;
     case SET_A_BUSY  :SETABUSY(Current);
                       if(DEBUG)
                       {
                        PrintEvent(Current);
                        getchar();
                       }
                       free(Current);
                       break;
     case SET_A_FREE  :SETAFREE(Current);
                       if(DEBUG)
                       {
                        PrintEvent(Current);
                        getchar();
                       }
```

```
                    free(Current);
                    break;
    case SET_B_BUSY  :SETBBUSY(Current);
                    if(DEBUG)
                    {
                     PrintEvent(Current);
                     getchar();
                    }
                    free(Current);
                    break;
    case SET_B_FREE  :SETBFREE(Current);
                    if(DEBUG)
                    {
                     PrintEvent(Current);
                     getchar();
                    }
                    free(Current);
                    break;
   }
 }


 printf("- p = %f  q = %f Prob(blocking) = %f  Beta = %f \n\n- Lambda =
 %f Mu = %f => Load  = %f\n\n- Prob(A Busy) = %f  Prob(B Busy) = %f\n\n",
 (((double)ForwardedToB)/(double)ForwardedPKTs),(((double)ForwardedToA)
 /(double)ForwardedPKTs),((double)DropPKTs/(double)ForwardedPKTs),
 (1-(DFreeTime/(EndTime-BeginTime))),((ForwardedPKTs/(EndTime-BeginTime))),
  (1/PKTmean), (((ForwardedPKTs/(EndTime-BeginTime)))/(1/PKTmean)),
 (ABusyTime/(EndTime-BeginTime)), (BBusyTime/(EndTime-BeginTime)));
 if(DEBUG)
  printf("Total Packets = %ld ForwardedPKTs = %ld\nForwarded To A =
  %ld ForwardedToB = %ld\nDropPKTs = %ld  Lambda = %f\nFDLPKTs = %d
  Simulation Time = %f\n", TotalPKTs, ForwardedPKTs, ForwardedToA, ForwardedToB,
  DropPKTs, (ForwardedPKTs/(EndTime-BeginTime)), FDLhasPkt,(EndTime-BeginTime));
 printf("Done...");

return 0;
}


%%%%%%%%%%%%%%%%%%%%%%%%%%
Constant.h
%%%%%%%%%%%%%%%%%%%%%%%%%%

#define DEBUG 0
double Load;              /* the load at the output buffer*/
long TotalSIM=10000000; /* the total simulation time before checking for the steady state. */
double PKTmean; /* the packet mean transmission time */
struct Event * Head = NULL; /* the header of the liked list representing the sequence of */
/*the events to be processed */
struct Event * Current = NULL; /* the main structure of the even */
double IATmean=0; /* mean inter-arrival time between the packets*/
double D=0; /* the FDL length*/
int remainingTime=0; /* the remaining time for the packet to reach port A (F) of the FDL*/
int FDLhasPkt=0; /* A flag to identify if the FDL contains a packet. 0=NO, 1=Yes.
long DropPKTs=0; /* Number of packets get dropped*/
long ForwardedPKTs=0; /* Number of packets get forwarded to the FDL*/
long ForwardedToB=0; /* Number of packets get forwarded to port B of the FDL */
long ForwardedToA=0; /* Number of packets get forwarded to port A of the FDL */
long TotalPKTs=0; /* Total packets get processed*/
double BeginTime=0; /* The beginning time of the current cycle of the simulation*/
double EndTime=0; /* The end time of the current cycle of the simulation */
double DFreeTime=0; /* The total time the FDL was free of packets*/
double DBecomeFree=0; /* The beginning time at which the FDL is free*/
double BBusyTime=0; /* The total time port B was busy by a packet*/
double BBecomeBusy=0; /* The beginning time at which port B is busy by a packet*/
```

```
double ABusyTime=0; /*The total time port A was busy by a packet*/
double ABecomeBusy=0; /*The beginning time at which port B is busy by a packet*/
int SeedIAT = 7; /* the initial value of the seed for the inter-arrival time random generator*/
int SeedPktSize = 15; /* the initial value of the seed for the packet size random generator*/
long lastIssuedIAT = 0; /* the last value of the inter-arrival time generated*/
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
Definitions.h
%%%%%%%%%%%%%%%%%%%%%%%%%%%

enum YesNo {Yes, No}; // general Yes/No flag

enum SfaEfa {SFA, EFA}; // Flag to identify the used algorithm

enum SfaEfa Algorithm;

enum HighLow {High, Low}; // general high/low flag

enum Flag {FREE, BUSY}; // flag to identify the state of a port

enum Flag Astatus = FREE; // initial state of port A is set to FREE state

enum Flag Bstatus = FREE; // initial state of port B is set to FREE state

// definitions of different event types
enum EventType {NEW_ARRIVAL, SET_A_BUSY, SET_A_FREE, SET_B_BUSY, SET_B_FREE};

// Data structure of the Event that to be inserted into the linked list then later gets processed
typedef struct Event{
        enum EventType  event_type;
        enum HighLow    level;
        enum YesNo      CameFromB;
        double          arrival_time;
        double          time_to_be_processed;
        double          pkt_end_time;
        int             pkt_size;
        int             simultaneous_count;
        struct Event * next;
        struct Event * previous;
        struct Event * next_simultaneous;
        struct Event * previous_simultaneous;
        };

typedef struct Timing{
        enum YesNo FirstPkt;
        double last_arrival;
        double last_pkt_bit;
        double first_pkt_bit;
        int IATseed;
        int PKTseed;
        int Useed;
        };
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
Functions.h
%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
// Inter-arrival time random number generator
long GenerateIAT()
{
   long a;
   double u;
   long value_gen=0;

   a =abs( (16807 * (SeedIAT)) % (2147483647) );
   SeedIAT = a;
   u = ((double)(a) / 2147483647);
   value_gen =(int)( (- IATmean)*log(u) + 0.5);
   return value_gen;
}

// Packet size random number generator
long GeneratePktSize()
{
   long a;
   double u;
   long value_gen=0;

   a =abs( (16807 * (SeedPktSize)) % (2147483647) );
   SeedPktSize = a;
   u = ((double)(a) / 2147483647);
   value_gen =(int)( (- PKTmean)*log(u) + 0.5);
   return value_gen;
}

// Routine to generate a new packet, of type NEW_ARRIVAL, to inset
//into the liked list.
struct Event* GeneratePkt()
{
 long arrival=0;
 long size=0;
 double u=0;

 struct Event* tmp = (struct Event *)malloc(sizeof(struct Event));

 TotalPKTs++;

 size = GeneratePktSize();
 if( (InPort.FirstPkt) == No)
   {
     arrival = GenerateIAT();
     tmp->pkt_size = size;
     InPort.last_arrival = (InPort.last_arrival)+arrival;
     InPort.first_pkt_bit = InPort.last_arrival;
     InPort.last_pkt_bit=(InPort.first_pkt_bit)+ size;
   }
 tmp->pkt_size=size;
 tmp->event_type = NEW_ARRIVAL;
 tmp->CameFromB = No;
 tmp->arrival_time = InPort.first_pkt_bit;
 tmp->time_to_be_processed = InPort.first_pkt_bit;
 tmp->pkt_end_time = InPort.last_pkt_bit;
 tmp->simultaneous_count = 0;
 tmp->next = NULL;
 tmp->previous = NULL;
 tmp->next_simultaneous = NULL;
 tmp->previous_simultaneous = NULL;
 InPort.FirstPkt = No;

 return tmp;
}

// Routine to initialize different parameters
```

```
int Initialization()
{
 int i;


 IATmean = PKTmean/Load;

 FDLhasPkt = 0;
 ForwardedPKTs=0;
 ForwardedToB=0;
 ForwardedToA=0;
 Astatus = FREE;
 Bstatus = FREE;

 InPort.first_pkt_bit = GenerateIAT();
 InPort.last_pkt_bit=(InPort.first_pkt_bit)+GeneratePktSize();
 InPort.last_arrival = InPort.first_pkt_bit;
 InPort.FirstPkt = Yes;


 return 0;
}

// routine used to insert an event into the liked list
int InsertEvent(struct Event* ievent){

 struct Event* dmy1;
 struct Event* dmy2;

 dmy1 = Head;
 if(Head == NULL)
   {
    Head = ievent;
    return 1;
   }
 else
   {
    while((dmy1->time_to_be_processed) < (ievent->time_to_be_processed))
      {
       dmy2 = dmy1;
       dmy1 = dmy1->next;
       if(dmy1 == NULL)break;
      }
    if(dmy1 == NULL)
      {
       dmy2->next = ievent;
       ievent->previous = dmy2;
       return 1;
      }
    else
         if((dmy1 == Head) && ((dmy1->time_to_be_processed) >
         (ievent->time_to_be_processed)))
           {
            ievent->next = Head;
            Head->previous = ievent;
            Head = ievent;
            return 1;
           }
         else
             if((dmy1 == Head) && ((dmy1->time_to_be_processed) ==
             (ievent->time_to_be_processed)))
                 {
                   if(dmy1->next_simultaneous == NULL)
                     ievent->next_simultaneous = NULL;
                   else
                      {
```

```
                    ievent->next_simultaneous = dmy1->next_simultaneous;
                    dmy2 = ievent->next_simultaneous;
                    if(dmy2 != NULL)
                      dmy2->previous_simultaneous = ievent;
                   }
                 ievent->previous_simultaneous = dmy1;
                 dmy1->next_simultaneous = ievent;
                 dmy1->simultaneous_count++;
                 return 1;
               }
              else
                 if((dmy1 != NULL) && ((dmy1->time_to_be_processed) >
                 (ievent->time_to_be_processed)))
                   {
                     ievent->next = dmy1;
                     ievent->previous = dmy1->previous;
                     dmy1->previous = ievent;
                     dmy2 = ievent->previous;
                     if(dmy2 != NULL)
                        dmy2->next = ievent;
                     return 1;
                    }
                   else if((dmy1 != NULL) && ((dmy1->time_to_be_processed) ==
                   (ievent->time_to_be_processed)))
                          {
                  if(dmy1->next_simultaneous == NULL)
                              ievent->next_simultaneous = NULL;
                           else
                               {
                                 ievent->next_simultaneous = dmy1->next_simultaneous;
                                 dmy2 = ievent->next_simultaneous;
                                 if(dmy2 != NULL)
                                    dmy2->previous_simultaneous = ievent;
                                }
                           ievent->previous_simultaneous = dmy1;
                           dmy1->next_simultaneous = ievent;
                           dmy1->simultaneous_count++;
                           return 1;
                          }
   }
return 1;
}


// routine to initially build a liked list
int BuildLL(void)
{
 int  i;


   for(i=0;i<100;i++)
    {
    Current = GeneratePkt();
    if((InsertEvent(Current)) != 1)
      {
       printf("cannot insert the event into LL\n");
       getchar();
       exit(1);
  }
}



return 0;
}
```

```
// routine to search for a nearest packet in the FDL to port A (F)
double searchInEventList(void){

 struct Event* tmp1;
 struct Event* tmp2;

    tmp1 = Head;
if((tmp1->event_type == SET_A_BUSY) &&  (tmp1->CameFromB == Yes))
  {
return (tmp1->time_to_be_processed);
  }
else
  {
while(tmp1 != NULL)
  {
tmp2 = tmp1;
if(tmp1->simultaneous_count != 0)
  {
    while(tmp2->next_simultaneous != NULL)
      {
      tmp2 = tmp2->next_simultaneous;
    if((tmp2->event_type == SET_A_BUSY) &&  (tmp2->CameFromB == Yes))
      {
       return (tmp2->time_to_be_processed);
    break;
      }
      }
  }
else
  {
if((tmp1->event_type == SET_A_BUSY) &&  (tmp1->CameFromB == Yes))
  {
return (tmp1->time_to_be_processed);
break;
  }
  }
tmp1 = tmp1->next;
  }
  }
//return 0;
}

// routine used to decide if a packet can be inserted into the time gap between
//delayed packets
int canBeInserted(struct Event * tmpc)
{
remainingTime = (int)(searchInEventList() - (tmpc->time_to_be_processed));
if(remainingTime >= 0)
  {
if((tmpc->pkt_size) < (remainingTime))
  {
return 1;
  }
else
  {
return 0;
  }
  }
else
  {
printf("remainingTime is in negative = %d \n",remainingTime);
exit(1);
  }
}

// routine to process a newly arrival packet to the buffer
```

```
int NEWARRIVAL(struct Event * tmp)
{

 struct Event *dmy1;
 struct Event *dmy2;
 struct Event *dmy3;

 if(BeginTime == 0)
 {
  BeginTime = tmp->arrival_time;
  DBecomeFree = BeginTime;
 }

 ForwardedPKTs++;

 if (Algorithm == SFA)
   { // Begin: For SFA

 if(Astatus == FREE)
     {
  if(FDLhasPkt == 0)
       { // no pck exist in the FDL
 dmy1 = (struct Event *)malloc(sizeof(struct Event));
 dmy1->event_type = SET_A_BUSY;
 dmy1->CameFromB = No;
 dmy1->arrival_time = tmp->arrival_time;
 dmy1->time_to_be_processed = tmp->time_to_be_processed;
 dmy1->pkt_end_time = tmp->pkt_end_time;
 dmy1->pkt_size = tmp->pkt_size;
 dmy1->next = NULL;
 dmy1->previous = NULL;
 dmy1->simultaneous_count = 0;
 dmy1->next_simultaneous = NULL;
 dmy1->previous_simultaneous = NULL;
 InsertEvent(dmy1);
       }
  else
       { // there is at least a pck exist in the FDL
  if(Bstatus == FREE)
    {
  dmy1 = (struct Event *)malloc(sizeof(struct Event));
  dmy1->event_type = SET_B_BUSY;
  dmy1->CameFromB = No;
  dmy1->arrival_time = tmp->arrival_time;
  dmy1->time_to_be_processed = tmp->time_to_be_processed;
  dmy1->pkt_end_time = tmp->pkt_end_time;
  dmy1->pkt_size = tmp->pkt_size;
  dmy1->next = NULL;
  dmy1->previous = NULL;
  dmy1->simultaneous_count = 0;
  dmy1->next_simultaneous = NULL;
  dmy1->previous_simultaneous = NULL;
  InsertEvent(dmy1);

  dmy3 = (struct Event *)malloc(sizeof(struct Event));
  dmy3->event_type = SET_A_BUSY;
  dmy3->CameFromB = Yes;
  dmy3->arrival_time = tmp->arrival_time;
  dmy3->time_to_be_processed = ((tmp->time_to_be_processed)+D);
  dmy3->pkt_end_time = tmp->pkt_end_time;
  dmy3->pkt_size = tmp->pkt_size;
  dmy3->next = NULL;
  dmy3->previous = NULL;
  dmy3->simultaneous_count = 0;
  dmy3->next_simultaneous = NULL;
  dmy3->previous_simultaneous = NULL;
```

```
  InsertEvent(dmy3);
}
 else{
   DropPKTs++;
   if(DEBUG){
       printf("\nPacket %f has been dropped\n",tmp->arrival_time);
     }
   dmy1=GeneratePkt();
   InsertEvent(dmy1);
     }
   }
 }
   else if(Bstatus == FREE)
          {
            dmy1 = (struct Event *)malloc(sizeof(struct Event));
            dmy1->event_type = SET_B_BUSY;
            dmy1->CameFromB = No;
            dmy1->arrival_time = tmp->arrival_time;
            dmy1->time_to_be_processed = tmp->time_to_be_processed;
            dmy1->pkt_end_time = tmp->pkt_end_time;
            dmy1->pkt_size = tmp->pkt_size;
            dmy1->next = NULL;
            dmy1->previous = NULL;
            dmy1->simultaneous_count = 0;
            dmy1->next_simultaneous = NULL;
            dmy1->previous_simultaneous = NULL;
            InsertEvent(dmy1);

            dmy3 = (struct Event *)malloc(sizeof(struct Event));
            dmy3->event_type = SET_A_BUSY;
            dmy3->CameFromB = Yes;
            dmy3->arrival_time = tmp->arrival_time;
            dmy3->time_to_be_processed = ((tmp->time_to_be_processed)+D);
            dmy3->pkt_end_time = tmp->pkt_end_time;
            dmy3->pkt_size = tmp->pkt_size;
            dmy3->next = NULL;
            dmy3->previous = NULL;
            dmy3->simultaneous_count = 0;
            dmy3->next_simultaneous = NULL;
            dmy3->previous_simultaneous = NULL;
            InsertEvent(dmy3);
          }
         else
            {
              DropPKTs++;
              if(DEBUG)
    {
                    printf("\nPacket %f has been dropped\n",tmp->arrival_time);
                 }
              dmy1=GeneratePkt();
              InsertEvent(dmy1);
            }

   } //END: For SFA
 else if(Algorithm == EFA)
   { // Begin: For EFS
   if(Astatus == FREE)
     {
  if(FDLhasPkt == 0)
     { // no pck exist in the FDL
dmy1 = (struct Event *)malloc(sizeof(struct Event));
dmy1->event_type = SET_A_BUSY;
dmy1->CameFromB = No;
dmy1->arrival_time = tmp->arrival_time;
dmy1->time_to_be_processed = tmp->time_to_be_processed;
dmy1->pkt_end_time = tmp->pkt_end_time;
```

```
dmy1->pkt_size = tmp->pkt_size;
dmy1->next = NULL;
dmy1->previous = NULL;
dmy1->simultaneous_count = 0;
dmy1->next_simultaneous = NULL;
dmy1->previous_simultaneous = NULL;

InsertEvent(dmy1);

    }
  else
    { // there is at least a pck exist in the FDL
  if(canBeInserted(tmp) == 1)
    { // can be inserted in the idle time
   dmy1 = (struct Event *)malloc(sizeof(struct Event));
   dmy1->event_type = SET_A_BUSY;
   dmy1->CameFromB = No;
   dmy1->arrival_time = tmp->arrival_time;
   dmy1->time_to_be_processed = tmp->time_to_be_processed;
   dmy1->pkt_end_time = tmp->pkt_end_time;
   dmy1->pkt_size = tmp->pkt_size;
   dmy1->next = NULL;
   dmy1->previous = NULL;
   dmy1->simultaneous_count = 0;
   dmy1->next_simultaneous = NULL;
   dmy1->previous_simultaneous = NULL;
   InsertEvent(dmy1);
    }
  else
    { // canNot be inserted in the idle time
  if(Bstatus == FREE)
    {
  dmy1 = (struct Event *)malloc(sizeof(struct Event));
  dmy1->event_type = SET_B_BUSY;
  dmy1->CameFromB = No;
  dmy1->arrival_time = tmp->arrival_time;
  dmy1->time_to_be_processed = tmp->time_to_be_processed;
  dmy1->pkt_end_time = tmp->pkt_end_time;
  dmy1->pkt_size = tmp->pkt_size;
  dmy1->next = NULL;
  dmy1->previous = NULL;
  dmy1->simultaneous_count = 0;
  dmy1->next_simultaneous = NULL;
  dmy1->previous_simultaneous = NULL;
  InsertEvent(dmy1);

  dmy3 = (struct Event *)malloc(sizeof(struct Event));
  dmy3->event_type = SET_A_BUSY;
  dmy3->CameFromB = Yes;
  dmy3->arrival_time = tmp->arrival_time;
  dmy3->time_to_be_processed = ((tmp->time_to_be_processed)+D);
  dmy3->pkt_end_time = tmp->pkt_end_time;
  dmy3->pkt_size = tmp->pkt_size;
  dmy3->next = NULL;
  dmy3->previous = NULL;
  dmy3->simultaneous_count = 0;
  dmy3->next_simultaneous = NULL;
  dmy3->previous_simultaneous = NULL;
  InsertEvent(dmy3);
}
 else
    {
 DropPKTs++;
 if(DEBUG)
    {
      printf("\nPacket %f has been dropped\n",tmp->arrival_time);
```

```
   }
 dmy1=GeneratePkt();
 InsertEvent(dmy1);
   }
    }
 }
 }
   else if(Bstatus == FREE)
         {
           dmy1 = (struct Event *)malloc(sizeof(struct Event));
           dmy1->event_type = SET_B_BUSY;
           dmy1->CameFromB = No;
           dmy1->arrival_time = tmp->arrival_time;
           dmy1->time_to_be_processed = tmp->time_to_be_processed;
           dmy1->pkt_end_time = tmp->pkt_end_time;
           dmy1->pkt_size = tmp->pkt_size;
           dmy1->next = NULL;
           dmy1->previous = NULL;
           dmy1->simultaneous_count = 0;
           dmy1->next_simultaneous = NULL;
           dmy1->previous_simultaneous = NULL;
           InsertEvent(dmy1);

           dmy3 = (struct Event *)malloc(sizeof(struct Event));
           dmy3->event_type = SET_A_BUSY;
           dmy3->CameFromB = Yes;
           dmy3->arrival_time = tmp->arrival_time;
           dmy3->time_to_be_processed = ((tmp->time_to_be_processed)+D);
           dmy3->pkt_end_time = tmp->pkt_end_time;
           dmy3->pkt_size = tmp->pkt_size;
           dmy3->next = NULL;
           dmy3->previous = NULL;
           dmy3->simultaneous_count = 0;
           dmy3->next_simultaneous = NULL;
           dmy3->previous_simultaneous = NULL;
           InsertEvent(dmy3);
         }
        else
           {
             DropPKTs++;
             if(DEBUG)
    {
                  printf("\nPacket %f has been dropped\n",tmp->arrival_time);
                }
             dmy1=GeneratePkt();
             InsertEvent(dmy1);
           }
   } // END: for EFA

return 0;
}

// routine to process SET_A_BUSY event
int SETABUSY(struct Event * tmp)
{
 struct Event *dmy1;

      Astatus = BUSY;
  ABecomeBusy = tmp->time_to_be_processed;
      if(DEBUG)
      {
       printf("\nA is set BUSY for %f\n",tmp->arrival_time);
      }
      dmy1 = (struct Event *)malloc(sizeof(struct Event));
      dmy1->event_type = SET_A_FREE;
      dmy1->CameFromB = tmp->CameFromB;
```

```
      dmy1->arrival_time = tmp->arrival_time;
      if( (tmp->CameFromB) == No)
        {
        dmy1->time_to_be_processed = ((tmp->arrival_time)+(tmp->pkt_size));
        ForwardedToA++;
        }
  else{
        dmy1->time_to_be_processed = ((tmp->arrival_time)+D+(tmp->pkt_size));
    }
      dmy1->pkt_end_time = tmp->pkt_end_time;
      dmy1->pkt_size = tmp->pkt_size;
      dmy1->next = NULL;
      dmy1->previous = NULL;
      dmy1->simultaneous_count = 0;
      dmy1->next_simultaneous = NULL;
      dmy1->previous_simultaneous = NULL;

      InsertEvent(dmy1);

return 0;
}


// routine to process SET_B_BUSY event
int SETBBUSY(struct Event * tmp)
{

 struct Event *dmy1;

      ForwardedToB++;
      Bstatus = BUSY;
      BBecomeBusy = tmp->time_to_be_processed;
      if(FDLhasPkt == 0)
       DFreeTime = DFreeTime + ((tmp->time_to_be_processed)- DBecomeFree);
      FDLhasPkt++;
      if(DEBUG)
      {
       printf("\nB is set BUSY for %f\n",tmp->arrival_time);
       printf("\nD has %d packets\n\n",FDLhasPkt);
      }
      dmy1 = (struct Event *)malloc(sizeof(struct Event));
      dmy1->event_type = SET_B_FREE;
      dmy1->CameFromB = No;
      dmy1->arrival_time = tmp->arrival_time;
      dmy1->time_to_be_processed = ((tmp->arrival_time)+(tmp->pkt_size));
      dmy1->pkt_end_time = tmp->pkt_end_time;
      dmy1->pkt_size = tmp->pkt_size;
      dmy1->next = NULL;
      dmy1->previous = NULL;
      dmy1->simultaneous_count = 0;
      dmy1->next_simultaneous = NULL;
      dmy1->previous_simultaneous = NULL;

      InsertEvent(dmy1);

return 0;
}

// routine to process SET_A_FREE event
int SETAFREE(struct Event * tmp)
{

 struct Event *dmy1;

 Astatus = FREE;
```

```
 ABusyTime = ABusyTime + ((tmp->time_to_be_processed)- ABecomeBusy);

 if((tmp->CameFromB) == Yes)
 {
   FDLhasPkt--;
   if(FDLhasPkt == 0)
     DBecomeFree = tmp->time_to_be_processed;
 }
 if(DEBUG)
 {
  printf("\nA is set FREE for %f\n",tmp->arrival_time);
  printf("\nD has %d packets\n\n",FDLhasPkt);
 }
 dmy1=GeneratePkt();
 InsertEvent(dmy1);
return 0;
}



int SETBFREE(struct Event * tmp)
{
 Bstatus = FREE;

 BBusyTime = BBusyTime + ((tmp->time_to_be_processed)- BBecomeBusy);

 if(DEBUG)
 {
  printf("\nB is set FREE for %f\n",tmp->arrival_time);
 }

return 0;
}

// routine used to print the content of an event
void PrintEvent(struct Event * tmp)
{
  if(tmp == NULL)
   printf("NULL\n");
  else{
       switch(tmp->event_type){
        case NEW_ARRIVAL    :printf("event_type: NEW_ARRIVAL\n");
                             break;
        case SET_A_BUSY :printf("event_type: SET_A_BUSY\n");
                     break;
        case SET_A_FREE :printf("event_type: SET_A_FREE\n");
                     break;
        case SET_B_BUSY :printf("event_type: SET_B_BUSY\n");
                      break;
        case SET_B_FREE :printf("event_type: SET_B_FREE\n");
                             break;
                             }
       if(tmp->CameFromB == Yes)
         printf("It came from port B\n");
       printf("arrival_time: %f\n",(tmp->arrival_time));
       printf("time_to_be_processed: %f\n",(tmp->time_to_be_processed));
       printf("pkt_end_time: %f\n",(tmp->pkt_end_time));
       printf("pkt_size: %d\n",(tmp->pkt_size));
       printf("simultanous_count: %d\n",(tmp->simultaneous_count));
      }
}



// routine to pick the soonest event to get processed from the linked list
```

```
struct Event* PickEvent(void){

 struct Event* tmp1;
 struct Event* tmp2;
 struct Event* tmp3;
 struct Event* picked;

 tmp1 = Head;
 picked = tmp1;
 if(tmp1 != NULL)
   {
    if(tmp1->simultaneous_count != 0)
      {
        Head = picked->next_simultaneous;
        Head->simultaneous_count = picked->simultaneous_count;
    (Head->simultaneous_count)--;
    if(picked->next == NULL)
          {
       picked->next_simultaneous = NULL;
       Head->previous_simultaneous = NULL;
       picked->simultaneous_count = 0;
   return picked;
          }
         else
            {
             tmp1 = picked->next;
             if(tmp1 != NULL)
               tmp1->previous = Head;
     Head->next = picked->next;
     picked->next = NULL;
     picked->next_simultaneous = NULL;
     Head->previous_simultaneous = NULL;
              picked->simultaneous_count = 0;
     return picked;
            }
      }
      else
         {
           Head = Head->next;
       if(Head != NULL)
              Head->previous = NULL;
         picked->next = NULL;
           picked->simultaneous_count = 0;
      return picked;
        }

   }
   else
      return NULL;

}
```

Appendix C

ANALYTICAL MODEL SOURCE CODE: FEEDBACK OPTICAL BUFFERS

The Maple source code of the analytical model used to evaluate the performance of optical switches with feedback optical buffers under different configurations and parameters is shown below. The probability distribution of the number of packets getting forwarded to the FDL is derived using matrix inverse transform internally built in Maple using the *Solve* command. From this distribution, the blocking probability, delay distribution, and FDL utilization is derived.

```
###############################################################################################
# File Name:
# - Analysis.mws
#
# Purpose:
# - To identify the probability of blocking, delay distribution, and FDL utilization for an
# optical switch with feedback optical buffer.
#
# Input parameters:
# - rho: the offered load at each input to the switch
# - K: number of shared FDL
# - N: number of input/output ports of the switch
#
# Output parameters:
# - Edrop/Earrive: the probability of blocking
# - recirculates: the probability of a packet re-circulate the switch
# - Delay: the probability distribution of the number of re-circulations a packet encounters
# in the switch
# - meanDelay: the average delay
# - TTL: the delay most of the packets encounters in the switch
# - NormalizedFDLUtilization: the FDL utilization
###############################################################################################

 restart:
 rho:=0.1:
 K :=8:
 N :=64:

# Define the probability distribution of the number of packet
# arrival at the primary input
```

```
Y := (i) -> binomial(N,i)*(rho^i)*((1-rho)^(N-i)):

# Define the probability of forwarding T packets to FDLs
# given Z arrivals using Surjective Mapping technique
TgivenZ := (t,z) -> (binomial(N,z-t)*(sum('((-1)^ii)*(binomial(z-t,ii))*((z-t-ii)^z)',

# Define the probability distribution of the number of packet
# arrival at the secondary input
'ii'=0..(z-t))))/(N^z):
for iij from 0 by 1 to K do
    XX := (iij) -> if (iij < K) then T(iij);
   else sum('T(j)','j'=K..(N+K-1));
end if;
end do:
for ii from 0 by 1 to K do
 XX(ii);
 X(ii):= %;
end do:

# define the probability distribution of the number of packet
# arrival at the output of the switch
for w from 0 by 1 to (N+K) do
    ZZ := (w) -> if (w <= K) then sum('Y(x)*X(w-x)','x'=0..w);
                  elif (w <= N) then sum('Y(w-K+xx)*X(K-xx)','xx'=0..K);
                  else sum('Y(w-K+xxx)*X(K-xxx)','xxx'=0..(K-(w-N)));

end if;
end do:
for iijj from 0 by 1 to (N+K) do
 ZZ(iijj);
 Z(iijj):= %;
end do:

# Evaluating the distribution of the number of packet arrival
# at the FDLs
for t from 0 by 1 to (N+K-1) do
   # the case where T = 0
   Tmp:= (t) -> if (t = 0) then (Z(0)+sum('TgivenZ(t,v)*Z(v)','v'=(t+1)..(N+t)));
   # the case where 0<T<=K
elif (t <= K) then sum('TgivenZ(t,q)*Z(q)','q'=(t+1)..(N+t));
   # the case where T>K
else sum('TgivenZ(t,qq)*Z(qq)','qq'=(t+1)..(N+K));
end if;
end do:
for iii from 0 by 1 to (N+K-1) do
 Tmp(iii)-T(iii);
assign(TT(iii),%);
# TT is the matrix representing linear equations
# each of which is the probability of T
end do:
# prepare the Linear equation of T in a list form
# so that it can be solved using SOLVE command
sum('T(i)','i'=0..(N+K-1))=1:
assign(Total,%);
eq[0]:={TT(0)}:
for i from 1 by 1 to (K+N-2) do
eq[i] := eq[i-1] union {TT(i)}
end do:
eq[(K+N-1)] := eq[(K+N-2)] union {Total}:
assign(Equations,eq[K+N-1]);
var[0]:={T(0)}:
for i from 1 by 1 to (K+N-1) do
var[i] := var[i-1] union {T(i)}
end do:
assign(Variables,var[K+N-1]);
s:=solve(Equations,Variables): # Using SOLVE command to solve the linear equations of T
```

```
 assign(s);

# Expected number of packet drop
 Edrop :=sum('(iEdrop-K)*T(iEdrop)','iEdrop'=(K+1)..(N+K-1)):

# Expected number of packet arrival at the switch
 Earrive:=N*rho:
 Edrop/Earrive;

# The probability of a packet re-circulating in the switch
 a1:=sum('sum('(t/z)*TgivenZ(t,z)*value(Z(z))','z'=(t+1)..(N+K))','t'=1..K):
 a2:=sum('sum('(t/z)*(K/t)*TgivenZ(t,z)*value(Z(z))','z'=(t+1)..(N+K))',
 't'=(K+1)..(N+K-1)):
 recirculates := a1+a2; # = Xi

# the probability of a packet leave the switch
 leaves:=sum('sum('((z-t)/z)*TgivenZ(t,z)*value(Z(z))','z'=(t+1)..(N+K))',
 't'=0..(N+K-1)):

# the probability of a packet dropped at the switch
 dropped:=sum('sum('(t/z)*((t-K)/t)*TgivenZ(t,z)*value(Z(z))','z'=(t+1)..(N+K))',
 't'=(K+1)..(N+K-1)):

# the probability distribution of the number of re-circulation
# a packet encounters in the switch
 Recirculation := (i) ->(recirculates^i)*(1-recirculates):

# the probability that a packet eventually leave the switch
 PrEvetuallyLeave := sum('(recirculates^j)*leaves','j'=0..infinity):

# the delay distribution in the switch in terms of the number
# of re-circulations
 Delay := (i) -> ((recirculates^i)*leaves)/PrEvetuallyLeave:
 # the same as Recirculation(i)
 meanDelay:=sum('i*Delay(i)','i'=0..infinity);

# the delay that most of the packet encounters in the switch
 epsilon:=0.0001:
 TTL:=ceil((ln(epsilon)/ln(recirculates))-1);

 # the FDL utilization
 NormalizedFDLUtilization:=(sum('i*T(i)','i'=0..K)+sum('K*T(i)',
 'i'=(K+1)..(N+K-1)))/K;
```

Appendix D

SIMULATOR SOURCE CODE: FEEDBACK OPTICAL BUFFERS

The source code of the simulator used in this research is shown below. It was written in Matlab and then was used to evaluate the performance of optical switches with feedback under different configurations and parameters. The code is based on a loop, where each cycle of the loop represents a new time slot. To generate different random numbers, Multiplicative Linear Congruential generator is used in the simulator. The simulator was built in such a way that the switch performance is evaluated after the system reaches the steady state.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File name:
% - Simulation.m
%
% Purpose:
% - To identify the probability of blocking as well as the average delay for an optical
%  switch with feedback optical buffer.
%
%
% Input parameters:
% - D: the length of D in terms of time slot
% - K: number of FDLs,
% - N: of input/output ports
% - rho: the load at each of the input port
%
% Output parameters:
% - tblocked/(tblocked+tpassed): the blocking probability of the switch
% - Histo: the delay distribution of the switch
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% length(nonzeros(x))
clc
clear all
% Parameters initialization
totalProcessedPKT = 10000; % total number of packets get processed before checking for the
%steady state
maxLoad = 1;  the maximum considered load
first = 999; %to identify the first cycle of the code for steady state calculation
next = 0; %the second cycle of the code to compare its performance with the first cycle for
%the steady state calculation
```

```
steady = 0; %flag to determine if the steady state has been reached
CurrentTimeSlot = 0; # the current time slot


Load = 0.1:0.1:maxLoad;

D = 1; % # of time slot, in terms of pkt size, the length of D is
K = 8;  % # of FDLs,
N = 64;  % # of input/output ports
rho = 0.1; % the load at each of the input port


PKTFromFDLTimeSlot = zeros(K,D); % reset the array representing the packets coming from the FDL

ArrivalTimes = zeros(N,N+K); % reset the array used for the delay calculation


while steady == 0 % begin: To calculate the steady state

    passed = zeros(1,N); % reset the array representing the number of packets left the switch
    blocked = zeros(1,N); % reset the array representing the number of packets get dropped
    HistoGram = zeros(10000,D); % reset the array representing the delay distribution inside
    %the switch
    z=0;
    recir = 0;
    processedPKT = 0;


    while processedPKT <= totalProcessedPKT


        CurrentTimeSlot = CurrentTimeSlot +1;
        outputHasAlreadyPacket = zeros(1,N);



        count = zeros(1,N);             %
        #s of packets destained to each output are set to 0's


        if D > 1
            for FDL = 1:1:K
                for j = 1:1:(D-1)
                    PKTFromFDLTimeSlot(FDL,j) = PKTFromFDLTimeSlot(FDL,j+1);
                     % Left shift
                end
                PKTFromFDLTimeSlot(FDL,D) = 0;
            end
        end

        for FDL = 1:1:K
      % Check the packet at the input of each FDL to see where, if  any, it destaines to
            if(PKTFromFDLTimeSlot(FDL,1)  > 0)
                count(PKTFromFDLTimeSlot(FDL,1)) = count(PKTFromFDLTimeSlot(FDL,1)) +1;
            end
            if D == 1
                PKTFromFDLTimeSlot(FDL,D) = 0;
            end
        end

        for input = 1:1:N
        % Check the input of 1:N
            x = rand(1);
            if(x <= rho)
```

```
    % There is a packet at that input at this time slot
        destination = mod((ceil(rand(1)*100000)),N)+1;
        % and that packet intended to "destination" output
        count(destination) = count(destination) +1;
        processedPKT = processedPKT + 1;
        ArrivalTimes(destination,
        length(nonzeros(ArrivalTimes(destination,:)))+1) =
        CurrentTimeSlot;
    end
end

z= z + sum(count);


for FDL = 1:1:K


    % pick one each of the outputs randomly to start
    % counting the pkts intended to that output
    % then forward a pkt to that output, then to the
    % FDL if any, then drop  the rest
    checked = zeros(1,N);
    countChecked = 0;
    while(countChecked < N)

        out2 = 0;
        while(out2 == 0)
            yy = mod((ceil(rand(1)*1000)),N)+1; % Pick an output randomly
            if(checked(yy)==0)
            % Check if it has been checked previously
                checked(yy) = 1;
                out2 = 1;
            end
        end
        countChecked = countChecked +1;
        output = yy;

        % One packet is destained to that output while
        %the output has not received ant pkt yet %
        if((count(output) ==1) &&  (outputHasAlreadyPacket(output) == 0))
            passed(output) = passed(output) + 1;
            % Only one will leave the node
            count(output) = count(output) - 1;
            % Subtract one packet from the packets intended to that output port
            outputHasAlreadyPacket(output) = 1;
            % Drop one of the packets addressed to that output
            % randomly. Calculate the daly that packet encounters
            % by subtracting the time it enters the switch from the
            % current simulation time and insert that into the
            % histogram array. Note that the first element of the
            % histogram HistoGram(1) corresponds to the number of
            % times packets encoutered zero circulations, while
            % HistoGram(2) corresponds to the number of
            % times packets encoutered one circulations and so on.
            nzmat = nonzeros(ArrivalTimes(output,:));
            Len = length(nzmat);
            if (Len > 0)
                tmp1 = mod((ceil(rand(1)*100000)),
                length(nonzeros(ArrivalTimes(output,:))))+1;
                HistoGram(CurrentTimeSlot-ArrivalTimes(output,tmp1)+1) =
                HistoGram(CurrentTimeSlot-ArrivalTimes(output,tmp1)+1)+1;
                nzmat(tmp1) = 0;
                nzmat = nonzeros(nzmat);
                for j = 1:1:Len-1
                    ArrivalTimes(output,j) = nzmat(j);
                end
```

```
            for i = (Len):1:N+K
                ArrivalTimes(output,i) = 0;
            end
        end
    end

    % One packet is destained to that output while the output has
    %received a pkt previously and the FDL into consideration is FREE %
    if ((count(output) ==1) &&  (outputHasAlreadyPacket(output) == 1) &&
    (PKTFromFDLTimeSlot(FDL,D) == 0))
        PKTFromFDLTimeSlot(FDL,D) = output;     % Forward one packet, from
        %the rest of the packets intended to that output port, to the FDL(i)
        count(output) = count(output) - 1;  % Subtract one packet from the
        %rest of the packets intended to that output port
        recir = recir + 1;
    end

    % More than one packet is destained to that output while the output
    % has not received ant pkt yet %
    if((count(output) >=2) &&  (outputHasAlreadyPacket(output) == 0))
        passed(output) = passed(output) + 1;    % One will leave the node
        count(output) = count(output) - 1;      % Subtract one packet from
        %the packets intended to that output port
        outputHasAlreadyPacket(output) = 1;
        nzmat = nonzeros(ArrivalTimes(output,:));
        Len = length(nzmat);
        if (Len > 0)
            tmp1 = mod((ceil(rand(1)*100000)),
            length(nonzeros(ArrivalTimes(output,:))))+1;
            HistoGram(CurrentTimeSlot-ArrivalTimes(output,tmp1)+1) =
            HistoGram(CurrentTimeSlot-ArrivalTimes(output,tmp1)+1)+1;
            nzmat(tmp1) = 0;
            nzmat = nonzeros(nzmat);
            for j = 1:1:Len-1
                ArrivalTimes(output,j) = nzmat(j);
            end
            for i = (Len):1:N+K
                ArrivalTimes(output,i) = 0;
            end
        end
        if (PKTFromFDLTimeSlot(FDL,D) == 0)
            PKTFromFDLTimeSlot(FDL,D) = output;
            % Forward one packet, from the rest of the packets
            %intended to that output port, to the FDL(i)
            count(output) = count(output) - 1;
            % Subtract one packet from the rest of the
            %packets intended to that output port
            recir = recir + 1;
        end
    end

    % More that one packet is destained to that output while
    %the output has received a pkt previously and the FDL into
    %consideration is FREE %
    if ((count(output) >=2) &&  (outputHasAlreadyPacket(output) == 1)
    && (PKTFromFDLTimeSlot(FDL,D) == 0))
        PKTFromFDLTimeSlot(FDL,D) = output;     % Forward one packet, from
        %the rest of the packets intended to that output port, to the FDL(i)
        count(output) = count(output) - 1;  % Subtract one packet from the rest
        %of the packets intended to that output port
        recir = recir + 1;
    end

end
```

```
        end


        for output = 1:1:N      % Start Calculating # of packets blocked at each output
        %which equals what is left after forwarding to the input
            blocked(output) = blocked(output) + count(output);
            if(count(output) >= 1)
                for i=1:1:count(output)
                    nzmat = nonzeros(ArrivalTimes(output,:));
                    Len = length(nzmat);
                    if (Len > 0)
                        tmp1 = mod((ceil(rand(1)*100000)),
                        length(nonzeros(ArrivalTimes(output,:))))+1;
                        nzmat(tmp1) = 0;
                        nzmat = nonzeros(nzmat);
                        for j = 1:1:Len-1
                            ArrivalTimes(output,j) = nzmat(j);
                        end
                        for m = (Len):1:N+K
                            ArrivalTimes(output,m) = 0;
                        end
                    end
                end
            end
        end



    end

    tblocked = 0;
    tpassed = 0;
    for output = 1:1:N
        tblocked = tblocked + blocked(output);      % Calculate the total blocked
        tpassed = tpassed + passed(output);         % Calculate the total passed
    end

    tblocked
    tpassed
    processedPKT
    break;



    if first == 999
        first = tblocked/(tblocked+tpassed)
    else
        next = tblocked/(tblocked+tpassed)
        if abs(next - first) < 0.01
            steady = 1;
        else
            first = next
        end
    end



end % end: To calculate the steady state


% Histo contains the distribution of the Delay in terms of the number of
% recirculation
Histo = nonzeros(HistoGram);
Histo = Histo/sum(Histo);
```

```
x=1:1:length(Histo);

rho
tblocked/(tblocked+tpassed)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```