INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

PERFORMANCE MODELLING OF THE TERABIT FREE SPACE OPTICAL BACKPLANE

Ka Veng Ho

Department of Computer Science
McGill University, Montréal

April 1997

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements of the degree of
Master of Science in Computer Science

ABSTRACT

The Canadian Institute for Telecommunication Research (CITR) has undertaken a five year "Major Project" in Optical Systems and Devices. As part of this project, researchers in the Microelectronics and Computer System (MACS) Laboratory at McGill University are developing a reconfigurable free-space optical backplane architecture capable of supporting terabits per second (Tbps) throughput [21]. The optical backplane can be dynamically reconfigured to support the switching schemes used in both multiprocessor systems and in telecommunication systems. In this thesis, we will consider the performance of the optical backplane when it is configured to support a 160 Gigabit per second (and a 640 Gigabit per second) ATM switch in a standard telecommunication environment.

The optical backplane with 160..640 Gigabits per second throughput will have a far higher throughput than any ATM switching system yet developed. This throughput advantage will be negated if severe congestion within the backplane degrades performance. Hence, it is very important to consider queueing schemes which control congestion within the backplane, and to build an accurate simulator model to predict the behavior of the backplane before the actual implementation takes place. In this thesis, we assume that the backplane is configured to support a traditional 3-stage crossbar switching system, with electrical switches in the first and third stages, and optical switches in the second stage. A discrete event simulation model is developed to analyze numerous architectural variations of this opto-electronic switching system.

Through simulations, we illustrate the tradeoffs between the complexity of the optical switches in the second stage and the performance of the system, measured in terms of the throughput, delay and loss rate. In particular, we illustrate that large optical switches in the second stage yield the highest performance, at the expense of relatively complex logic in the opto-electronic integrated circuits ("smart pixel arrays"). These relationships form a guideline which can be used for designing the smart pixel arrays for the optical backplane.

RÉSUMÉ

L' Institut Canadien de Recherches en Télécommunication (CITR) a entrepris un "important projet" de cing ans dans le domaine de Systèmes et d'Appareils Optiques. Prenant part à ce projet, des chercheurs en Micro-Electronique et du Laboratoire de Systèmes d' Informatiques (MACS) de l'Université de McGill ont developpé une architecture de système optique d'interconnexions librement espacées reconfigurable pouvant supporter Terabits par seconde (Tb/s) de transfert [**21**]. Le système optique d'interconnexions peut être dynamiquement reconfiguré pour supporter les schémas de switch utilisés pour les deux systèmes de multiprocesseurs et de télécommunication. Dans cette thèse, nous allons considérer la performance du système optique d'interconnexions quand il est configuré pour supporter un ATM switch de 160 Gigabits par secondes (et de 640 Gigabits par secondes) dans un environnement de télécommunication standard.

Le système d'interconnexions avec 160..640 Gigabits par seconde de tranfert permettra un plus grand transfert que le système de switch ATM déjà développé. L'avantage de ce transfert sera moindre si plusieurs congestions dans le système d' interconnexions détériorent la performance. Donc, il est très important de considérer les schémas de la file d'attente qui contrôlent la congestion dans le système d'interonnexions et de construire un modèle de simulateur précis pour prévoir les réactions du système d'interconnexions avant que l'implémentation soit faite. Dans cette thèse, nous assumons que le système d'interconnexions est configuré pour supporter un système de switch de bar croisé traditionnel à 3-phases avec des switchs électriques au première et au troisième phase et des switchs optiques dans la seconde phase. Un modèle de simulation d'évènement discret est développé pour analyser plusieurs variations d'architectures de ce système de switch Opto-Electronique.

A travers des simulations, nous illustrons les avantages et les désavantages entre la complexité des switchs optiques dans la seconde phase et la performance du système mesurée en terme de transfert, délai et taux de perte. En particulier, nous illustrons que ces grands

switchs optiques dans la seconde phase apportent une très grande performance en dépit de la logique relativement complexe dans les circuits intégrés d'Opto-Electronique. "Tableaux de pixel intelligent" ces relations forment une règle de base qui peut être utilisée pour concevoir des tableaux de pixel intelligent pour le système d'interconnexions optiques.

ACKNOWLEDGEMENTS

I wish to thank my supervisor, Professor Ted Szymanski, of the Department of Electrical Engineering for introducing me to research in general, and to network simulation in particular. His enthusiasm and encouragement, which have made this thesis possible, is most appreciated.

I also wish to thank my co-supervisor, Professor David Avis, for his help and suggestions whenever I needed them.

Thanks are due to my colleague, Boonchuay Supmonchai, who spent countless hours to answer my questions whenever I ran into difficulties. Thanks to Thomas Obenaus, Sherif Sherif, Michael Kim, and Palash Desai who warmly welcomed me to the lab and showed me its social side when I started in the summer of 1994. The friendships I formed made my stay in the lab an enjoyable one.

I also thank Pung Hay Chitra for helping to translate the abstract into French.

Finally, I thank my dear friend, Owen Cheung, for his companionship over the last two years, and my family for their constant support, encouragement, and guidance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

The Canadian Institute for Telecommunications Research (CITR) [3] has undertaken a large project to develop a free-space optical backplane capable of supporting terabits per second (Tbps) bandwidth. The backplane is dynamically reconfigurable and can be used for parallel computing and communication applications.

To develop an efficient optical backplane, it is important to build an accurate model to predict its behavior before actual implementation takes place. Analytic models are accurate if they are built correctly; however, it is often quite cumbersome to develop such models due to the complexity involved. A good alternative is to simulate the model via computer programs and observe the statistical behavior of the system. Simulations, therefore, are able to give quick results and allow easy modification and generalization of the assumptions. Moreover, they can be used to validate the results of analytical network models.

The goal of this thesis is to describe a software tool that performs simulations of the free-space optical backplane. By studying the statistical behavior of the backplane, an efficient network configuration can be found, given the unique hardware constraints (e.g. size of switch and size of buffer).

1.1. Switches and Queues

A *switching network* usually consists of *switches, queues,* and *links.* An $N \times M$ *switch* (or *switch element*) is a device that has N inlets (or input links) and M outlets (or output links). *Inlets* are communication channels (e.g., wires or radio transmitters) for a switch to receive information from other devices (or simply from other switches). *Outlets* are communication channels for a switch to transmit information to other devices. The function of a *switch* is to "switch" messages from an inlet (out of N inlets) to an outlet (out of M outlets). A message is represented by long strings of bits and is broken into shorter bit strings called *packets.* A graphical representation of a switch is shown in Figure 1.1(a).

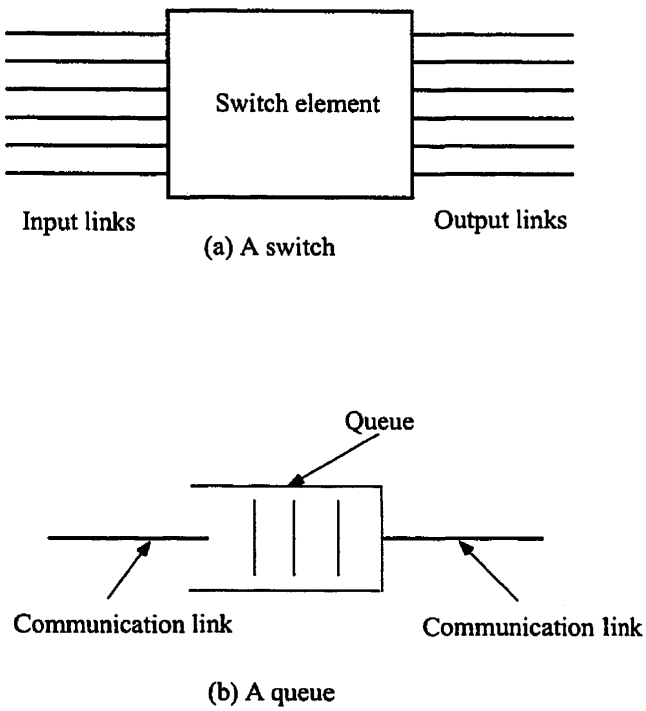


FIGURE 1.1. Graphical representations of: (a) a switch, and (b) a queue.

Queues are buffers that store packets. The well known *"First In First Out"* (FIFO) queue is one example. Packets in a FIFO queue are transmitted in the order that they arrive at the queue. A graphical representation of a queue is shown in Figure 1.1(b).

1.2. Multistage Interconnection Switching System

The networks which we consider in this thesis are constructed by interconnecting switch elements. The function of such a network is to transmit packets from input links of the source switches to output links of the destination switches, by traversing appropriate subsets of switch elements within the network. Furthermore, switch elements in a network can be organized into *stages*, in which the interconnections of the switch elements are restricted to switch elements in adjacent stages. Networks with switches organized into stages are known

as *multistage switching networks* (see Figure 1.2) [**4, 9**]. In this thesis, the convention used for labeling switches is:

$$S_{\text{stage \#,switch \#}}$$

For example, $S_{2,1}$ refers to the second switch in stage 2.

In this thesis, we assume that the backplane is configured to support a *multistage switching network*. However, we would like to point out that there are many other possible variations that can be also supported by the backplane.

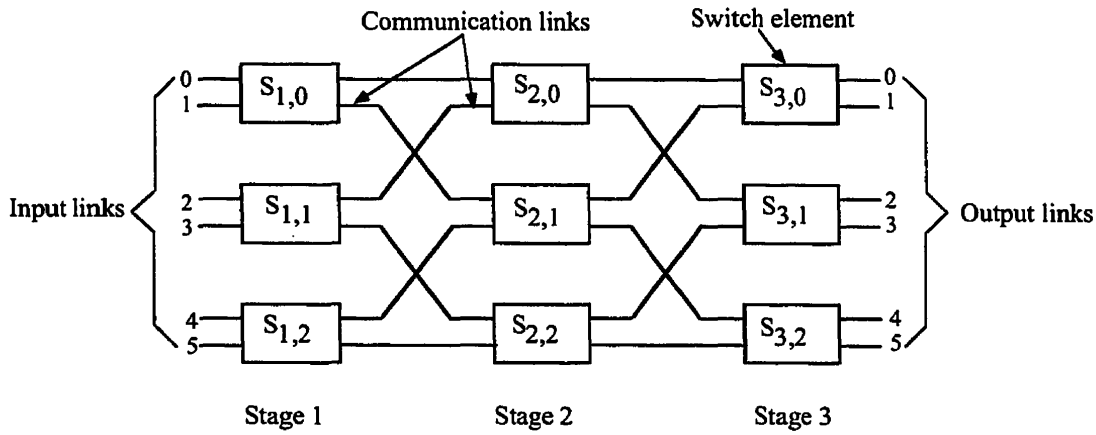


FIGURE 1.2. Graphical representation of a multistage switching network.

1.3. Author's Contribution

This thesis has three main contributions. The first is to specify a simulator model which could analyze numerous architectural variations of the switching system. The second contribution is in the implementation of the simulator model as a computer program that is capable of performing simulations with various configurations. In order to allow easy revision and expansion of the software, much effort was devoted to create a well structured and documented simulator. The third contribution is the experiments performed with this software. With the knowledge gained from the experiments, this thesis illustrates the tradeoffs between the complexity of optical switches and the performance of the backplane, measured in terms of throughput, delay and loss rate. These relationships form a guideline which can be used for designing a cost-efficient optical backplane.

1.4. Network Simulation and Thesis Overview

Network Simulation, as stated by Kheir in [7], involves the construction of a network's mathematical and logical model, followed by experimenting with the model on a computer. Therefore, our simulation study is organized into three phases: (1) system specification, (2) model development, and (3) experimentation. The discussion of the simulation study begins in Chapter 2, which presents an introduction to the optical backplane and other related topics that are intended to give the readers an adequate background in order to understand the material in the rest of this thesis.

Chapter 3 gives a detailed discussion of the multistage switching system to be modelled. Although the backplane can support many network configurations, we will only consider multistage networks in this thesis. The chapter begins with the description of how optical switches in the backplane and the electrical switches that are connected to the optical switches can be viewed as a multistage switching system [18]. Following this, an architecture for the system is described and its operation is revealed. In addition, two alternative configurations that employ different sizes and number of optical switch(es) are given at the end of the chapter.

Given the specification of the multistage switching system, the next objective is to describe how a model is constructed for the system. Chapter 4 documents how the essential features of the system are captured in a computer program. The software development model and the validation criteria used for building the network model are also explained.

Having developed the software, Chapter 5 presents the numerical results of the experimentation. A thorough analysis and comparison of different configurations of the multistage network are presented. Moreover, the tradeoffs between the complexity of the optical switches and the performance of the system, measured in terms of the throughput, delay, and loss rate, are illustrated.

Finally, Chapter 6 draws the conclusion of the work, and some additional future work that could possibly arise from this research are discussed.

CHAPTER 2

Literature Review

2.1. Computer Simulation

Computer simulation involves the construction of a mathematical and logical model, followed by experimentation of the model on a computer [**7, 11, 12, 14**]. A *model* is an object that captures the essential features of the *system* under investigation [**7**]. The model must contain some representation of the *entities* in the *system*, and be able to reflect the *activities* in which these *entities* engage.

A *system* contains objects, called *entities*; a *network system* is a system that contains entities of a network. For example, the entities of a *network system* may include packets, switches, links and queues. Each entity has properties or attributes. The attributes of a switch include, for example, the number of input and output links (dimension of the switch), and the number of queues.

A simulation model describes the functionality of a system. The functionality is executed by processes called *activities*. Adding a packet to a queue and selecting a path for a packet are examples of activities in a network system.

In this thesis, the network system under consideration is an *open* system. Systems with activities in the environment that affect the system are considered as *open* [**14**]. For example, a network system is said to be an *open* system if packets are injected into the system at the input and are removed from the system at the output once they arrive at their destinations.

The simulation model in this thesis is also a *discrete* system model. A system model is said to be *discrete* when activities in the system only occur at fixed time intervals. Simulations conducted with discrete system models are, therefore, *discrete event simulations* [**14**].

Finally, the simulation model is interpreted as a computer program. This computer program resembles closely the behavior of the system and serves the following purposes:

(a) to collect data and information about various quantities of interest, and

(b) to obtain good estimates of the desired performance measures.

2.2. The Free-Space Optical Backplane

The goal of this thesis is to conduct a simulation study of the free-space optical backplane. As described in [21] (see Figure 2.1), a free-space optical backplane consists of optical communication channels (OCCs) that connect printed circuit boards (PCBs). The OCCs are created by smart-pixel arrays (SPAs) [21]. A SPA is an optoelectronic device mounted on a PCB that processes optical and electrical inputs and outputs (I/Os).

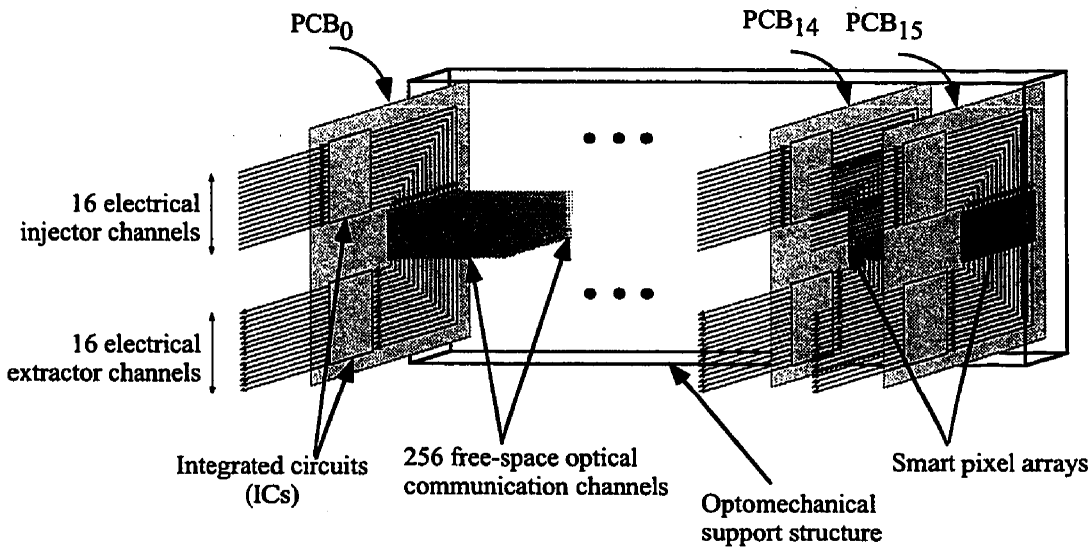


FIGURE 2.1. Free-space optical backplane.

To communicate with other PCBs, a PCB transmits electrical signals to its SPA through electrical channels. The SPA then converts the electrical signals to optical signals and transfers them to the destination PCB via the optical channels. At the destination PCB, the SPA extracts the optical signals and converts them back to electrical signals. The connectivity provided by a unique combination of optical channels and electronic channels creates a design space called the *"HyperPlane"* [21]. The backplane is predicted to offer

over 10,000 high-performance connections per PCB yielding an aggregate bandwidth of over 1 Tbps.

The PCBs also contain integrated circuits (ICs) used for specific applications. As described in [21], the ICs could be elements of parallel computing machines for distributed computing. In this thesis, the ICs are high performance switching elements for telecommunication applications.

2.3. Bus-based Switches

As stated in [22], by programming the SPA appropriately, a *HyperPlane* with k unidirectional optical channels in each direction can embed a number of conventional point-to-point graphs of degree k [1]. A crossbar switch [1] is one example of a network which can be embedded. In this thesis, we assume that the *HyperPlane* has 256 bidirectional optical channels and each optical channel is 8 bits wide. In the following sections, the standard crossbar and the dilated crossbar switches will be discussed in detail.

2.3.1. Standard Crossbar

Standard crossbar switches, also referred to as matrix and fully connected [2] switches [17], provide N^2 paths between N inputs and N outputs. In a standard crossbar switch, busses which connect to the inputs and outputs are assumed to run at the same speed. A bus is a data path for two devices to communicate. Figure 2.2 shows the structure of a crossbar switch with two different architectures. In both architectures, each input can send a packet over a *horizontal broadcast bus*, and each output may receive from one of the N inputs at each clock cycle by using *vertical concentrator busses*. In fact, when a standard crossbar is embedded in the *HyperPlane*, the *horizontal broadcast busses* are the optical channels in the backplane.

2.3.1.1. *Architecture*

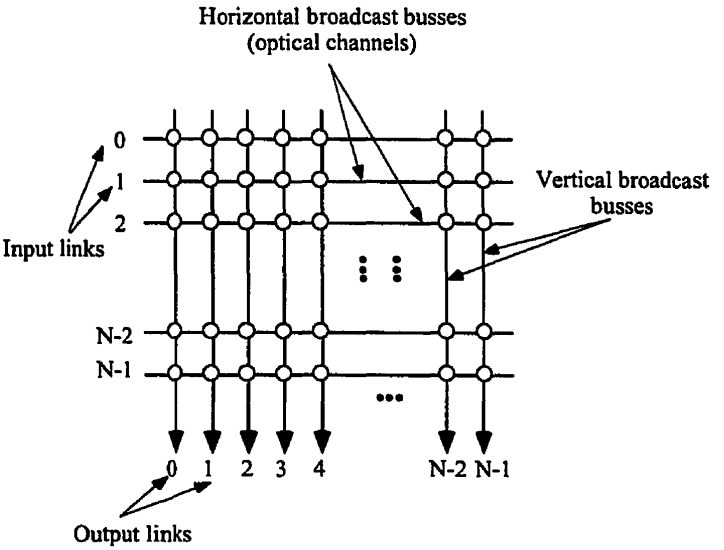
Figure 2.2(a) shows the structure of a standard crossbar switch with *distributed concentrators* [3]. In this architecture, each vertical concentrator bus uses a distributed concentrator to select one of the inputs to forward to the output, when two or more inputs are simultaneously trying to send a packet to the same output.

[1] Any graph in which the number of edges incident to a vertex is upper bounded by some constant k is a k-degree graph.

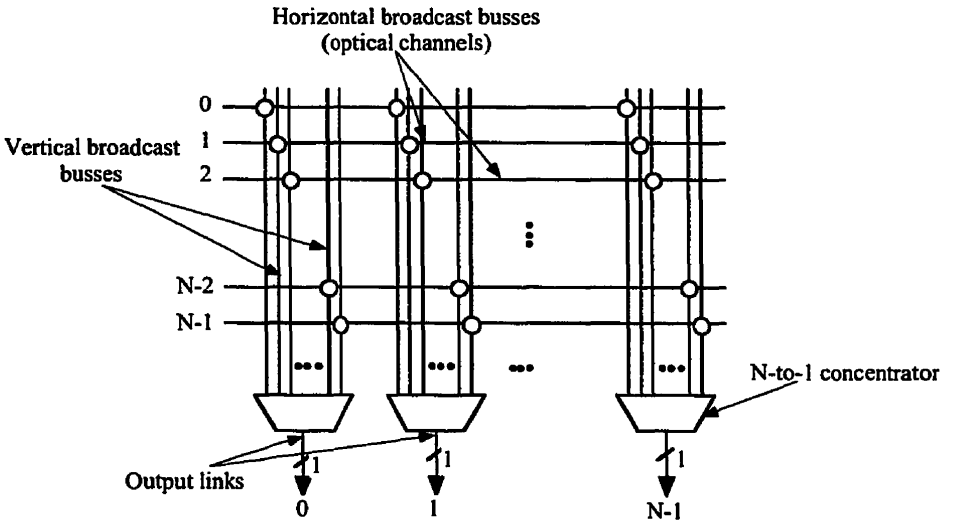
[2] A graph where each vertex connects to every other vertex is a fully connected graph.

[3] A concentrator is a device that reduces the number of input links to a smaller number of output links.

Standard Crossbar



(a) Standard crossbar with distributed concentrators



(b) Standard crossbar with centralized concentrators

FIGURE 2.2. Structures of standard crossbar.

Figure 2.2(b) illustrates an alternative structure for a standard crossbar switch with *centralized concentrators*. As shown in the figure, there are vertical concentrator busses connecting each input and output. Vertical concentrator busses that are connected to the same output are grouped and connected to an N-to-1 concentrator, in which the concentrator is used to select one packet for the output when contention occurs because more than one input are simultaneously trying to send a packet to the same output. The number of vertical broadcast busses used in this structure is N times more than that used in the structure containing distributed concentrators.

2.3.1.2. *Non-Blocking Property*

Blocking occurs when two or more packets contend for the same output link and only one of them can get through. A switch is said to be *internally non-blocking* [4] when it is guaranteed to deliver any permutation of packets without blocking. Since every input link in the crossbar switch has a direct path to every output, it will not suffer from packets competing for the same resource (i.e., switching path) internally, when the traffic forms a permutation. In other words, this architecture will not lose packets internally, but only at the outputs; therefore, all crossbar switches are internally non-blocking switches.

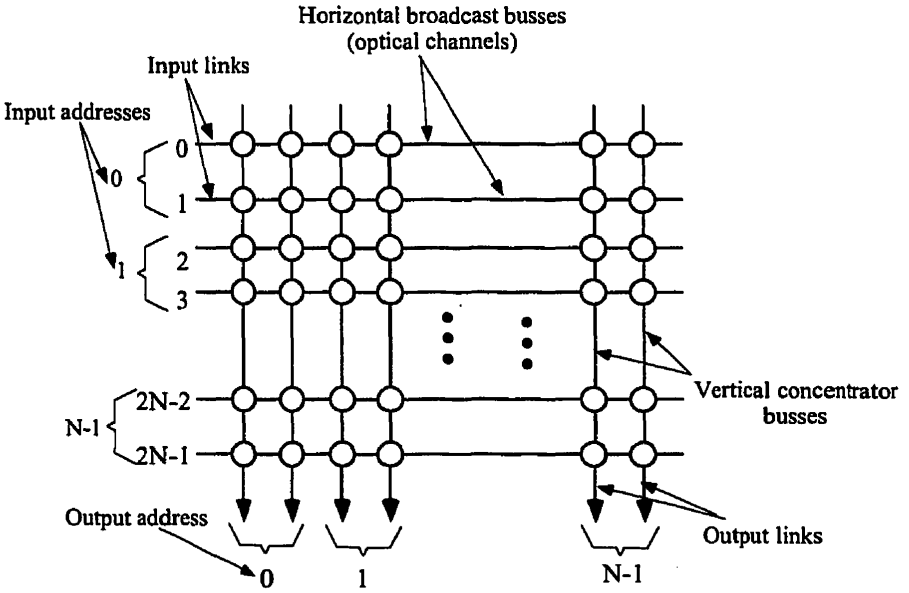
2.3.2. Dilated Crossbar

The performance of a standard crossbar switch can be improved by allowing more than one input link to have the same address (see Figure 2.3) [**8, 20, 27**]. Similarly, it can also be improved by allowing more than one output link to have the same address. Crossbar switches of this kind are known as *dilated crossbar switches*. For example, if a switch has aN input links consisting of N groups of a input links each, a is referred to as the *input-dilation* of the switch. Similarly, if a switch has bN output links consisting of N groups of b output links each, b is referred to as the *output-dilation* of the switch. A dilated crossbar switch is, therefore, similar to the standard crossbar, except that input-dilation and/or output-dilation is greater than one. Figure 2.3 shows an example of a $2N \times 2N$ dilated crossbar switch with input-dilation $= 2$ and output-dilation $= 2$.

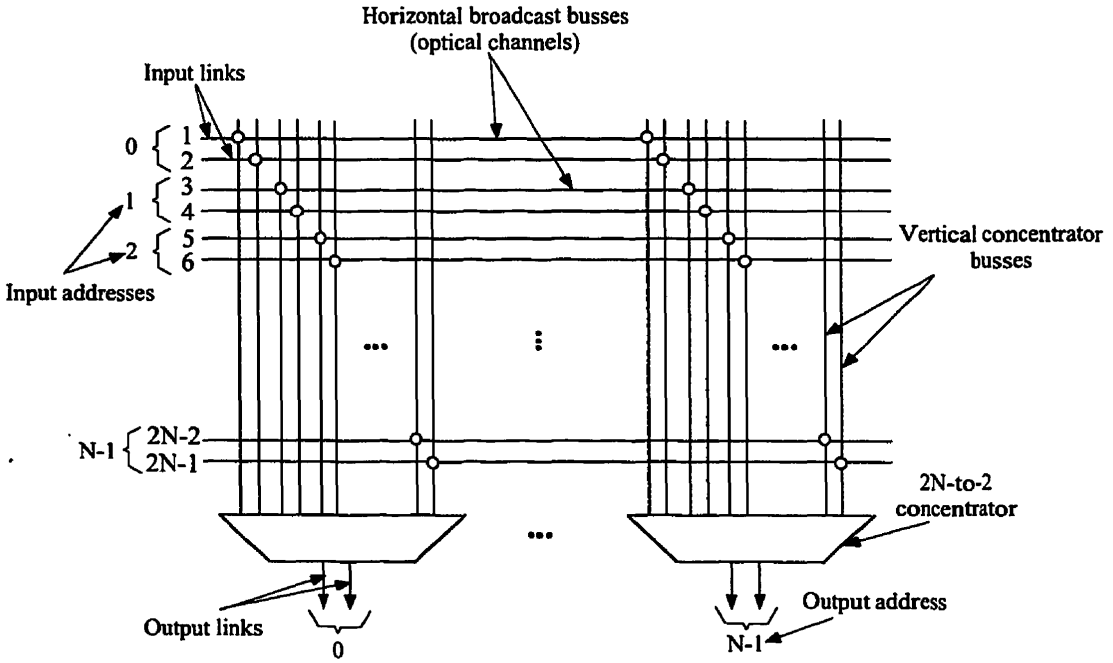
2.4. Queueing Disciplines

Given a random traffic model, in every clock cycle it is possible that the number of packets arriving at an output is greater than its output-dilation, causing packet loss. To reduce packet loss, buffering (or queueing) can be used. There are three main queueing strategies that can be implemented in a switch, namely: input queueing, output queueing,

Dilated Crossbar



(a) Dilated crossbar with distributed concentrators
(input-dilation = 2, output-dilation=2)



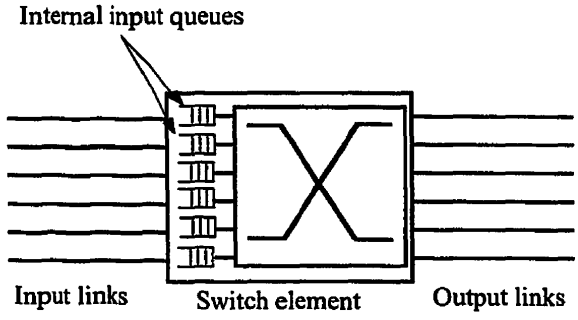
(b) Dilated crossbar with centralized concentrators
(input-dilation=2, output-dilation=2)

FIGURE 2.3. Structure of dilated crossbar (*input-dilation=2, output-dilation* = 2) [20].

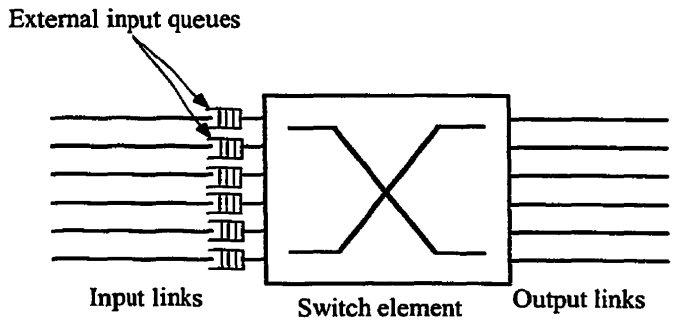
and central queueing [4]. These strategies differ by the physical location of the buffers, i.e., at the input, at the output, or inside of the switching element. Since only input queueing and output queueing are used in the simulation model developed in this thesis, the reader may wish to skip the section on central queueing (Section 2.4.3).

2.4.1. Input Queueing

The first strategy solves the contention problem by input queueing. As shown in Figure 2.4, each input link contains a queue (referred to as *input queue*), and arriving packets are stored in the queue until the selection policy (such as round robin or random) determines that the packet may be transferred. Hence, packets will be sent from the input buffer to their destination outputs without contention.



(a) Buffered switch element with internal input queues



(b) Unbuffered switch element with external input queues

FIGURE 2.4. Input queueing.

Input queues can be implemented either *within* or *outside* a switch, as shown in Figure 2.4. Input queues implemented *within* switches are referred to as *internal input queues*; otherwise, they are said to be *external input queues.*

2.4.1.1. *Head-of-Line Blocking*

Suppose packet i at the head of a *FIFO*[4] (*first in first out*) queue is blocked because some other packets are also destined for packet i's destination. Moreover, suppose packet j is right behind packet i and is destined to an output where no other packets are destined. Packet j cannot be transferred even though its destined output is idle because packet i is blocking its transmission. This type of blocking is referred to as *Head-of-Line* (HOL) blocking [**4**], and is a major disadvantage of input queueing.

2.4.1.2. *Performance*

It is known that switching elements based on the input queueing principle are limited in performance. This limitation is mainly due to HOL blocking. According to [**13**], the maximum obtainable load of this type of queueing is only 58.6%. Nevertheless, the performance of input buffered switches can be improved by

- increasing the speed of the input/output channels [**8, 28**],
- increasing the output-dilation [**27**], or
- using an alternate input queueing approach (e.g., random discipline) other than FIFO discipline [**16**].

2.4.2. Output Queueing

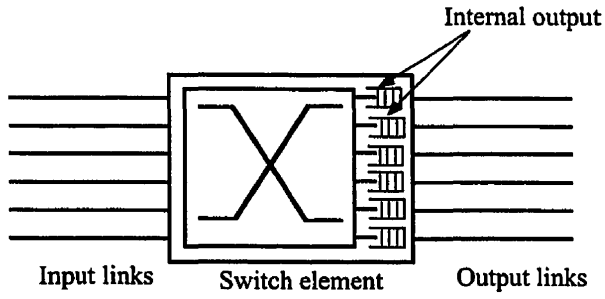
As shown in Figure 2.5, contention can also be solved by placing queues at each outlet of the switch. At an output link, packets arriving from different input links are queued in the output queue in FIFO manner. In each clock cycle, the maximum number of packets that may leave an output queue is equal to the output-dilation of the switch. One classic example of a switch that adopts this method is the knockout switch proposed by Yeh, Hluckyj and Acampora in 1987 [**26**].

Similar to input queues, output queues can also be implemented *within* or *outside* a switch. Output queues implemented *within* switches are referred to as *internal output queues*; otherwise, they are said to be *external output queues* (see Figure 2.5).

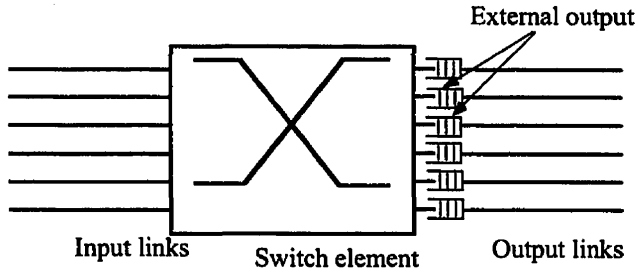
2.4.2.1. *Performance*

In order to ensure that packets remain in the correct sequence, simple FIFO discipline is used to control the output queues. Moreover, since HOL blocking will not occur in output queueing, the mean queue length (or mean waiting time) will be lower and the mean

[4] *FIFO* is a service discipline of a queue [2]. Packets in a FIFO queue will be transmitted in the order that the arrive at the queue.



(a) Buffered switch element with internal output queues



(b) Unbuffered switch element with external output queues

FIGURE 2.5. Output queueing.

throughput will be higher for output queueing switches than for input queueing switches. An analytical performance comparison of input and output queueing is given in [13].

2.4.3. Central Queueing

Central queueing [5] (see Figure 2.6) uses a shared buffer for all inputs and outputs. In other words, all incoming packets to a switch are stored in a central queue, and every output will select the packets destined for itself from the central queue in the FIFO discipline.

2.4.3.1. *Performance*

It can be intuitively explained that the mean waiting time and mean throughput of central queueing is similar to that of output queueing [4]. Moreover, since the buffer is shared, a more effective use of the buffer can be achieved. Because all packets are stored in one shared buffer, the FIFO discipline in central queueing will be much more complicated than that of output queueing. Also, there is a limitation to the maximum achievable size of a shared-memory type switch [10].

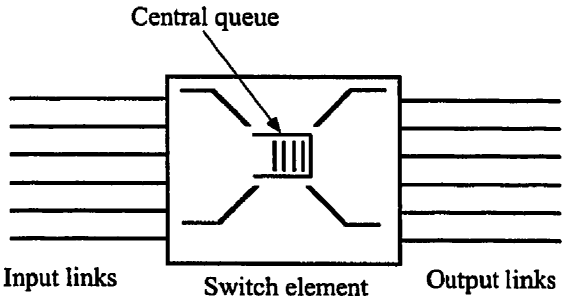


FIGURE 2.6. Central queueing.

2.5. Summary

In this chapter, an introduction to the free-space optical backplane have been given, and the standard crossbar and dilated crossbar switches which can be embedded into the backplane has been described. In addition, an overview of queueing disciplines for the switches has been given. The topics discussed in this chapter are intended to give the reader an adequate background for understanding the rest of the thesis.

CHAPTER 3

Architecture

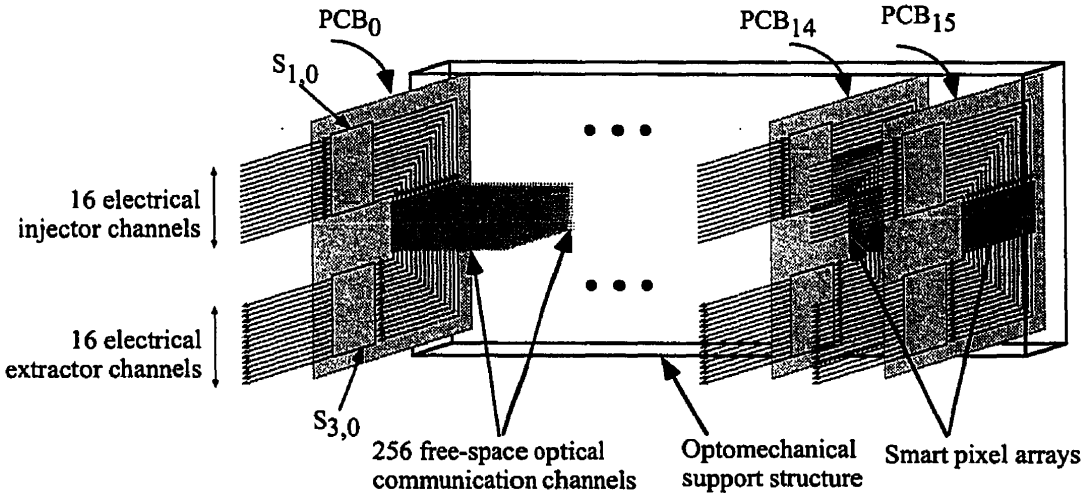
3.1. Problem Formulation

In the optical backplane, PCBs communicate by sending packets through optical channels (see Figure 3.1(a)). In this way, the backplane is functioning as a *switch* (or switches). Since packets are transported by means of optical channels, this kind of switch is referred to as an *optical switch* [**25**].

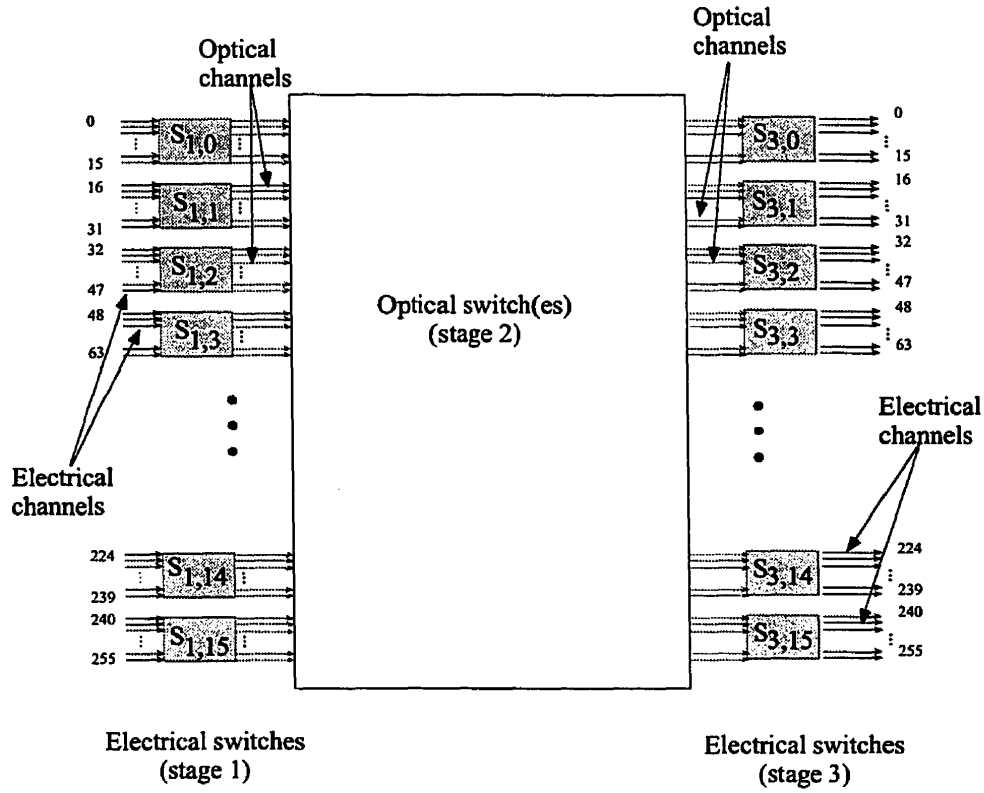
As mentioned in Chapter 2, the ICs on the PCBs are high performance switch elements. These switch elements are referred to as *electrical switches* because switching is performed electrically. Furthermore, these electrical switches are used for sending and receiving packets to and from the optical switch(es).

Figure 3.1(b) illustrates the interconnection between the electrical switches and optical switch(es) in the optical backplane [**18**]. One may note that these switch elements are organized into three stages. The first stage consists of electrical switches for sending packets to the optical switch(es), the second stage consists of optical switch(es), and the third stage consists of electrical switches for receiving packets from the optical switch(es). With this connectivity, the switch elements in the optical backplane and the PCBs can be viewed as a *3-stage switching network* (refer to Section 1.2) [**18**].

In our simulation, we will model the optical switches in the backplane and the electrical switches that are connected to the optical switch(es) as a *3-stage switching network*. Specifically, three different configurations of this network, namely, System I, System II and System III, are investigated. These systems have different optical and SPA complexities; as such, they have different number of optical switches in the second stage. Through simulations, we find that the use of different numbers of optical switches with different complexities translate into different throughputs, delays and loss rates of the network. The results of these simulations will be used to obtain the relationships between the number/complexity



(a) Free-space optical backplane



(b) 3-stage switching network

FIGURE 3.1. Modelling the free-space optical backplane as a 3-stage switching network.

of the switches used and the performance of the network. These relationships can be used as a guideline for building an efficient backplane.

In addition, the simulated network will be constrained to be an *open network*, wherein packets are injected to the network and are always removed from the network once they arrive at their destination. This network is also *internally non-blocking*, so packets can only be lost at the inputs of the first stage. For the rest of this chapter, the specification of this model will be discussed in detail.

3.2. Setting Objectives

In [**20**], several internally nonblocking switches based on multiple channels are proposed for embedding into the backplane. In this thesis, we will focus on studying the performance of three-stage switching networks with dilated crossbar switches embedded in the backplane. In particular, we will restrict our studies to systems that have sixteen PCBs and 1, 4 or 16 optical switches embedded in the backplane [**18**]. Furthermore, each PCB are constrained to have two electrical switches, with one for sending packets to the optical switch(es) and the other for receiving packets from the optical switch(es). We will refer to the opto-electronic switching systems that we are going to investigate as Systems I, II and III, respectively. Since these three systems are very similar, it would be redundant to discuss all of them in detail. In this chapter, we will first discuss the characteristics of System II, and then the structures of Systems I and III will be presented.

3.3. System II

As mentioned in Section 3.2, each PCB on the backplane consists of one electrical switch for sending data to the optical switch, one electrical switch for receiving data from the optical switch, and a smart pixel array [1] for processing optical and electrical I/O's (see Figure 3.1(a)). In System II, four optical switches are embedded in the backplane. Each optical switch has sixty-four input links and sixty-four output links, and each electrical switch has sixteen input links and sixteen output links. Moreover, each switch in this system consists a dilated crossbar switch.

Figure 3.2 illustrates the interconnection of the switches in System II. The first stage consists of sixteen electrical switches, the second stage consists of four optical switches, and the third stage consists of sixteen electrical switches. Every switch in stage one has 4 channels communicating with every optical switch in stage two, thereby using up all 16

[1] A smart pixel array is an optoelectronic device mounted on a PCB that processes optical and electrical inputs and outputs (I/Os).

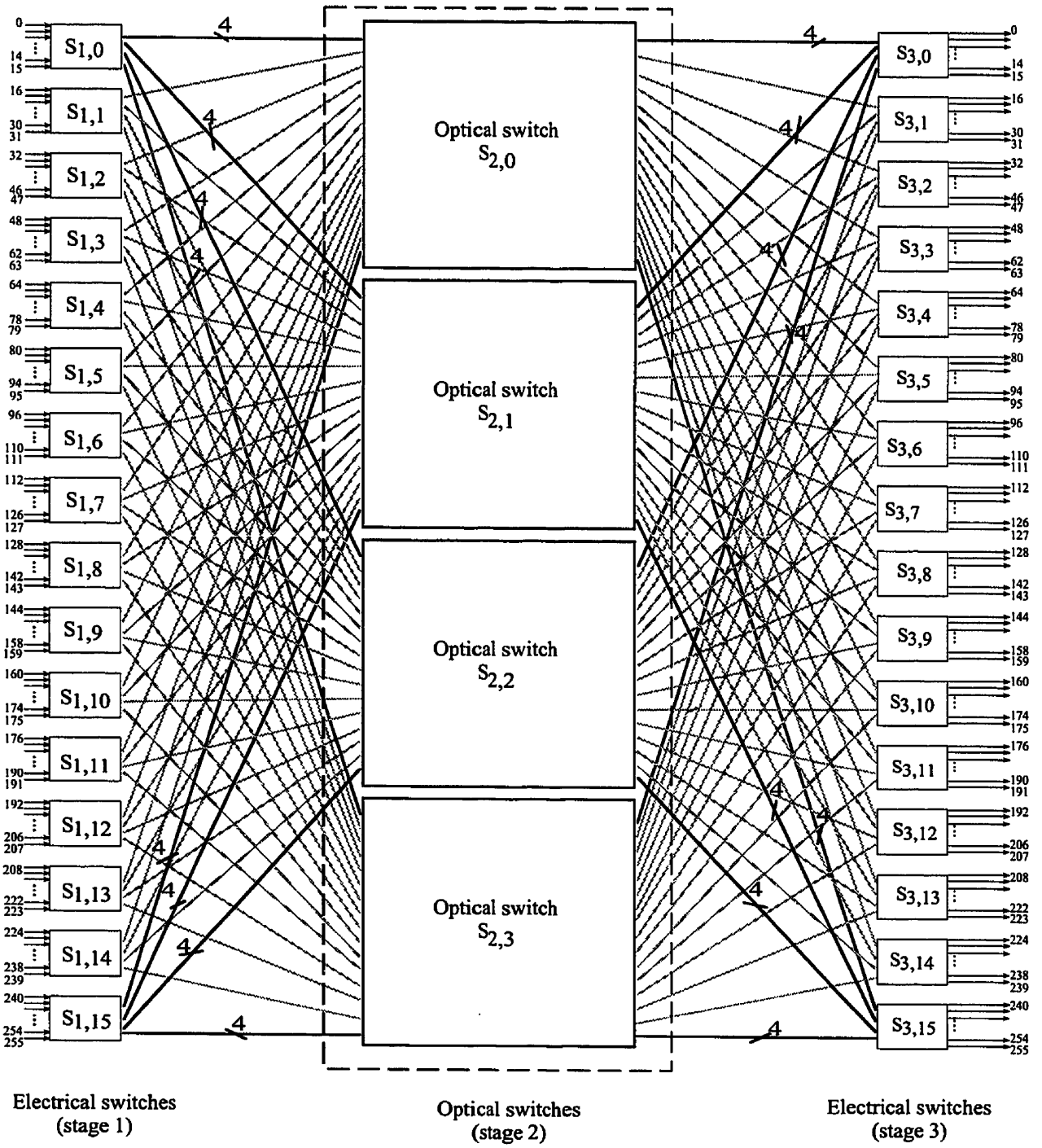


FIGURE 3.2. Interconnection of switches in System II [18].

output links of the stage one switch. Groups of 4 channels interconnect every optical switch in stage two to every electrical switch in stage three, thereby using up all 16 input links of the stage three switch. The way that the switches are connected in this network system, together with the fully-connected property of crossbar switches, enables packets from any input link of a switch in stage one to reach any output link of a switch in stage three.

3.3.1. System II with External Queues

Owing to the performance advantages offered, output queueing [**4, 6, 13**] is employed in this thesis. However, practical output queue size limitations may require additional queueing at the inputs. Therefore, we allow the option of adding external queues at the input links in the first stage of the systems.

3.3.2. Switch Designs

Figure 3.3 shows the architectures of the switches for each of the three stages in System II. This figure, together with Figure 3.2 on page 18, provides a precise picture for the architecture of System II. In the following subsections, the switch architecture of each stage will be described in detail. Furthermore, Table 3.1 (on page 21) provided at the end of this section gives a summary of the switch architectures for System II in each stage.

3.3.2.1. *Stage One*

As shown in Figure 3.3(a), each switch in stage one has sixteen input links and sixteen output links. Each switch element in stage one is output buffered.

The crossbar switches in stage one have sixteen input links and d times sixteen output links. Each of these switches is 1 input-dilated and d output-dilated [2] (where $1 \leqslant d \leqslant 16$). Moreover, groups of d output links from the crossbar switch are connected to a single output queue, and the output of this queue is connected to an output link of the overall switch element.

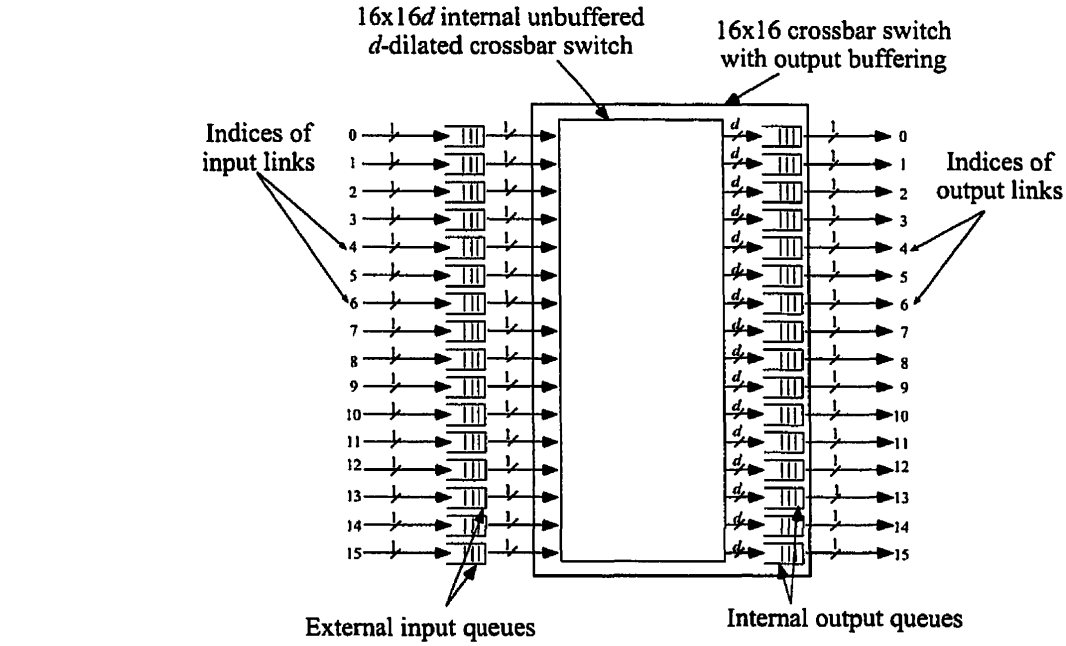
3.3.2.2. *Stage Two*

The architecture for the switches in stage two is shown in Figure 3.3(b). A switch in stage two has sixty-four input links and sixty-four output links. Each of these switch elements has an input-dilation of one and an output-dilation of four.

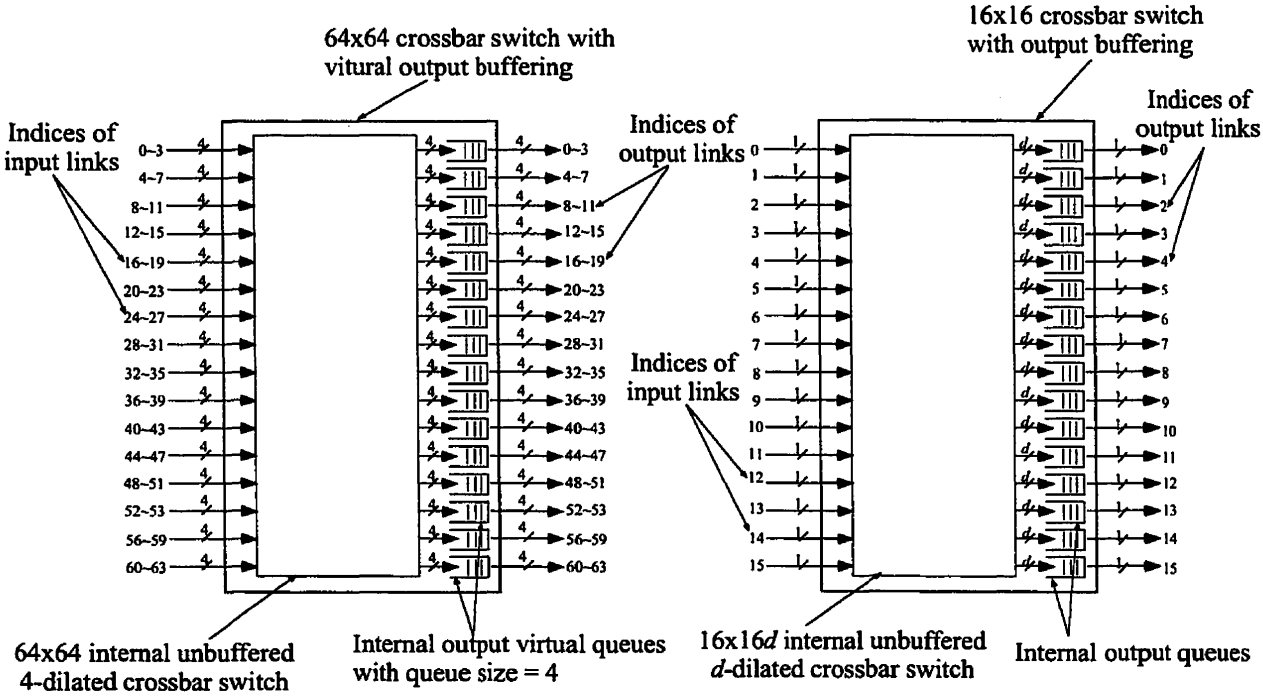
Furthermore, switch elements in stage two can be either *buffered* or *unbuffered* [3]. If switches are buffered, queues within each switch are *"real"*; otherwise, queues within each

[2] Dilation *"d"* is a user-defined parameter.

[3] Buffered or unbuffered switch(es) in stage two are defined as a user option.



(a) Stage 1



(b) Stage 2
(assuming switches are unbuffered in this stage)

(c) Stage 3

FIGURE 3.3. Designs of switches for System II.

switch are *"virtual."* A *"real"* queue is a buffer that can store packets; whereas, a *virtual* queue is an *imaginary* queue that represents the occupancy of a communication link[4].

If the switches in stage two are *buffered*, the crossbar switches within each of these switches are assumed to have an input-dilation of one and an output-dilation of d [5] (where $1 \leqslant d \leqslant 16$).

If the switches in stage two are *unbuffered*, the crossbar switches within each of these switches are assumed to have an input-dilation of one and out-dilation of four, in order to keep the blocking probability low. The output queues are *"virtual"* and each of them has a capacity of four. As mentioned before, a *virtual queue* is an imaginary queue which represents the occupancy of the link(s) to which it is connected. Therefore, even if the switches in stage two contain virtual queues, they are still unbuffered.

3.3.2.3. *Stage Three*

Figure 3.3(c) on page 20 shows the architecture of the switches in stage three. A switch in stage three has sixteen input links and sixteen output links. Every switch element in stage three is output buffered.

The architecture of switches in stages one and three are essentially identical.

Summary of the Switch Designs for System II			
	Stage 1	Stage 2	Stage 3
Type of switch	dilated crossbar	dilated crossbar	dilated crossbar
# of input links	16	64	16
# of output links	16	64	16
Input-dilation	1	1	1
Output-dilation	1	4	1
# of External input queues	0 or 16	0	0
# of Internal output queues	16	16	16

TABLE 3.1. Summary of the switch architectures for System II.

[4] A more detail explanation of *virtual queue* is given in Section 3.3.4.
[5] Dilation *"d"* is a user-defined parameter.

3.3.3. Terminology

Before discussing queueing and packet routing in this switching system, it is vital that various terms are clearly defined. In this chapter and the next, we will repeatedly make references to several terms to describe packets, links, and queues. These terms are:

- *Full*: When a queue reaches its maximum capacity for storing packets, the queue is said to be "*full*".
- *Space*: A queue is said to have "*a space*" or "*spaces*" when it has not reached its maximum capacity for storing packets.
- *Availability*: An "*available*" link is a link that is currently idle.
- *Move*: A packet is "*moved*" when it is removed from a queue and put into an *available* link, making the link unavailable in the process.
- *Forward*: A packet is "*forwarded*" when it is stored into a queue that has a *space*.
- *Move forward*: A packet "*moves forward*" when it is removed from a queue and is stored into its next destination queue that has a *space*.

3.3.4. Queueing

In this thesis, we allow switch elements in stage two to be either *buffered* or *unbuffered*. For a system with buffered switches in stage two, queues are *real* and each packet will stay in each stage for at least one clock cycle before moving to the next stage. For a system with unbuffered switches in the second stage, queues in that stage are *virtual* and packets arriving will attempt to move to stage three in the same clock cycle. Table 3.2 provides a summary of the characteristics of queues for System II with *virtual* queues in stage two.

When queues in stage two are *virtual*, they are not able to buffer packets. Packets that fail to move forward in the same clock cycle as they arrive at stage two will be removed from the queues at the end of each clock cycle. To prevent this kind of internal packet loss, the following method is used to simulate unbuffered stage two:

- First, in stage one, create a copy of each packet that attempts to move forward.
- Attempt to forward the copied packet to the destination virtual queue in stage two.
- (a) If the copied packet is forwarded to stage two, an attempt is made to transmit the copied packet to stage three immediately. (b) Otherwise, the copied packet is removed from the network, leaving the original packet in stage one.

- If the copied packet is forwarded to its destination queue in stage three, an acknowledgement message is returned along the open path through which the copied packet had passed, and the original copy of the packet in stage one is removed upon receiving the acknowledgement.

Summary of the Queues for System II				
	External Input Queues	Internal Output Queues		
		Stage 1	Stage 2	Stage 3
Queueing Discipline	FIFO	FIFO	FIFO	FIFO
Queueing Capacity	b $(1 \leqslant b \leqslant \infty)$	s $(1 \leqslant s \leqslant \infty)$	4	s $(1 \leqslant s \leqslant \infty)$
Maximum # of arrivals (packets per clock cycle)	1	d	4	d
Maximum service rate (packets per clock cycle)	1	1	4	1

TABLE 3.2. Summary of queues for System II. Assuming unbuffered switches in stage two.

3.3.5. Packet Routing

In this system, each packet carries its own destination address and is routed through the system independently. This way, packets of the same source input link and destined for the same output link may have different paths through the system. Since this system is composed of 16 PCBs, each containing 16 inlets and outlets, there are a total of 256 possible destinations. In order to have a *unique* destination address, outlets in stage three are labeled from 0 to 255. Therefore, an outlet's address is its corresponding label. Similarly, inlets in stage one are labeled from 0 to 255. These labelings are shown in Figure 3.2 on page 18. Furthermore, Figure 3.4 illustrates some examples of the routing of a packet.

3.3.5.1. *Switching Packets in Stage One*

In the case when there are external input queues for each inlet in stage one and a packet is generated by an external process, the packet will be buffered in the external input queue if the queue is not full; otherwise, it will be removed from the network, i.e., there will be overflow.

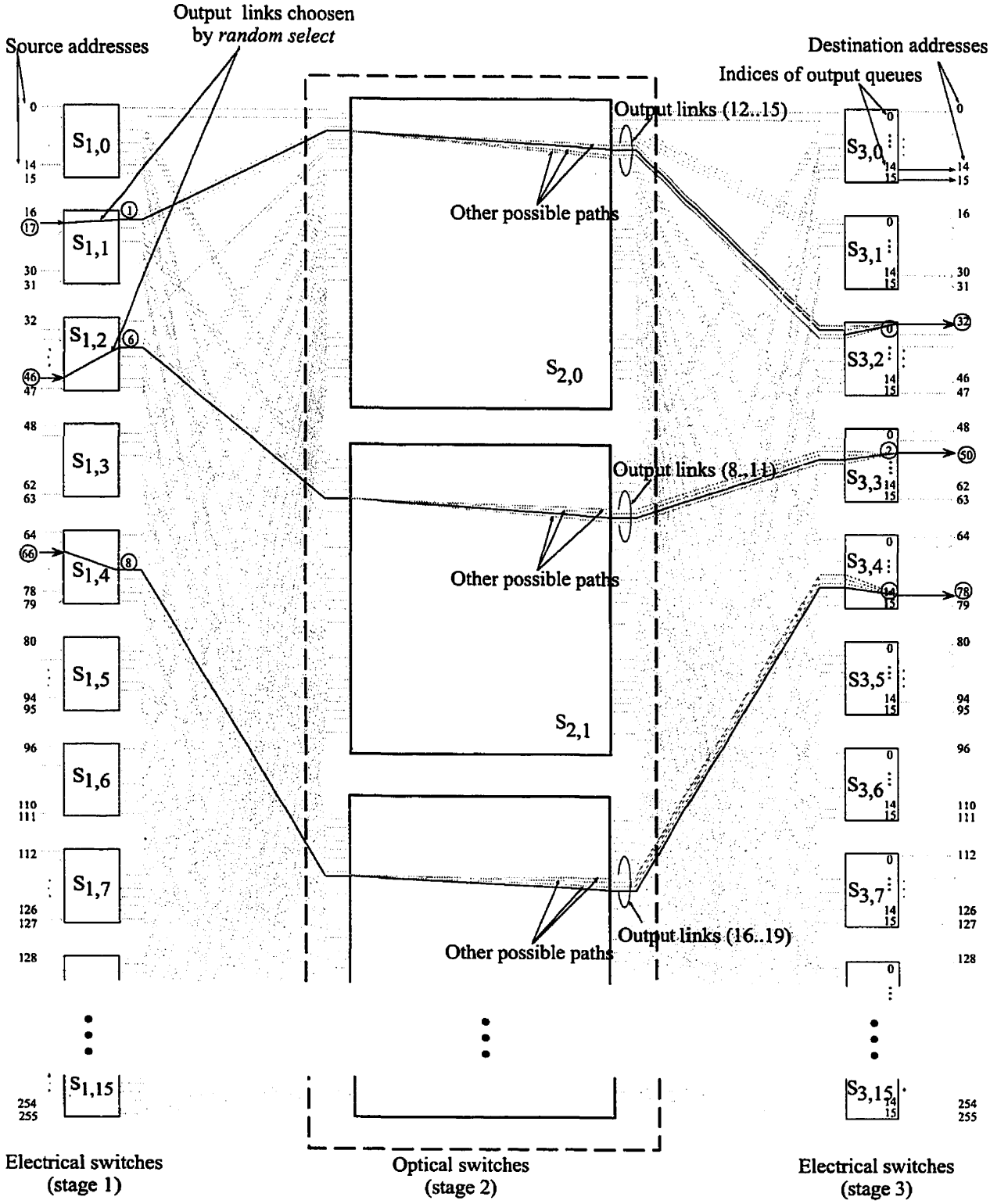


FIGURE 3.4. Examples for routing packets.

Three routing methods are considered to route packets from an external input queue to an output queue in the first stage.

- *Straight Through*:
  (a) An attempt will be made to forward the packet in the ith external queue of a switch to the ith output queue of the switch in stage one (i.e., "straight-through").
  (b) If the attempt fails, (a) is repeated in the next clock cycle.
- *Random Select*:
  (a) An output link of the stage one switch is randomly selected.
  (b) An attempt is made to forward the packet to the selected output queue.
  (c) If the attempt fails, (a) and (b) are repeated in the next clock cycle.
- *Repeat Select*:
  (a) An output link of the stage one switch is randomly selected.
  (b) An attempt is made to forward the packet to the selected output queue.
  (c) If the attempt fails, (a) and (b) are repeated in the same clock cycle until the packet is forwarded, or until all output links have been tried and failed.
  (d) If all output links have been tried and failed, the process is repeated in the next clock cycle.

When there are no external input queues, a packet arriving at an input link will attempt to move into an output queue in stage one immediately. In this case, the methods used for routing packets to the output links in stage one are the same as above, with the exception that a packet will be *"dropped"* if it is unable to be buffered in stage one in the same clock cycle in which it was generated.

Repeat select maximizes the use of queueing capacity in the output queues but consumes more processing time than *random select*. Note that both *random select* and *repeat select* do not depend on the packet's destination. The primary purpose of the two strategies is to randomize the incoming traffic in order to prevent unbalanced traffic in switches of stages two and three [**19**].

3.3.5.2. *Switching Packets in Stage Two*

The purpose of switching in stage two is to switch packets to appropriate switches in stage three.

3.3.5.3. *Switching Packets in Stage Three*

Packets arriving at an input in stage three are switched to their destination output links. Packets which arrive at their destined output queues in stage three will be removed from the network using the FIFO discipline.

3.4. Unbalanced Traffic Pattern using Straight Through Routing in Stage One

In stage two, a packet will be routed to an output link that connects to the packet's destination switch in stage three. In the case where the straight through routing method is used for stage one, there will be some traffic patterns in which more than four packets with the same destination will attempt to route to the same output queue (or communication channel) in stage two, while other output queues (or communication channels) that connect to the same switch remain available or not fully utilized.

Consider the example in Figure 3.5: each of the first four links of switches $S_{1,0}$, $S_{1,1}$, $S_{1,2}$, and $S_{1,3}$ attempts to send a packet through stage two to the same destination switch ($S_{3,0}$) in stage three. According to the interconnection of switches for System II (Figure 3.2), all packets will be routed along to one of the first four output links of $S_{2,0}$. Therefore, 16 packets will contend for four links, resulting in the blocking of 12 packets, while other links (in other switches of stage two) that are connected to the packets' destination switch will remain available or not fully utilized.

Such kind of traffic pattern increases average packet delay and decreases the throughput of the network. It is, therefore, not desirable to have traffic patterns that cause unbalanced utilization of communication links. The routing options *"random select"* and *"repeat select"* will eliminate this unbalanced traffic pattern by "randomizing" the traffic [**19**].

3.5. Other Network Architectures

In this thesis, the three systems differ by the number and size of optical switches used. The following two sections will give a brief description of Systems I and III [**18**]. Table 3.3 on page 29 summarizes the configurations of the three systems.

3.5.1. System I

In this system (refer to figure 3.6 on page 28), a dilated crossbar switch is embedded in the backplane. In other words, there is only one big optical switch in the second stage of

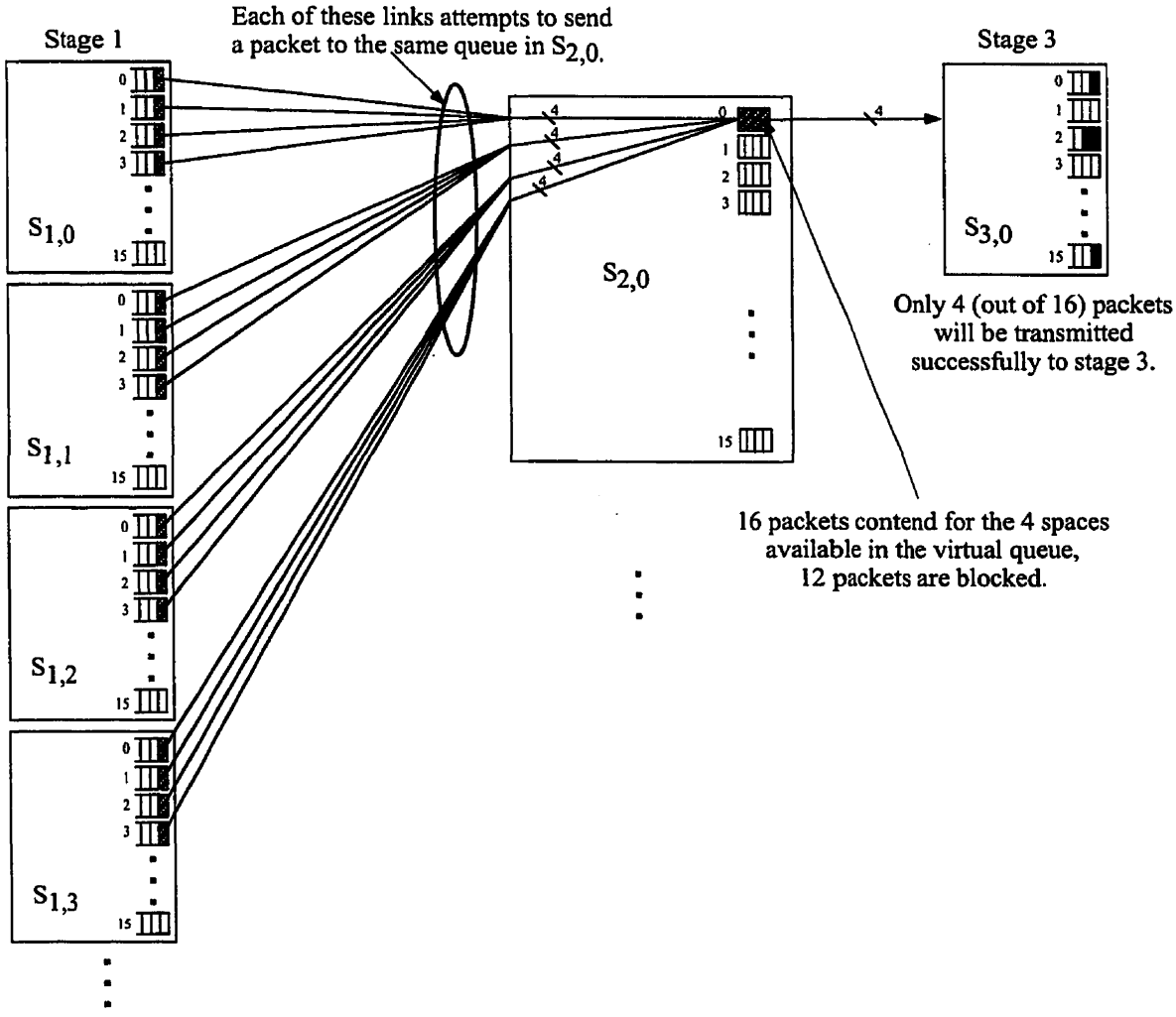


FIGURE 3.5. Example of an unbalanced traffic pattern in System II using straight through routing in stage one.

the network. Moreover, the switch is 256×256 unbuffered, 1 input-dilated, and 16 output-dilated. Therefore, all traffic going out of the first stage will go to the same switch in the second stage. There is no unbalanced traffic pattern in this system. Figure 3.6 shows the interconnection of switches in System I.

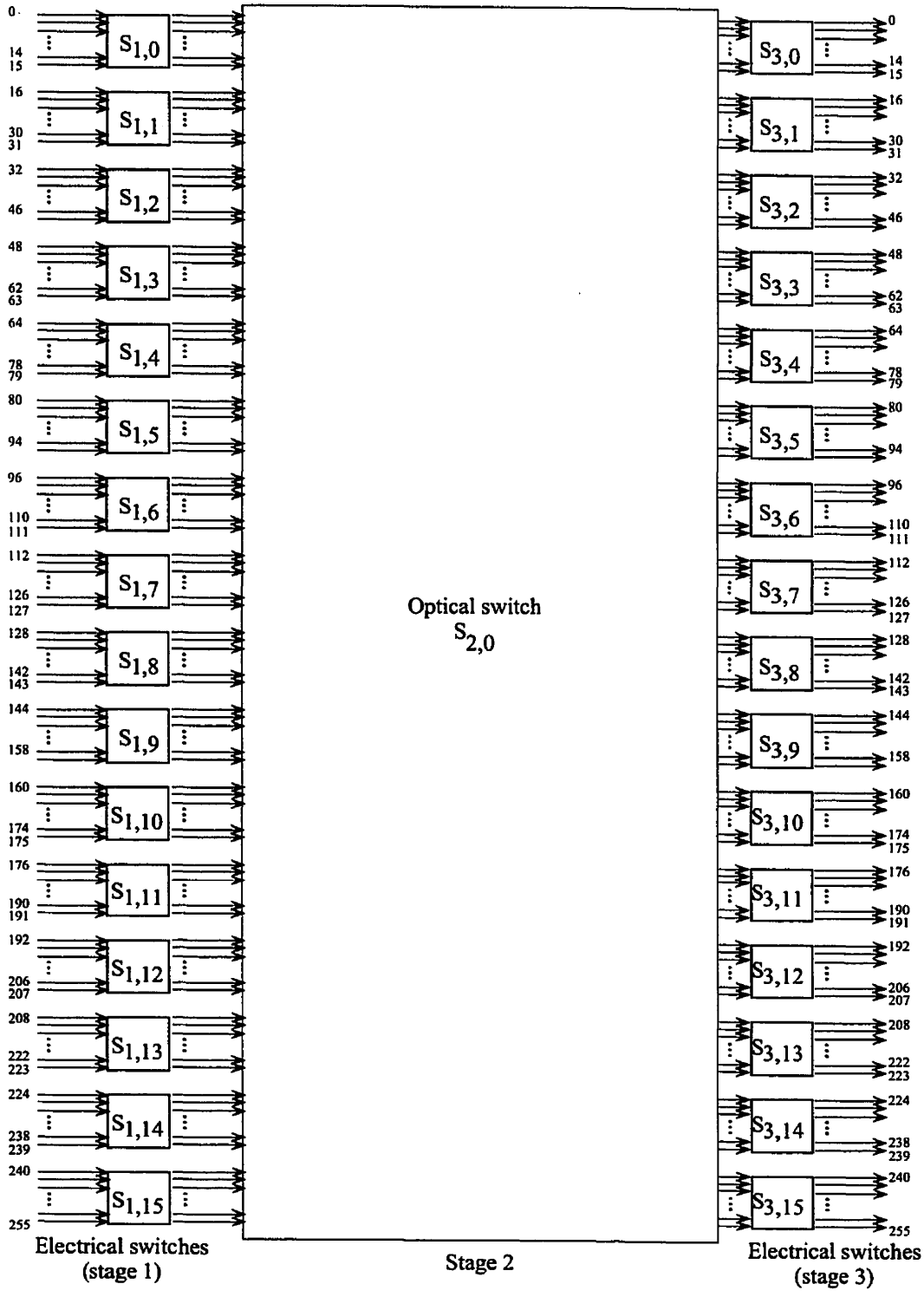


FIGURE 3.6. Interconnection of switches in System I.

3.5.2. System III

The interconnection of this system is shown in figure 3.7 on page 30. In this system, sixteen smaller crossbar switches are embedded in the backplane. In order to create 256 inputs and 256 outputs for stage two, each of the sixteen switches in the second stage has sixteen input links and sixteen output links.

Summary of the system configurations.						
	stage 1		stage 2		stage 3	
	# of electrical switches	dimension	# of optical switches	dimension	# of electrical switches	dimension
system i	16	16x16	1	256x256	16	16x16
system ii	16	16x16	4	64x64	16	16x16
system iii	16	16x16	16	16x16	16	16x16

TABLE 3.3. Summary of the system configurations.

3.6. Summary

In this chapter, we have described the modelling of the switches in the optical backplane and the PCBs as a 3-stage switching network. Specifically, three systems (referred to as Systems I, II and III) with 16 PCBs and 1, 4, or 16 optical-crossbar switches are discussed. These three network models share the same routing and queueing strategies and differ by the number of optical-crossbar switches used. Therefore, the interconnections of switches in each system are different. In the actual implementation, these three systems differ by the complexity of the smart pixel arrays used [18]. The purpose of this chapter is to provide the specifications for the preparation of a computer program, so that when run, the program will exhibit the behavior of the system.

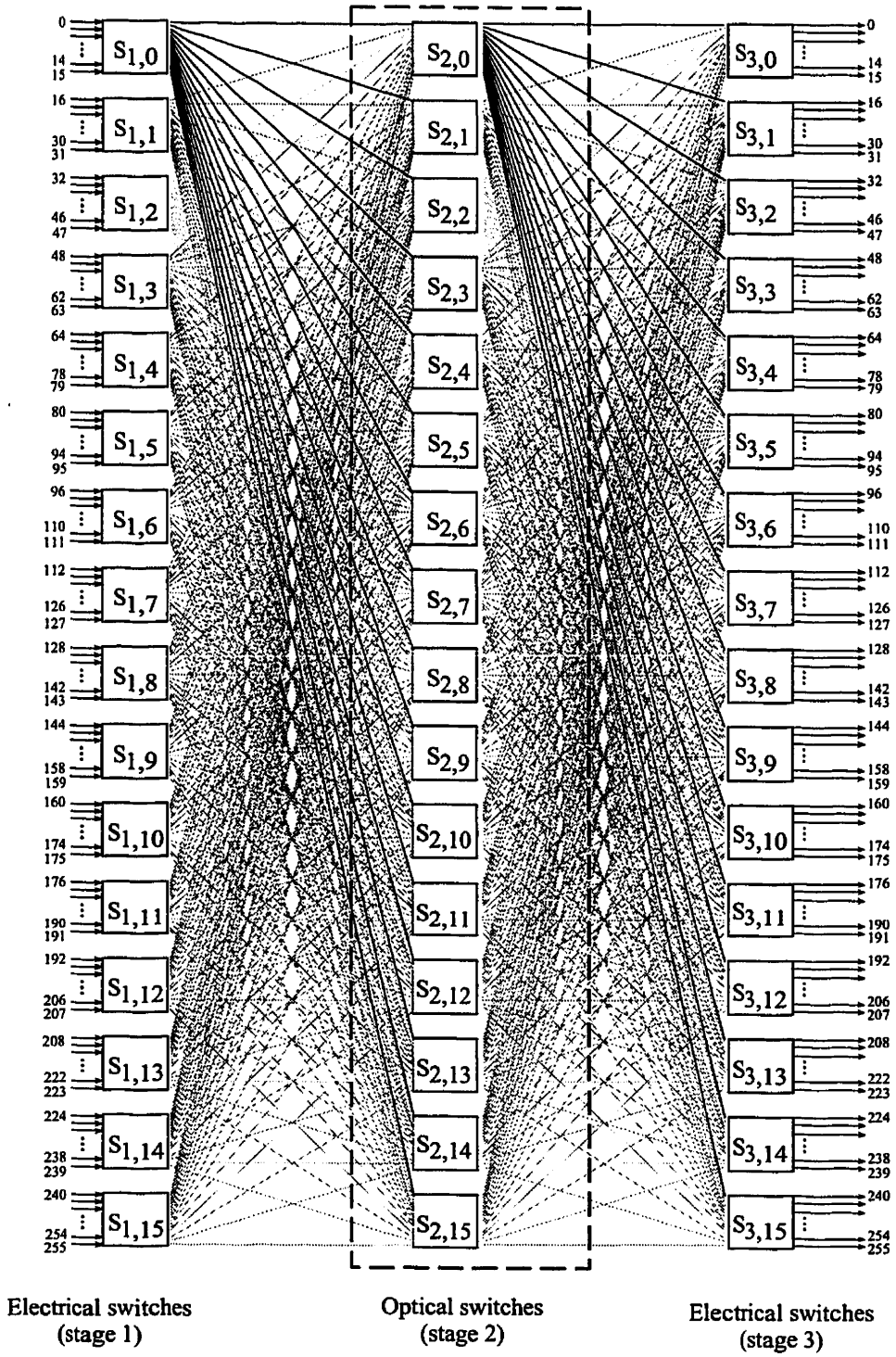


FIGURE 3.7. Interconnection of switches in System III.

CHAPTER 4

Software Simulation

4.1. Objectives and Methodology

As described in the previous chapter, the goal of this thesis is to develop a software simulation model to predict the behavior of the 3-stage crossbar switching network that can be supported by the backplane. In terms of software engineering, the process of building this software simulation model is a software maintenance activity. This activity involves the reverse engineering and re-engineering of a banyan network simulator developed by Prof. Ted Szymanski and graduate students at Columbia University, to a 3-stage crossbar network simulator. In order to maintain a systematic, sequential approach in re-engineering the software, the classic water fall software development model is followed [**15**].

The development cycle of the simulator begins with design recovery and enhancement of the data structures in the original simulator (i.e., the bayan network simulator), as shown in Figure 4.1. This is followed by the simulator design and implementation stages. In order to ensure correctness of the code, testing is performed during the implementation stage. Once the design has been completely implemented and the code has been thoroughly tested, the simulator is evaluated. During evaluation, we may discover that changes to the design would be necessary. If this is the case, the simulator is refined by implementing and testing these changes, after which evaluation is again performed. When no more changes to the design are necessary, an analytic model of the switching network is built and used in verifying the correctness of the simulator design. This model is based on the mathematical analysis which was provided by Prof. T. Szymanski [**18**]. Just as in the implementation of the simulator design, testing is performed during model building. Finally, after the correctness of the simulator has been verified, the development cycle ends with the collection of data from simulations of the switching network. The complete simulator development cycle is illustrated in Figure 4.1.

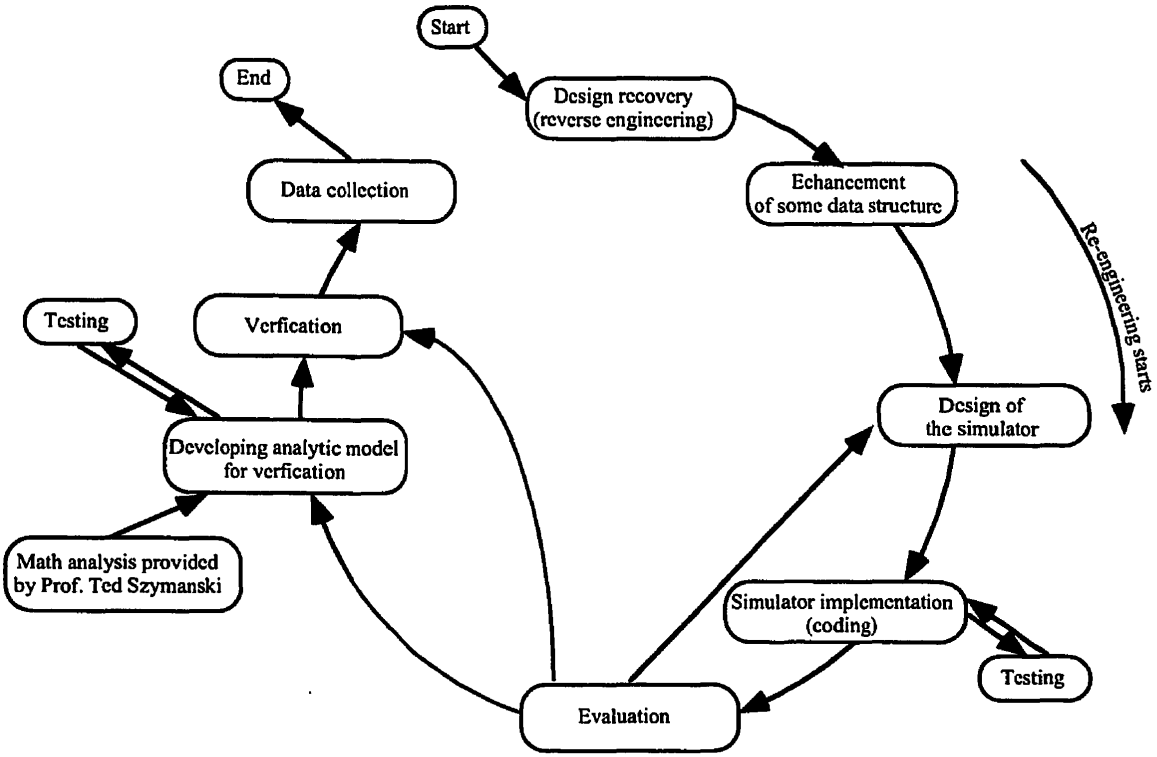


FIGURE 4.1. Software development model.

The fundamental goal in designing this software simulator is to investigate the effects that various parameters have on the 3-stage crossbar switching model described in Chapter 3. Specifically, three different configurations of this network, namely, System I, System II and System III, are investigated. These systems are of different optical and SPA complexities. The tradeoffs between the complexity of the optical switches in the second stage and the performance of the system, measured in terms of throughput, delay and loss rate will be discussed. These relationships form a guideline which can be used for designing the smart pixel array of the optical backplane.

The following sections discuss in detail how the features and characteristics of the proposed model are captured in the software simulator in order to achieve the goal.

4.2. Assumptions

Before we describe the simulator, we summarize the assumptions used in this chapter. We assumed that input and output links to a switch are of the same speed. Packets are assumed to have a constant length, and the channel time is slotted with a slot size equal to the length of a packet transmission time. Arrival of packets at each input link follows a *Binomial process* [24], wherein the probability of a packet arrival in a clock cycle is p, and the probability of no arrivals in a clock cycle is $1 - p = q$. Since we use the slot length as the unit of time, p also corresponds to the input traffic load (λ packets per clock cycle) to the input link.

Moreover, we assume that *transient traffic pattern* will not occur after 400,000 packets have been transmitted through the network [7]. Finally, *uniform random assignments* [1] are used in any decision making that might otherwise lead to biased statistical results of the simulator.

4.3. The Simulator

In this chapter, a discrete-event simulator of the 3-stage crossbar switching network will be described. The simulator is written in the C-programming language on a UNIX platform. The *gcc* compiler is used for compilation. Moreover, approximately 15 Megabytes of disk memory and 3000 kilobytes of shared memory are needed to run the simulator.

The discrete event simulations are driven by a global clock. In general, packets are transmitted from one stage to another in each clock cycle. Since this is an open network, packets are injected into the input links of stage one and removed from the network from output links in stage three.

4.4. Data Structure

The entities[2] of the network system considered in this thesis consist of *packets*, *queues*, and *switches*. In this section, the data structures of these entities will be described.

4.4.1. Packets

Since packets are of fixed length in this network system, each packet is represented as a record[3] with the following fields [4]:

[1] *Uniform random assignments* means that the N sample points are equiprobable with probability equal to $1/N$. [24]

[2] An entity is an object, item, or component of the system that requires explicit representation in the model [7].

[3] A record is a collection of some values under a single name.

[4] Fields are values of different types within a record.

- Packet ID
- Start time
- Source address
- Destination address (which is also the routing tag)
- Packet pointer

Figure 4.2(a) gives a graphical representation of a packet record. Each packet record contains a unique *Packet ID* to distinguish it from other packets. When a packet is inserted into the network, its *start time* (i.e., the time it enters the network), *source address* and *destination address* are recorded in the packet. A *source address* is the address of the input link (in stage one) at which a packet enters the network. As described in Section 3.3.5, each input link in stage one is uniquely addressed from 0 to 255. Similarly, a *destination address* is the address of the output link (in stage three) from which a packet leaves the network. As described in Section 3.3.5, each output link in stage three is uniquely addressed from 0 to 255.

A packet also contains a *packet pointer* [5] used for referencing in a linked list, which represents a queue.

4.4.1.1. *Pre-defining Packets as a Stack*

Since the packet capacities for all queues in this network system are finite, there is an upper bound for the number of packets ($MAX\_PACKETS$) that can circulate within the network at any time. $MAX\_PACKETS$ can be calculated by summing up the packet capacities of all queues (excluding virtual queues) in the network system. For example, in systems with external queues and unbuffered switches in stage two, there are $16 \times 16 = 256$ external queues before stage one, $16 \times 16 = 256$ output queues in stage one and $16 \times 16 = 256$ output queues in stage three. Note that, in this example, since the switches in stage two are unbuffered, there are no *real* queues in stage two. Given an external queue capacity of 8 and an output queue capacity of 16, there will be a maximum of $(8+16+0+16) \times 256 = 10,240$ packets (i.e., total capacity of external queues plus total capacity of output queues in stages one, two and three) circulating within the network at any time.

In order to avoid the need for allocating memory to a packet every time a packet is injected into the network, a stack of packets (referred to as *Packet Stack*) with size equal to $MAX\_PACKETS$ (or probably a bit larger) is defined and is initialized with unique $PacketID$'s before the simulation starts. During a simulation, when packets are popped

[5] A pointer is defined as a reference (or pointing) to a location of a particular type.

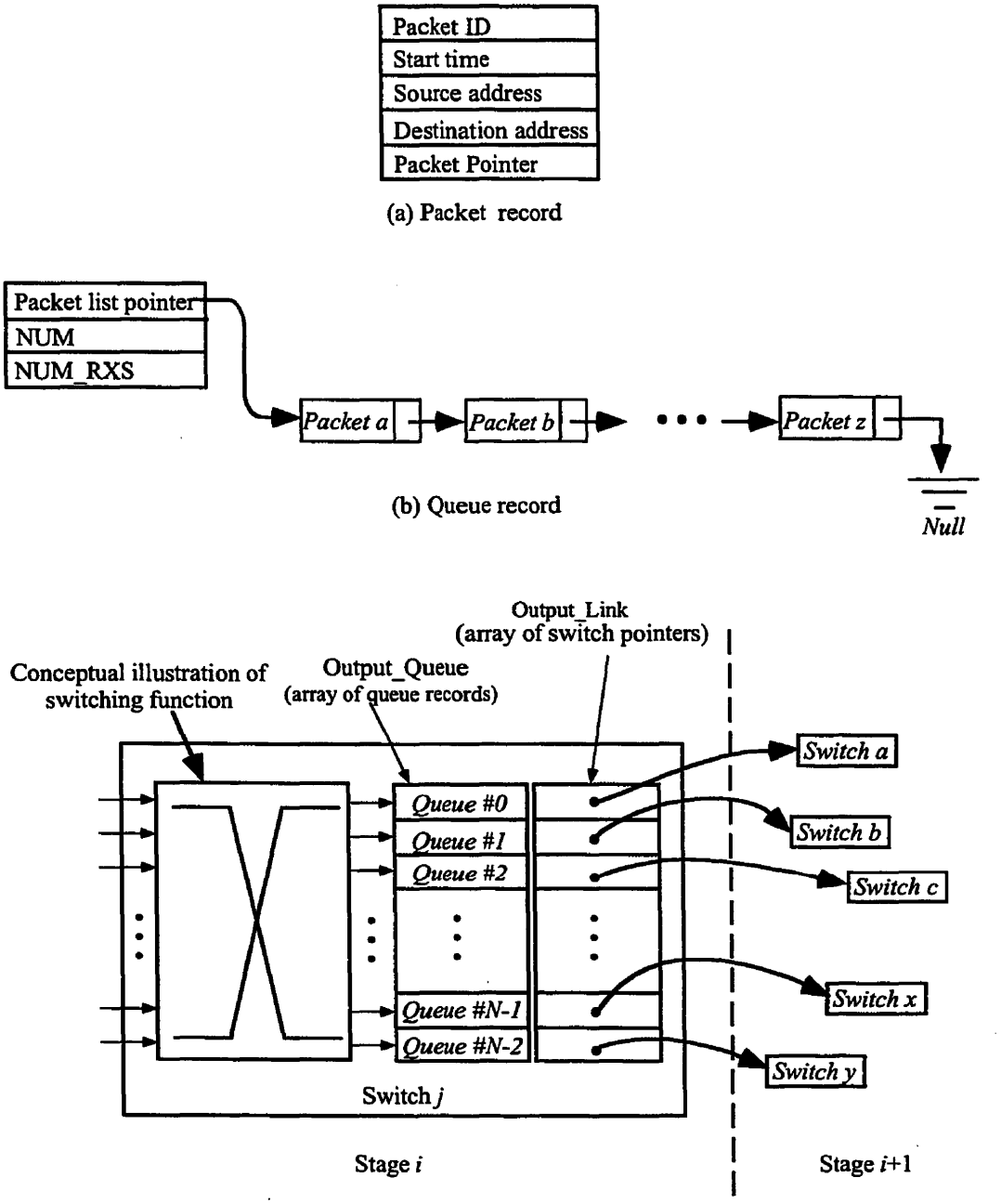


FIGURE 4.2. Graphical representation of entities in the network system.

out from the stack, their *start time*, *source address* and *destination address* are initialized and they are injected into the network. Packets removed from the network are labelled as *"idle"* and are pushed back to the stack. This way, processing time for the allocation of memory to packets during the simulation are saved. Figure 4.3 illustrates this concept graphically.

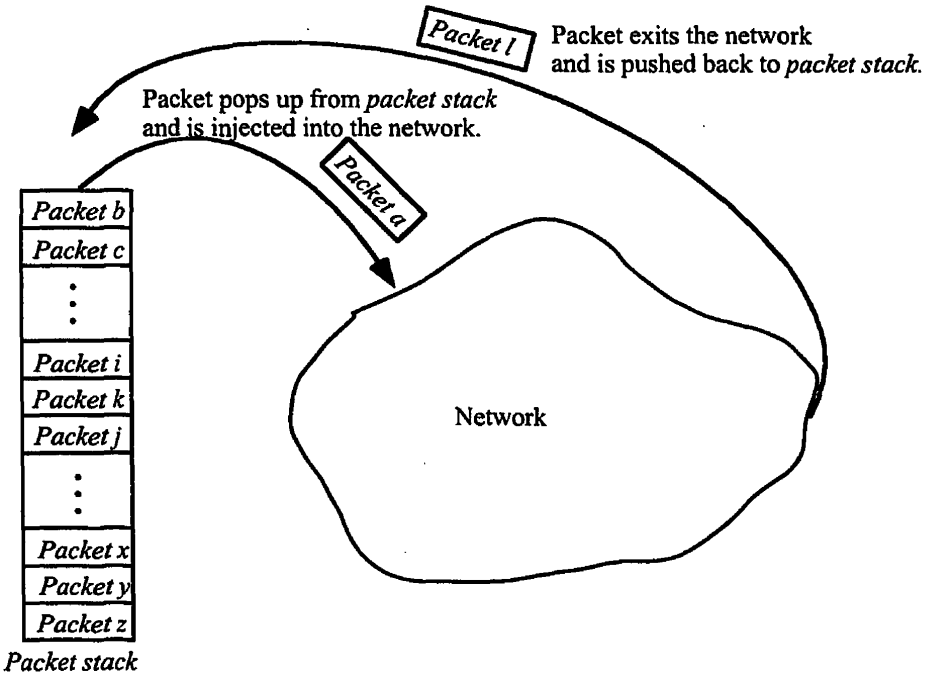


FIGURE 4.3. Pre-defining packets as a stack.

4.4.2. Queues

Although there are three different types of queues, namely: external input queues, internal output queues, and internal virtual queues, in this switching system, they have the same data structure in this model. Each queue is a record that contains the following fields:

- NUM
- NUM_RXS
- Packet list pointer

Figure 4.2(b) on page 35 gives a graphical representation of a queue. *NUM* stores the total number of packets in a queue, and *NUM_RXS* is a counter that counts the number

of packets that have entered the queue during a clock cycle. Consequently, a queue is said to be "*full*" when *NUM* is equal to the size of the queue, otherwise, it is said to have "*space*". Also, an "*available*" link to a queue can be found when *NUM_RXS* is less than the maximum number of packets a queue can receive in each clock cycle. Therefore, a packet may *move forward* to a queue if there is an *available* link to the queue and the queue has *space*.

The *packet list pointer* points to the first element of a linked list of packets. The linked list represents a queue and is referred to as a *packet list*. The first element of the list is referred to as the *head-of-queue packet*. In order to achieve the FIFO property of queues, packets are added to the tail of the list and are removed from the head of the list.

4.4.3. External Input Queues

For a system with external input queues, the queues are modelled as an array (*Ext_Queue*) with a size equal to the total number of input links in stage one (i.e., 256). Therefore, *Ext_Queue[i]* corresponds to the external input queue in the ith input link of the network.

4.4.4. Switches

In this system model, both electrical switches and optical switches share the same data structure. Figure 4.2(c) on page 35 provides a graphical representation of a switch. An $N \times N$ switch is represented as a record that contains the following fields:

- Output_Queue[N]
- Output_Link[N]

Output_Queue is an array of queue records, and *Output_Link* is an array of switch pointers. Since each output queue connects to a link and each link connects to a switch in the next stage, each of the queue records in *Output_Queue* has a corresponding pointer to a switch in the next stage, which is stored in *Output_Link*.

4.4.4.1. *Organizing Switches in the Model*

As described in Section 3.1, switches in this network system are organized into three stages. The first stage has 16 switches, the second stage has 1, 4 or 16 switches (depending on which system is selected), and the third stage has 16 switches. Switches can, therefore, be organized into a 3x16 two dimensional array, and are referred to by using the following notation:

Switch[*Stage #*][*Switch #*]

For example, *Switch*[2][4] refers to the fourth switch in stage 2.

4.5. Network Modelling

With the description of the data structure in the last section, the modelling of the network is described in this section.

4.5.1. Interconnection of Switches in the Model

The switches in this model have to be properly connected before a simulation can start. Each element in the array of switch pointers, *Output_Link*, in every switch is assigned according to the interconnectivity of the system specified in Chapter 3.

In System II, for example, there are sixteen 16x16 switches in stage one, four 64x64 switches in stage two and sixteen 16x16 switches in stage three. Each of the switches in stage one has 4 output links connected to each switch in stage two, and groups of 4 output links interconnect every switch in stage two to every switch in stage three. Therefore, the first four *Output_Link*s of a switch in stage one point to the first switch in stage two ($S_{2,0}$), the next four *Output_Link*s point to the second switch in stage two ($S_{2,1}$) and so on. Similarly, the first four *Output_Link*s of a switch in stage two point to the first switch in stage three ($S_{3,0}$), the next four *Output_Link*s point to the second switch in stage three ($S_{3,1}$) and so on. Queues in stage three are connected to some external devices and point to *NULL*.

In fact, we can calculate *Output_Link[k]* of *Queue[k]* in *Switch[stage #i][switch #j]* with the following formulas:

Computing *Output_Link[k]* for System I, II, and III			
	System I	System II	System III
Stage 1 ($Switch[1][j].Output\_Link[k]$)	0	$k \bmod 4$	k
Stage 2 ($Switch[2][j].Output\_Link[k]$)	$k \bmod 16$	$k \bmod 4$	k
Stage 3 ($Switch[3][j].Output\_Link[k]$)	*NULL*	*NULL*	*NULL*

4.5.2. Packet Routing

4.5.2.1. *Packet Entering and Exiting the Network*

The traffic of the simulator is generated by means of *traffic generators*. *Traffic generators* generate packets at the inputs of stage one according to a binomial probability distribution. The probability of generating a packet per clock cycle at an input link is equal

to the offered load (λ). In fact, a *traffic generator* at an input will pop a packet from the *Packet Stack* and inject it into an input link in stage one of the network whenever two conditions are satisfied:

(i) A sampling of a uniform random variable from [0,1) produces a value less than or equal to the offered load λ, and,

(ii) (a) for simulating a system that has external queues, there is space in the corresponding queue for the packet, or (b) for those without external queues, there is space in the packet's destination output queue in stage one. A packet is considered *dropped* (or loss) if this condition is not met. A *dropped* packet is labelled idle and returned to the packet stack.

Before the packet is injected into the network, the *Start time* and the *Source address* are recorded, and the *Destination address* of the packet is assigned the address of a randomly chosen output link in stage three. Upon arrival of the packet at its destination output, it is pushed back into the *Packet Stack* after statistics are recorded.

4.5.2.2. *Switching in Stage One*

In the previous chapter, three routing methods are considered to route packets to output links in the first stage. In the case where external input queues are present, these methods are implemented for the simulator as follows:

- *Straight Through*: The head-of-queue packet at $Ext\_Queue[i]$ attempts to move forward to $Output\_Queue[i]$ in the switch.
- *Random Select*: A uniform random variable, *rand*, from [0,15] is obtained and the head-of-queue packet at $Ext\_Queue[i]$ attempts to move forward to $Output\_Queue[rand]$ in the switch.
- *Repeat Select*: An array of a permutation of numbers from 0 to 15, *perm*, is randomly selected. Then, according to the order in *perm*, the packet attempts to move forward to $Output\_Queue[perm[i]]$ (where $i = 0..15$) until a successful attempt is achieved, or until all $Output\_Queue$'s have been tried and failed.

In the case where there are no external input queues, each of the generated packets at each link (instead of the head-of-queue packet at an external queue) attempts to move forward using one of the above methods.

4.5.2.3. *Switching in Stage Two*

As described in the previous chapter, the design of switches in stage two allows queues to transmit more than one packet in each clock cycle (see Figure 4.4(a)). As more than one packet can be transmitted in each clock cycle, the processing order of packets within a queue is important. Since the queues in the next stage fill up as the packets in the queues in the current stage (i.e., stage two) are processed, the earlier a packet in a queue in stage two is processed, the larger is the chance that the packet will be forwarded.

Since the processing order of packets is important, it would be ideal if FIFO discipline is employed. However, for the sake of implementation simplicity, an "*implemented switch model*" which employs random discipline was used. Instead of using queues of size d (where d is the output-dilation of the crossbar switch), d queues of size 1 are used. Instead of processing queues with FIFO discipline, groups of d queues are processed in random order. The structures of the *"proposed switch model"* and the *"implemented switch model"* for System II are shown in Figure 4.4 [6].

Also, the dilations of crossbar switches used in these two structures are different from each other. In the *proposed switch model* (Figure 4.4(a)), 4 output-dilated crossbar switches are used, whereas, 1 output-dilated crossbar switches are used in the *implemented switch model* (Figure 4.4(b)). Table 4.1 on page 43 summarizes the characteristics of switches implemented in the simulator.

As mentioned in the previous chapter, the purpose of switching in stage two is to switch packets arriving at stage one to output queues [7] that connect to their destination switches in stage three. For example, in System II, a packet arriving at an input in *Switch*[2][0] with *Destination address* = 32 may attempt to move to any output queue that is connected to *Switch*[3][2]. That is, when considering the *implemented switch model*, the packet may attempt to move to *Output_Queue*[12], *Output_Queue*[13], *Output_Queue*[14] or *Output_Queue*[15] until an successful attempt is achieved, or until all these *Output_Queue*'s have been tried and failed. This is illustrated in Figure 4.5.

In fact, the indices of the *Output_Queue* to which a packet with Destination address = x may attempt to move can be calculated as follows:

[6] Tests were conducted using these two switch designs and we found that the performance differences between these switches are negligible for the systems under consideration.

[7] These output queues are *virtual* if switches in stage two are unbuffered. Otherwise, they are *real*.

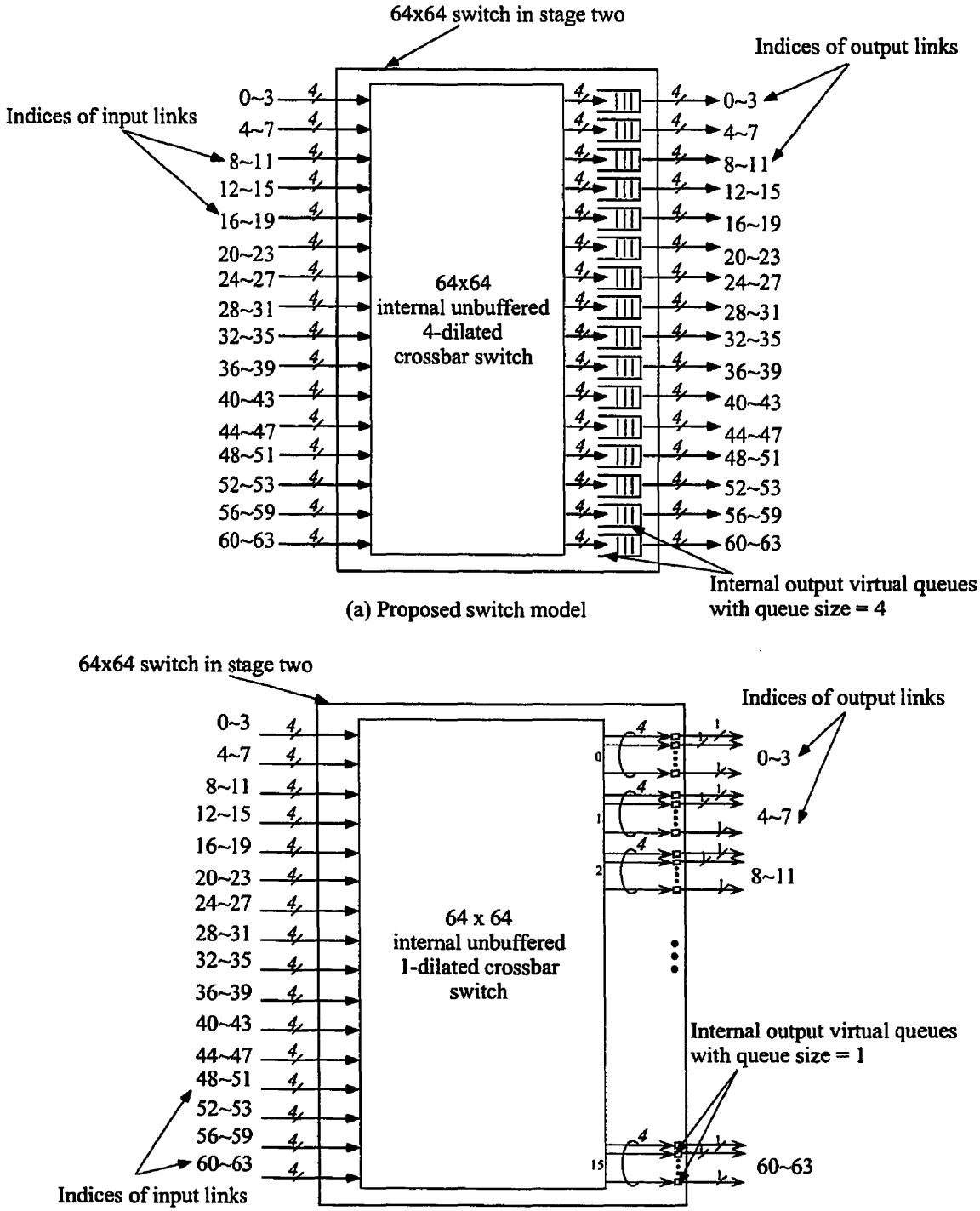
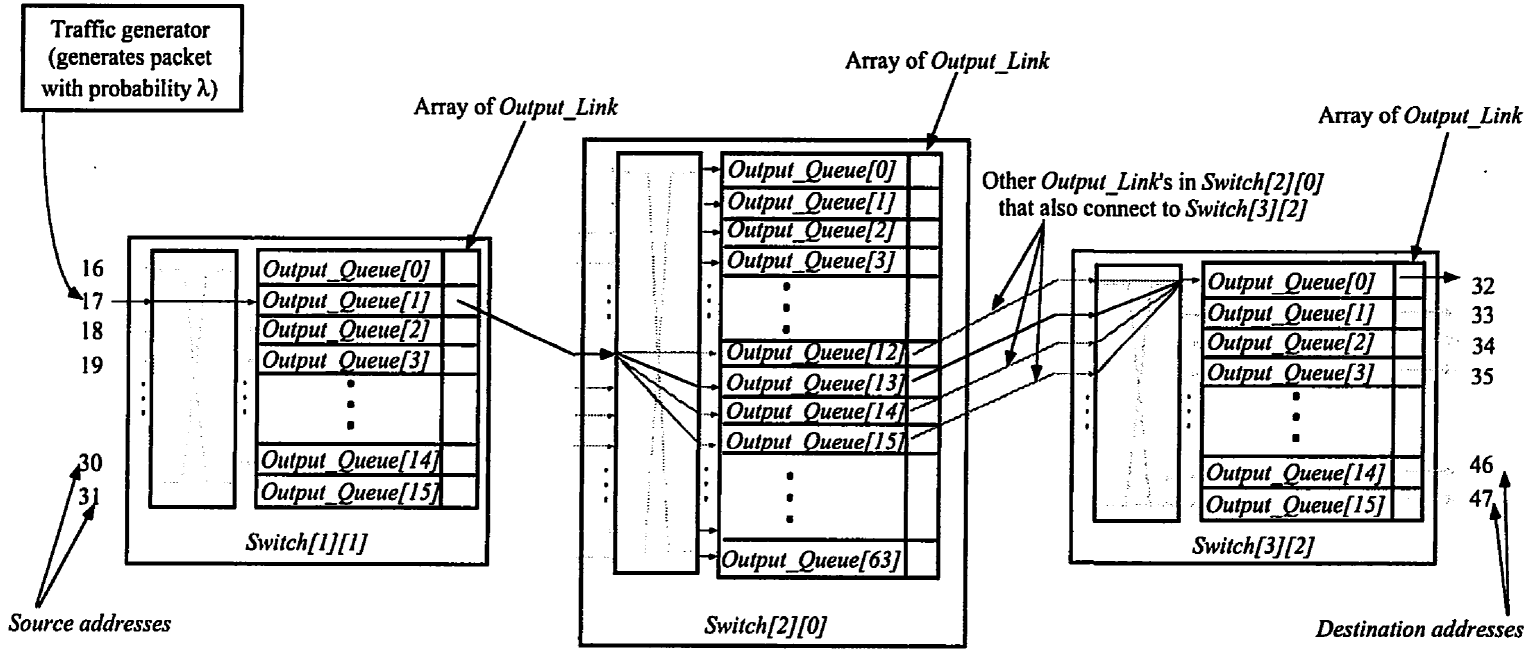


FIGURE 4.4. Software architectures of unbuffered crossbar switches for System II.



(1) A packet is generated at input link #17 with probability λ.
(2) *Destination address* = 32 is randomly choosen for the packet
(3) The packet is forwarded to *Output_Queue[1]* by *Random Select.*

(4) The packet arrives at stage two and attempts to move to *Output_Queue[12], Output_Queue[13], Output_Queue[14]* or *Output_Queue[15]* since these output queues' are connected to the packet's destination switch. In this example, the packet is forwarded to *Output_Queue[13].*

(5) The packet arrives at stage three and attempts to move to *Output_Queue[0]* since this leads to the packet's destination.

FIGURE 4.5. Example of Packet Routing.

Computing the range of index of *Output_Queues* that a packet arriving *Switch*[2][i] with Destination address = x may attempt to move to.	
System I	((x div 16) × 16) to (((x div 16) × 16) + 15)
System II	((x div 16) × 4) to (((x div 16) × 4) + 3)
System III	(x div 16) only

4.5.2.4. *Switching in Stage Three*

Packets are switched to the output queues that lead to their destinations in stage three. In System II, for example, a packet with *Destination address* = 32 arriving at an input in *Switch*[3][2] will attempt to move to *Output_Queue*[0] in the switch. This example is illustrated in Figure 4.5.

In general, a packet with *Destination address* = x arriving at an input in a switch in stage three will attempt to move to *Output_Queue*[x *mod 16*] of the switch.

Summary of the Switches in the Simulator			
	Stage 1	Stage 2	Stage 3
Type of Switch	dilated crossbar	dilated crossbar	dilated crossbar
# of input links (System I, System II, System III)	16, 16, 16	256, 64, 16	16, 16, 16
# of output links (System I, System II, System III)	16, 16, 16	256, 64, 16	16, 16, 16
Input-dilation	1	1	1
Output-dilation	1	1	1
# of External input queues	0 or 16	0	0
# of Internal output queues (System I, System II, System III)	16, 16, 16	256, 64, 16	16, 16, 16

TABLE 4.1. Summary of the characteristics of switches in the simulator.

4.6. Modelling Parallel Events in a Serial Simulator

Since we are implementing the simulator using a serial processor platform, it is impossible to simulate the activity of all packets in the network simultaneously. In fact, only one

queue can be processed at a time. In this section, we will discuss how the processing order can be arranged in order to achieve a correct simulation.

4.6.1. Processing Order of Switches within a Stage

The processing order of switches within a stage is important. The earlier the switch is processed, the larger is the chance the packets in that switch will be forwarded. In order to avoid this bias, the processing order of switches within a stage is performed in random order.

4.6.2. Processing Order of External Queues in Stage One

The processing order of queues within a switch is also important. The earlier the queue is processed, the larger is the chance the packet in the queue will be forwarded. In order to avoid bias, the processing order of output queues within a switch is performed in random order. Similarly, the processing of external queues that connect to the same switch are also done in random order.

4.6.3. Stage Processing Order

For the switching network with unbuffered switch(es) in stage two, the simulator will process the output queues in stages 3, 1, 2, and finally, the external queues in stage one. For the switching network with buffered switch(es) in stage two, the simulator will process the output queues in stages 3, 2, 1, and finally external queues in stage one. If stages are processed in order of increasing stage numbers, we say that the stages are processed in *forward* order. On the other hand, if stages are processed in order of decreasing stage numbers, we say that the processing is in *reverse* order. We will first study the impact of processing stages in each way (i.e., forward and reverse order) and then we will explain the reason for choosing *reverse* order for systems with buffered switch(es) in stage two and the reason for choosing a *combination* of both for systems with unbuffered switch(es) in stage two.

4.6.3.1. *Processing Stages in Forward Order*

Suppose the simulator processes the external queues first, followed by stages 1, 2 and 3. Consider the case when a queue in stage i ($i \leqslant 3$) is full at the beginning of a clock tick. No packet can be added into that queue even if the packet at the head of the queue leaves in the same clock tick, thereby, creating a space in that queue. Because stage *i-1* is processed *before* stage i, packets from stage *i-1* will attempt to enter the queue *before* the

packet at the head of the queue has attempted to leave for another queue. Therefore, this processing order will lead to unnecessary blocking.

4.6.3.2. *Processing Stages in Reverse Order*

Suppose the simulator processes stages 3, 2, 1, followed by the external queues. Because packets in stage i will attempt to leave *before* packets attempt to enter from stage *i-1*, this processing order will not have the problem caused by processing stages in forward order. Moreover, because packets always enter queues that are already processed, they will stay in the queues for at least one clock tick. For systems with unbuffered switches in stage two, since packets in stage two must leave the *virtual queues* in the same clock cycle in which they arrive at the queues, this processing order cannot be used in this case. However, if all switches in the network are buffered (i.e., when the buffering option for stage two is chosen), this processing order must be used.

4.6.3.3. *Processing Stages with Combination Order*

In this simulator, queues are finite and packets have to stay in the queues in stage one and three for at least one clock tick, so processing these stages in reverse order should be the choice. However, for systems with unbuffered switches in stage two, packets have to make an attempt to enter the virtual queues in stage two and then leave in the same clock tick. To implement this, stage one and two have to be processed in forward order. This processing order will not cause the problem discussed in Section 4.6.3.1 because all packets in stage two will be deleted by the end of each clock tick, which means all virtual queues will be emptied before packets attempt to enter. Therefore, for system with unbuffered switches in stage two, the processing order of stages will be: 3,1,2 and followed by external queues.

4.7. Statistical Evaluation

Each set of experimental data was simulated for twenty different values of offered load (λ, probability that a packet is created in each inlet per clock cycle) from 0.05 to 1. For each λ, nine sub-runs, with each sub-run having at least 400,000 packets transmitted through the network, were simulated. The first sub-run of each simulation was discarded to avoid

transients in the traffic pattern. This method of discarding the transient part of the simulation is referred to as *method of sub-runs* [8] [7, 14]. The results from the remaining eight sub-runs were used to compute statistical information.

After each simulation, statistical data such as average packet delay, average loss, average throughput, average head of external queue delay, and their confidence intervals are written to data files.

The throughput TP_i, delay $DELAY_i$, loss rate $Loss_i$, and delay for head of external queue packet $EXT\_HOQ\_DELAY_i$ for each sub-run i are calculated as follows:

(4.7.4.1)
$$TP_i = \frac{\frac{1}{2} \cdot (\text{total number of packets transmitted} + \text{total number of packets received})}{(\text{total number of ports}) \cdot (\text{cycles in subrun } i)}$$

(4.7.4.2)
$$DELAY_i = \frac{\sum \text{age of packets received in subrun } i}{\text{total number of packets received in subrun } i}$$

(4.7.4.3)
$$LOSS_i = \frac{\text{total number of packet lost in subrun } i}{(\text{total number of ports}) \cdot (\text{cycles in subrun } i)}$$

(4.7.4.4)
$$EXT\_HOQ\_DELAY_i = \frac{\sum \text{time of packets spent in head of external queue } i}{\text{number of packets transmitted in subrun } i}$$

From all the sub-runs, we calculate the mean throughput (TP), mean delay (DELAY) and mean loss rate (LOSS) with the standard deviation of the mean as a measure of error.

4.8. Description of Major Functions

The following is a precise description of some major functions used in the simulator.

do_one_pass
assumes: the network is initialized properly.
modifies: *Packet Stack*, *Ext_Queues*, and *Output_Queues* in all stages.
purpose: simulates all the switches for one clock cycle. Packets are generated and inserted into the *Ext_Queues* (if any) and head-of-line packets attempt to move forward. The order

[8] Although, without explicitly referring to the method in the *method of sub-runs*, Mitrani describes this method in [14] and so does Kheir in [7].

for simulating the processing of switches and queues is done as described in Section 4.6.

traffic_generation
assumes: no external input queues before first stage
modifies: *Output_Queues* in stage one and *Packet Stack*
purpose: generates packets at each input link in stage one with a probability of λ (refer to Section 4.5.2.1). Generated packets attempt to be forwarded to *Output_Queues* in stage one (refer to Section 4.5.2.2).

ext_traffic_generation
assumes: external input queues before first stage
modifies: *Ext_Queues*, *Output_Queues* in stage one, and *Packet Stack*
purpose: attempts to forward head-of-queue packet in the *Ext_Queues* to stage one (refer to Section 4.5.2.2). Then, packets are generated at each input link in stage one with probability of λ (refer to Section 4.5.2.1). Generated packets attempt to be forwarded to their correspond *Ext_Queues*.

add_pkt_to_dilated_switch(swtch:integer)
modifies: *Output_Queues* in stage one and stage two
purpose: head-of-queue packets in *Output_Queues* of *Switch[stage one][swtch]* attempt to move forward to *Output_Queues* in stage two (refer to Section 4.5.2.3).

rm_pkt_from_dilated_switch(swtch:integer)
modifies: *Output_Queues* in stage two and stage three
purpose: head-of-queue packets in *Output_Queues* of *Switch[stage 2][swtch]* attempt to move forward to *Output_Queues* in stage three (refer to Section 4.5.2.4).

simulate_switch(stage:integer, swtch:integer)
modifies: *Output_Queues* in *Switch[stage][swtch]* and *Output_Queues* in stage *stage*+1
purpose: head-of-queue packets in *Output_Queues* in *Switch[stage][swtch]* attempt to move forward to *Output_Queues* in stage *stage+1*. If *stage* equal to last stage of the switching network (stage three in this case), head-of-queue packets are removed from the network.
note: currently this function is only used for removing head-of-queue packets in stage three

(last stage). However, this function could be use in future applications.

verify_topology
modifies: *Output_Link*s of all switches in the network.
purpose: sets up the interconnections between the queues and nodes in the network (refer to Section 4.5.1).

next_out_link(dst:integer, stage:integer, link:integer)
returns: outlink:integer
purpose: computes the index of the next destination *Output_Queue* of a packet (refer to Section 3.3.5).

add_pkt(q:pointer to a queue record)
modifies: queue that is pointed to by q and the *Packet Stack*
purpose: pops a packet from *Packet Stack* (refer to Figure 4.3) and adds it to the tail of the queue pointed to by q.

rm_pkt(q:pointer to a queue record)
modifies: queue that is pointed to by q and *Packet Stack*
purpose: removes the head-of-queue packet in the queue pointed to by q and pushes it back to the *Packet Stack* (refer to Figure 4.3).

4.9. Pseudo-code for the Simulator

Below is the pseudo-code for the software simulator. This simulator is implemented using the C programming language.

```
Simulator(system, input_queue_size, external_queue_size)
/*
Purpose: A program that simulates the optical backplane and the PCBs that connect to the
backplane as a 3-stage switching network.

Parameters:
system: the configuration of the network chosen (System I, II, or III).
input_queue_size: external input queue size for the network.
output_queue_size: internal output queue size for the network.
*/
{
Initialize data structure of the network with the selected system configurations;
verify_topologies();                          /* Set up the interconnections of the network. */

   /* Simulate the system for twenty different offered loads from 0.05 to 1. */
   for (each offered load){
   Initialize statistical variables;

      /* Simulate nine sub-runs (trials) for each load. */
      for (each trial){

         /* Each sub-run must have at least MIN_PACKETS transmitted through the network
         before terminates. */
         while (total number of packets received < MIN_PACKETS){

         do_one_pass;                         /* Simulate the operation of switches for
                                                 each stage for one clock cycle. */

         /* Generate traffic to each input link. */
         if (external_queue_size > 0)
         ext_traffic_generation();            /* Function to generate traffic when external
                                                 queues are present. */

         else
         traffic_generation();                /* Function to generate traffic when there
                                                 are no external queues. */

         } /* end while */

      } /* end for */
      Compute and store statistical results to files;

   } /* end for */

} /* end Simulator */
```

4.10. Organization of the Software Simulator

The software simulator is organized into ten files, which are:

- *output_q.c*: contains the top-level control program which initiates the entire simulator. Moreover, all global variables are declared in this file.
- *engine.c*: contains the functions which are specific to the 3-stage simulation.
- *vvfast-packet.c*: contains the functions for manipulating packets and the *packet stack*.
- *stat.c*: contains the functions which compute the statistical data.
- *out_files.c*: contains the functions which manipulate the output data files.
- *std_in_out.c*: contains the functions which manipulate the standard input and output.
- *fast-double-nmy-math.c*: contains the functions to generate random numerical permutations.
- *my-math.h*: contains functions for generating random numbers.
- *user.h*: contains all user compile-time variable.
- *makefile*: contains the commands for compiling the software with the gcc compiler.

4.11. Verification Criteria

It is difficult to determine the correctness of the simulator with arbitrary test data. Nevertheless, it is easy to hypothesize the behavior of the simulator with certain extreme configurations (boundary cases). For example, by setting all destination addresses of packets to their source addresses and selecting the *Repeat Select* strategy for switching packets in stage one, contention of space in any queues in the system will be avoided and the simulation should have 100% normalized throughput and 0% loss rate. The correctness of the path that a packet chooses can be determined by tracing the path using the debug mode and running the simulation under the *gdb* environment. When the debug mode is on, the activity of each packet will be printed out on the screen of the terminal that is running the program, thereby, allowing its correctness to be determined manually. The gdb environment is used to set up *breakpoints* [9] so that the simulation can be traced step by step.

Furthermore, an analytic model based on this system was used to verify the simulator's correctness [**18**]. The results generated from this model give the approximate results that should be expected from the simulator.

[9] Breakpoints makes the programs to stop at certain points.

4.12. User Interface

This simulator uses a simple text interface. It prompts the user to enter the values for the run-time variables for the configuration of the simulation. These variables include:

- System configuration (System I, II or III)
- External input queue capacity
- Normal/Debug mode (ON/OFF)

4.13. Compile Time Variables

Other than the run time variables that users can choose for the configuration of a simulation, many compile time variables are used to allow the simulator to run with special characteristics. Some of these are:

- Flags to indicate which switching strategy is used for stage one (refer Section 4.5.2.2). Notice that only one of the three flags can be enabled in each simulation. The flags are:
  - *ST_THRU*: flag for the *straight through* strategy.
  - *RANDOM*: flag for the *random select* strategy.
  - *REPEAT_SELECT*: flag for the *repeat select* strategy.
- *PRUNE*: An option that restricts the number of output queues used in stage one and the output-dilation of switch(es) in stage two. *PRUNE* must be an integer less than or equal to 16. It represents the number of output queues that can be used in each stage one switch and the output-dilation of stage two switch(es). $PRUNE = 0$ indicates no restriction is applied.
- *BUFFERED_2NDSTG*: An option to allow buffering in stage two when it is enabled. When this option is enabled, another compile time variable *STAGE2_OQUEUE_SIZE* must be set to the desired queue capacity for output queues in stage two.
- *SELECT_QSIZE*: An option to allow different queue sizes for output queues in stage one and stage three. When this option is enabled, the compile time variables *STAGE1_OQUEUE_SIZE* and *STAGE3_OQUEUE_SIZE* must be set to the desired queue capacity for output queues in stage one and stage three, respectively. When this option is disabled, the output queue sizes of stage one and stage three will be assumed to be equal. Since no specific output queue capacity is given, the simulator will be simulated for seven different output queue sizes (1, 2, 4, 8, 16, 32, and 64) by default.

- *TRIALS*: The number of sub-runs for each simulation. The default value is 8.
- *MAX_TIME*: The maximum number of clock-ticks allowed for each sub-run. The default value is 1,000,000,000.
- *MIN_PACKETS*: The minimum number of packets transmitted through the network in each sub-run. The default value is 400,000.

These compile time variables are located in file *user.h*.

4.14. Compiling and Executing the Software Simulator

After setting all the compile-time variables in file *user.h*, the software is ready to be compiled. To compile the software, type *make*. After compilation, an executable file called *out* is generated. To run the software, type *out*.

The text interface as described in Section 4.12 will appear and the user is required to enter the value for the run-time variables. A sample output from the simulator is given in Appendix B.

4.15. Output of the Software Simulator

If debug mode is set, the activity of each packet will be printed on the screen of the terminal that is running the program.

Statistical results generated from a simulation are stored in data files. All the data files contain a 2-dimension array in MATLAB format. The rows of the array correspond to data of varying output queue sizes and the columns of the array correspond to the data of varying offered loads. The files are:

- *dlay.m*: contains the average delay for various offered loads and queue sizes in the simulation.
- *delay_ci.m*: contains the confident intervals of the delay reported in dlay.m.
- *throughput.m*: contains the average throughput for various offered loads and queue sizes.
- *throughput_ci.m*: contains the confident intervals of the throughputs reported in throughput.m.
- *loss.m*: contains the average loss for various offered loads and queue sizes in the simulation.
- *loss_ci.m*: contains the confident intervals of the loss rate reported in loss.m.

The MATLAB script files, *delay_graph.m* ,*throughput_graph.m* and *loss_graph.m*, are written to plot the delay, throughput and loss rate of a simulation, respectively. Graphs

generated by running these script files are displayed on the screen of the terminal that is running the program and are also saved as Encapsulated Postscript files under the same name as the data files, but with the *.eps* suffix.

4.16. Maintenance and Possible Extension to the Software Simulator

In this thesis, much effort was devoted to create a well-structured software simulator which could be easily revised and expanded in order to accommodate future modifications to the backplane. In order to facilitate maintenance in the future, several possible extensions to the software simulator are discussed.

- *Dimension of Switches*: The dimension of switches in stage one and three can be adjusted by changing the value of the global variable, *lines_per_PCB*, which is initialized in function *get_data* in file *std_in_out.c*. However, the value of *lines_per_PCB* must be divisible by the number of switches in stage two.
- *Number of Switches*: The number of switches in stage one and three can be adjusted by changing the value of the global variable, *no_of_PCB*, initialized in function *get_data* in file *std_in_out.c*. However, the value of *no_of_PCB* must be divisible by the dilation of the switch in stage two in the *proposed switch model.*
- *Interconnectivity of the Switching Network*: The interconnectivity of the switching network can be changed by replacing the function *verify_topology* located in file *engine.c* with an appropriate one.
- *Dilation of Switches in Stage Two*: Currently, the simulator only allows switches in stage two to have dilations of 1, 4, and 16. The dilation of switches in stage two can be adjusted by changing the value of the global variable, *dilation*, initialized in function *get_data* in file *std_in_out.c*. However, the value of *dilation* must be an integer that can evenly divide *no_of_PCB*.

4.17. Approximate Duration of the Research

This research was started on June 1995, and took one and a half years to finish. The first two months (June and July, 1995) of the research were spent for examining the code of the banyan network simulator and reading related papers. During the third month (August 1995), the banyan network was restructured and documented. Then, the next two months (October and November, 1995) were spent implementing the 3-stage crossbar network simulator which consists of approximately 2,000 lines of code. The following five

months (December, 1995 to April, 1996) was spent validating the simulator, adding user options and generating preliminary statistical results. The summer months after that (May to August 1996) were spent writing and debugging the analytic model. Verification of the simulator with the analytic model was also done during that period. The last four months (September, 1996 to December, 1996) was spent gathering final results and writing drafts for this thesis.

4.18. Summary

The design of the software simulator for simulating the network systems described in Chapter 3 was presented in this chapter. The simulator model development approach, statistical information collection, and set-up of the simulation parameters were described. The pseudo-code of the software simulator, validation criteria of the software, and possible extensions of the software were also given.

CHAPTER 5

Experimental Results

This chapter examines the performance of the three switching systems (Systems I, II, and III) described in Chapter 3. First, the simulation results of the three systems are compared and examined. Then, an extended version of System I, which has sixteen times less servers but with each server having a service rate that is sixteen times faster than those in the original System I, is described and simulated. The results obtained from the simulation are then compared against those which are obtained for the original System I. Finally, the performance of System III under three different selection strategies (*Straight Through*, *Random Select* and *Repeat Select*) for stage one is examined. All results presented in this chapter are obtained from simulations that are configured such that:

- the total queue capacities of the systems are equal,
- the total queue capacities of the PCBs (switches in stages one and three) are equal,
- the optical switches (in stage two) of each system are buffered, and
- each system has the same total queue capacity in stage two.

All results in this chapter are presented over normalized offered load, λ, from 0 to 1. λ is expressed in terms of number of packets per simulated clock cycle. Each simulated clock cycle is defined as the length of time for the transmission of a single packet. These normalized results give us an indication of the efficiency of each simulated system.

5.1. The Three Systems

As described in Chapter 2, the optical switches in the backplane are created by SPAs. As the switch becomes bigger, the SPA becomes more complex. The three systems that have been described in Chapter 3 represent the spectrum of the design of SPAs. System I contains a big switch (256×256, *16 output-dilated*), so the SPAs used to create this switch

are complex and expensive; System II contains four moderate switches (64 × 64, *4 output-dilated*), so the SPAs used to create these switches are of moderate complexity and cost; and System III contains sixteen small switches (16 × 16, *1 output-dilated*), so the SPAs used to create these switches are the least complex and relatively inexpensive. Therefore, comparing the performance of these three systems will allow us to know if it is effective to build a complex switch instead of several less complex ones.

5.1.1. Simulation Configurations

All simulated systems have no external input queues, and the output queue capacities for both stage one and stage three are 4. Since all three systems have 16 switches (with each switch having 16 queues) in both stage one and stage three, the total queueing capacity in each of these stages is $16 \times 16 \times 4 = 1024$ packets. Also, recall that for stage two, there are 1, 4, and 16 switches in Systems I, II, and III, respectively, with each switch having 16 queues [1]. In the simulations, the output queue capacities for stage two are 32, 8, and 2 in Systems I, II, and III, respectively. Therefore, the queueing capacity in stage two for all three systems is 512 packets [2]. Consequently, since the queueing capacities of stages one and three are also the same for all three systems, as described above, the total queueing capacities of the three systems are equal. For routing packets in stage one, the *Straight Through* method (refer to Section 4.5.2.2) is used. Table 5.1[3] summarizes the configurations of the three systems. The values of the *run-time* and *compile-time* variables for these three simulations are given in Table A.1 in the appendix.

5.1.2. Simulation Results

Figure 5.1 on page 58 shows the throughput, delay, and loss rate observed during the simulations of Systems I, II, and III. Error-bars indicate the width of the standard deviation about the mean[4].

We expect System I to have the best performance among the three systems because it has the highest output-dilation in stage two, which allows the resources (queues and links) to be allocated efficiently. On the other hand, we expect System III to have the worst performance because traffic entering stage two is split among 16 small switches, which may lead to resources not being allocated as efficiently as in Systems I and II.

---

[1] "*Proposed Switch Model*" is assumed here (refer to Section 4.5.2.3).

[2] System I: $1 \times 16 \times 32 = 512$; System II: $4 \times 16 \times 8 = 512$; and System III: $16 \times 16 \times 2 = 512$.

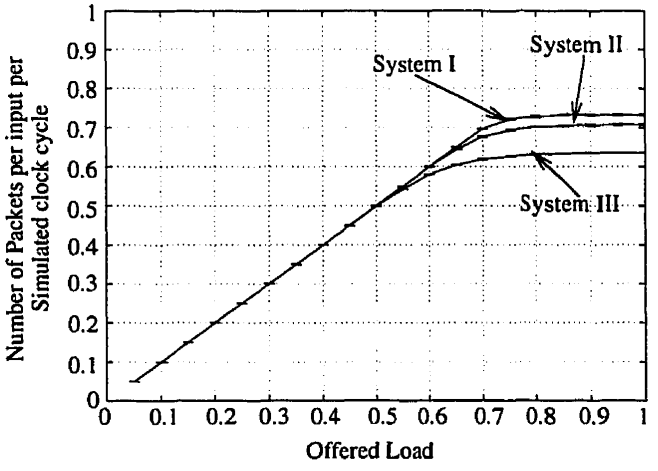
[3] "*Proposed Switch Model*" is assumed in stage 2 (see Section 4.5.2.3).

[4] Error-bars are not shown for the loss rate. This is due to the inability of MATLAB to display error-bars when an axis is displayed in logarithmic scale. However, the average standard deviation for the loss rate is less then 10^{-3}.

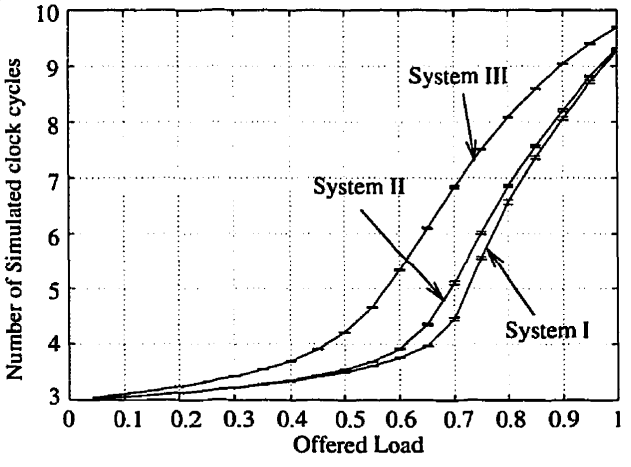
Configuration for simulating Systems I, II, and III.			
System I	stage 1	stage 2	stage 3
# of switches	16	1	16
Size of switch	16 × 16	256 × 256	16 × 16
input-dilation of switch	1	1	1
output-dilation of switch	1	16	1
# of queues per switch	16	16	16
Size of queues	4	32	4
Total queueing capacity	1024	512	1024
System II	stage 1	stage 2	stage 3
# of switches	16	4	16
Size of switch	16 × 16	64 × 64	16 × 16
input-dilation of switch	1	1	1
output-dilation of switch	1	4	1
# of queues per switch	16	16	16
Size of queues	16	8	16
Total queueing capacity	1024	512	1024
System III	stage 1	stage 2	stage 3
# of switches	16	16	16
Size of switch	16 × 16	16 × 16	16 × 16
input-dilation of switch	1	1	1
output-dilation of switch	1	1	1
# of queues per switch	16	16	16
Size of queues	4	2	4
Total queueing capacity	1024	512	1024

TABLE 5.1. Configuration of the three systems.

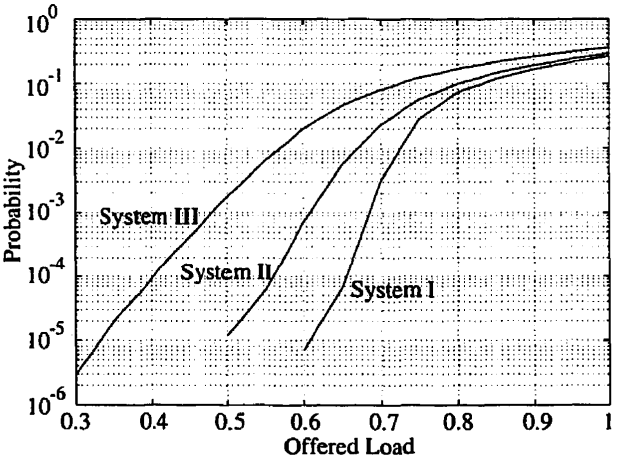
The results from the simulations show exactly the behavior just described. System I has the highest average throughput, the lowest average delay, and the lowest loss rate; System III has the lowest average throughput, the highest average delay, and the highest loss rate; and System II's performance lies somewhere in between.



(a) Average throughput vs. offered load



(b) Average delay vs. offered load



(c) Average loss rate vs. offered load

FIGURE 5.1. Throughput, delay and loss rate of Systems I, II, and III.

5.1.2.1. *Throughput*

At light load, the rate of increase in throughput of the three systems increases proportionally with the offered load λ since offered packets are almost always accepted by the network. As λ becomes larger, there is a decrease in the rate of increase in throughput

because packets are occasionally not accepted, due to most queues in stage one being *full*. The throughputs stabilize and the systems reach their maximum throughputs, as the offered load approaches 1. At full load ($\lambda = 1$), System I has the highest throughput of 73.2%, System II has a slightly lower throughput of 70.7%, and System III has the lowest throughput of 63.5%.

5.1.2.2. *Delay*

The average delays are consistent with the throughput. The average delays are expressed as the average number of simulated clock cycles each packet spends in the switching system. In this set of simulations, all systems have a delay of less than 10 simulated clock cycles for all values of λ. Since all packets have to stay in each stage for at least one simulated clock cycle, the delay of the three systems is at least 3 simulated clock cycles. The minimum delay of the three systems occurs at $\lambda = 0.05$, as shown in Figure 5.1.

It is observed that, at light loads ($\lambda \leqslant 0.4$), the average delays of System I and System II are approximately equal. This behavior is also observed in the analytic model (see Figure A.1 in Appendix A). Therefore, we can conclude that the 4-dilated switches used in System II are as powerful as the 16-dilated switch used in System I, under light loads. As the load becomes heavier, the performance of System II starts to degrade.

Furthermore, it is observed that as λ approaches 1, the average delays of System I and System II converge once again. The average delay is almost the same (approximately 9.3 simulated clock cycles) for System I and System II at $\lambda = 1$. This behavior can be explained by the following. The queues in stage two of System II are smaller than those of System I. Therefore, at heavy loads, queues in stage two of System II have a higher probability of becoming full than those of System I. As most queues in stage two of System II are full, backpressure [5] to queues in stage one is created, which causes packet overflow to increase. Hence, the probability of overflow for System II will be higher, resulting in a higher loss rate. This also suggests that the total number of packets within System II is smaller than the total number within System I. According to Little's Law [6] [2], the result is that their delays, each of which corresponds to the ratios of *number of packets in system* to *throughput* of the systems, remain close to each other.

[5] Backpressure, in which by means of suitable backward signalling, the number of packets actually switched to each output queue is limited to current storage capacity in the corresponding output queue. In this case, all other HOL packets remain stored in their respective output queues in stage one.

[6] $Delay = \frac{Number\ of\ packets\ in\ system}{Throughput}$

5.1.2.3. *Loss Rate*

The average loss rates are also consistent with the throughput. For System I, packet loss is detected only when $\lambda \geqslant 0.6$; for System II, packet loss is detected when $\lambda \geqslant 0.5$; and, for System III, packet loss is detected at a light load of $\lambda \geqslant 0.3$. Since queues in stage two of System III are much smaller than those of Systems I and II, packet loss in this system is higher at light loads.

5.2. System I with Different Server Speeds

In this section, we compare the performance of two versions of System I, each with a different server speed. We chose to analyse System I, as opposed to System II or System III, because the results obtained for this system allow us to see more clearly the performance difference between the two versions. The busses connecting stages one and two, and stages two and three are 256 low bandwidth busses in the first version and 16 higher bandwidth busses in the second version; as such, the second version has a faster server speed but a fewer number of servers. To avoid confusion, we refer to the first version as *Original System I*, since it is the same as System I that was described in Chapter 3; and we refer to the second version as *Pruned System I*, since the 256 low bandwidth busses are *pruned* to 16 higher bandwidth busses.

5.2.1. Original System I versus Pruned System I

As described in Chapter 2, in *Original System I*, each communication link is 8 bits wide, so that a byte can be transmitted in each *electrical* clock cycle. Assuming the bit rate of the backplane is 200 Megahertz (MHz), each clock cycle is 5 nanoseconds (nsec). Assuming each packet is 64 bytes long [7], the backplane will take 64 *electrical* clock cycles, or $64 \times 5 = 320$ nsec, to move a packet from one stage to the next. As mentioned before, each simulated clock cycle is defined to be equal to the length of time for the transmission of a single packet. Therefore, each simulated clock cycle in Original System I is equivalent to 64 *electrical* clock cycles or 320 nsec.

Figure 5.2 shows the architecture of Pruned System I. In this system, all packets from the 16 traffic sources arriving at each switch in stage one are multiplexed to a *single* queue, resulting in sixteen times fewer servers than in stage one of Original System I. Therefore, there are a total of 16 inputs to the switch in stage 2. With output-dilation of 1, this

[7] We assume that the backplane transmits Asynchronous Transfer Mode (ATM) packets (53 bytes) together with 4 bytes header and 4 bytes error-checking bits. This makes a total of 61 bytes. However, for the sake of simplicity, we round up the length of each packet to 64 bytes.

switch has 16 output links, with each link connecting to a switch in stage three. Finally, traffic from each of these links in stage three are de-multiplexed to 16 outputs. With this architecture, only 16 high bandwidth busses are needed in the system. If the number of bit channels used is the same as in Original System I (i.e., $256 \times 8 = 2048$ bits), each of these high bandwidth busses will be $2048/16 = 128$ bits (16 bytes) wide.

For Pruned System I, since each bus is 16-bytes wide, 16 bytes can be transmitted within each *electrical* clock cycle, and a 64-byte packet will take 4 *electrical* clock cycles, or $4 \times 5 = 20$ nsec, to be moved. Therefore, each simulated clock cycle of Pruned System I is equivalent to 4 *electrical* clock cycles, or 20 nsec, so that each server in Pruned System I has a sixteen times faster service rate than that of Original System I. Furthermore, electrical switches are not needed in stage one or three; instead, *multiplexers* are used in stage one and *de-multiplexers* are used in stage three. Also, these multiplexers and de-multiplexers must process packets at a speed that is sixteen times faster than that of electrical switches used in Original System I.

5.2.2. Simulation Configurations

In Pruned System I, only a single queue, instead of 16 output queues, is used in each switch in stage one (i.e., traffic from multiple inputs is statistically multiplexed into a single queue). In order to maintain the same number of stage one buffers as in Original System I, the capacity of stage one queues is sixteen times larger than those in Original System I. Table 5.2 on page 63 summarizes the configurations of the two systems. For routing packets in stage one, the *straight through* method is used. The values of *run-time* and *compile-time* variables of the simulation of Pruned System I are given in Table A.2 in Appendix A. For Original System I, the same values for the simulation of System I in Section 5.1 are used.

5.2.3. Adjustments for the Simulation of Pruned System I

As discussed in the previous section, one simulated clock cycle of Pruned System I represents 4 electrical clock cycles, whereas one simulated clock cycle of Original System I represents 64 electrical clock cycles. Therefore, one simulated clock cycle in Original System I is equivalent to 16 simulated clock cycles in Pruned System I. In order to make the results of Pruned System I comparable to Original System I, the arrival rate (offered load) must be equal for both systems in terms of electrical clock cycles. Since Pruned System I's simulated clock is 16 times faster than Original System I's, its offered load λ_{PRUNE} (expressed in packets per simulated clock cycle in Pruned System I) must be 16

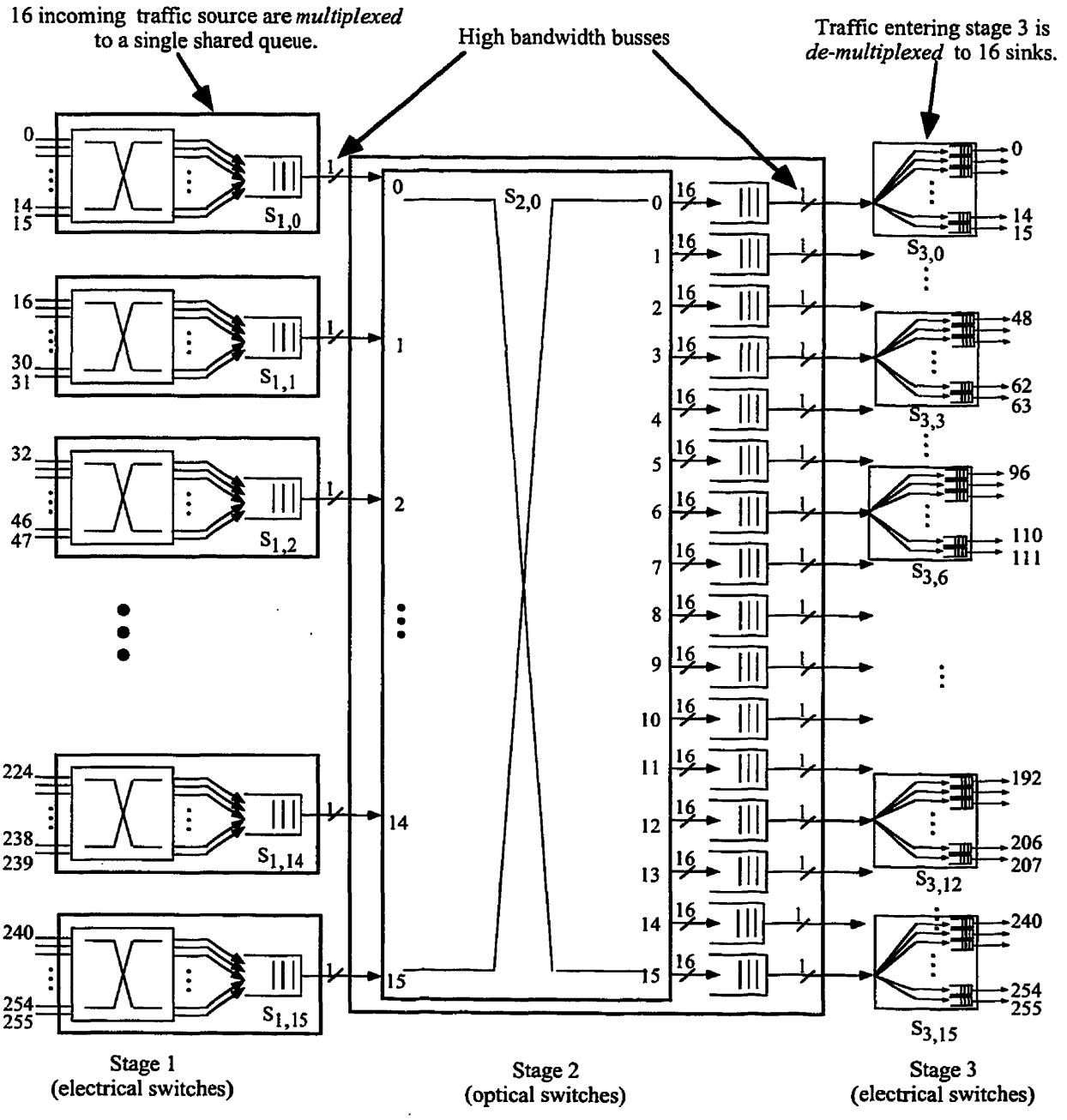


FIGURE 5.2. Structure of Pruned System I [18].

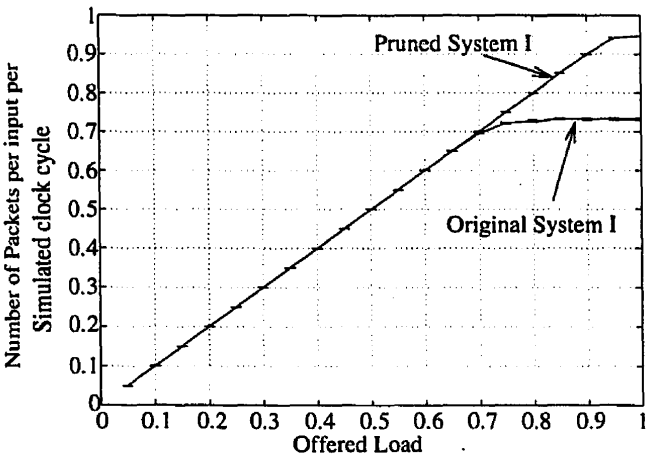
Configuration for Simulating Two Different Server Speeds of System I.			
Original System I	stage 1	stage 2	stage 3
# of switches	16	1	16
Size of switch	16 × 16	256 × 256	16 × 16
Input-dilation	1	1	1
Output-dilation	1	16	1
# of queues per switch	16	16	16
Size of queues	4	32	4
Total queueing capacity	1024	512	1024
Pruned System I	stage 1	stage 2	stage 3
# of switches	16	1	16
Size of switch/ multiplexer/de-multiplexer	16-1 *multiplexer*	16 × 16 *switch*	1-16 *de-multiplexer*
Input-dilation	1	1	1
Output-dilation	1	1	1
# of queues per switch	1	16	16
Size of queues	64	32	4
Total queueing capacity	1024	512	1024

TABLE 5.2. Configuration for simulating two different server speeds of System I.

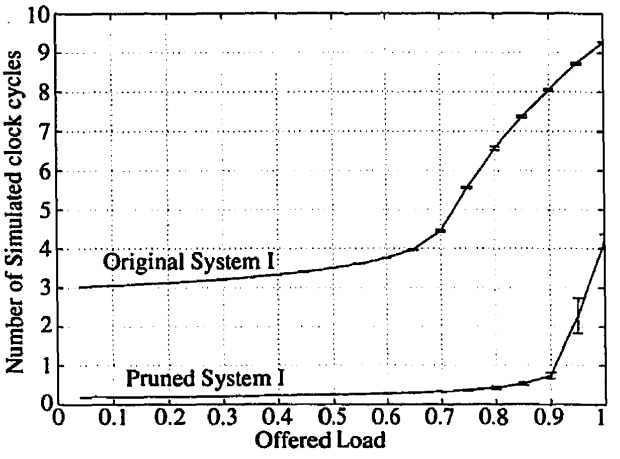
times lower than Original System I's offered load $\lambda_{ORIGINAL}$ (expressed in packets per simulated clock cycle in Original System I).

As a result, in order to fairly compare the throughput of both systems, we multiply the throughputs of Pruned System I by 16 so that the throughputs of both systems will be in terms of the same unit, namely, average number of packets transmitted through the system per simulated clock cycle of Original System I. For the same reason, the loss rate of Pruned System I must also be multiplied by 16.

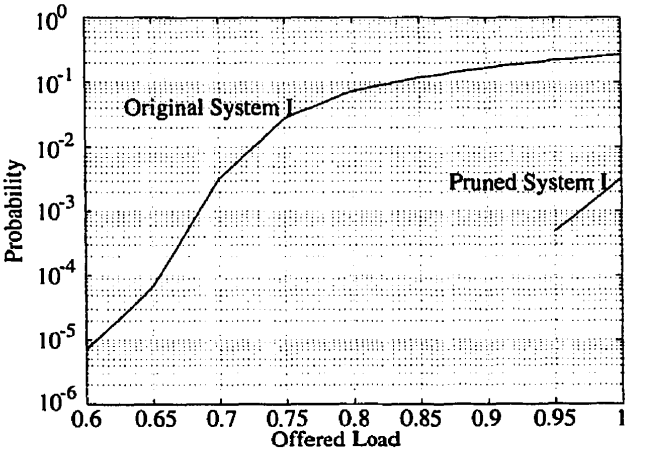
Similarly, we must divide the delays of Pruned System I by 16 so that the delays of both systems can be compared in terms of the same unit, namely, average number of simulated clock cycles of Original System I that a packet spends in the network.



(a) Average throughput vs. offered load



(b) Average delay vs. offered load



(c) Average loss rate vs. offered load

FIGURE 5.3. Throughput, delay and loss rate of System I with different server speed.

5.2.4. Simulation Results

Figure 5.3 shows the throughput, delay, and loss rate observed during the simulations of Original System I and Pruned System I, using the the configurations described in Table 5.2 in page 63. Note that in these figures, *simulated clock cycle* is referred to as simulated clock cycle of *Original System I* and offered load is referred to as $\lambda_{ORIGINAL}$. For the sake of simplicity of comparison, we will use $\lambda_{ORIGINAL}$ as offered load for both Pruned System I and Original System I (λ_{PRUNE} can be easily found by dividing $\lambda_{ORIGINAL}$ by 16).

We expect the performance of Pruned System I to be better than that of Original System I because it has the advantage of using faster servers. This is, in fact, the behavior that is observed in the results.

5.2.4.1. *Throughput*

The rate of increase in average throughput for Pruned System I is proportional to the increase in offered load until $\lambda_{ORIGINAL} = 1$. At $\lambda_{ORIGINAL} = 1$, the average throughput of Pruned System I is approximately 94%, which is 20% higher than the throughput of Original System I.

5.2.4.2. *Delay*

The average delays are consistent with the throughput. Pruned System I has much lower delay than Original System I. For example, in terms of simulated clock of Original System I (i.e., 1 *simulated clock cycle* = 64 *electrical clock cycles*), when $\lambda_{ORIGINAL} = 0.25$, the average delay is 0.21 clock cycles for Pruned System I and 3.17 clock cycles for Original System I; that is, the average delay of Pruned System I is around fifteen times lower than that of Original System I. At full load, the average delay is roughly 4.1 clock cycles for Pruned System I, and about 9.2 clock cycles for Original System I, which is more than twice the average delay in Pruned System I.

5.2.4.3. *Loss Rate*

The average loss rates are also consistent with the throughput. For Pruned System I, overflow is detected only when $\lambda_{ORIGINAL} \geqslant 0.95$; at $\lambda_{ORIGINAL} \geqslant 0.95$, the system suffers from only a very low loss rate of less than 0.3%. On the other hand, in Original System I, overflow of packets is detected when $\lambda_{ORIGINAL} \geqslant 0.6$, and the loss rate is 27% at full load.

5.3. System III with Different Switching Strategies for Stage One

As described in the previous chapters, switching in stage one can be done by several selection strategies, namely, *straight through, random select,* and *repeat select* (refer to 3.3.5.1). The performance of System III with these three selection strategies are examined in this section. We chose to analyse System III, as opposed to System I and System II, because the results obtained for this system allow us to see more clearly the performance difference between these three strategies. We refer to these three versions of System III as *Straight*

Through System, *Random Select System*, and *Repeat Select System*, respectively. The arrival of packets at each input link follows a *Binomial process*, thus generating a *uniform*[8] traffic. The configurations for simulating System III with different selection strategies are shown in Table 5.3. The values of the *run-time* and *compile-time* variables for the simulations are given in Table A.3 in Appendix A. Note that only the values of the flags for the selection strategies (*ST_THRU*, *RANDOM* and *REPEAT_SELECT*) differ for the three simulations.

Configuration for simulating System III with Different Selection Strategies.			
	stage 1	stage 2	stage 3
# of switches	16	16	16
Size of switch	16 × 16	16 × 16	16 × 16
input-dilation of switch	1	1	1
output-dilation of switch	1	1	1
# of queues per switch	16	16	16
Size of queues	4	2	4
Total queueing capacity	1024	512	1024

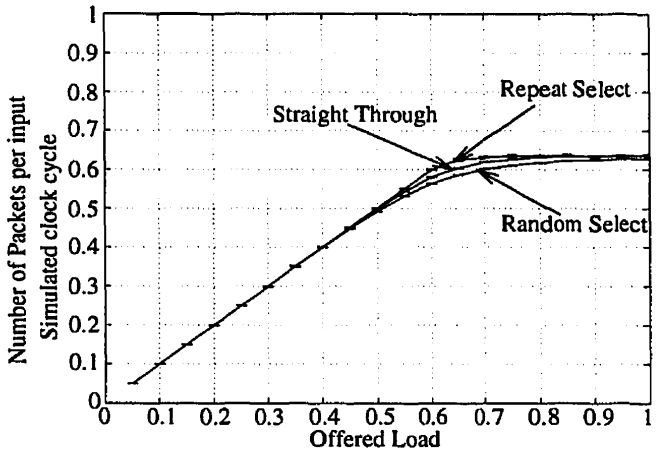
TABLE 5.3. Configuration for simulating System III for testing different selection strategies.

5.3.1. Simulation Results

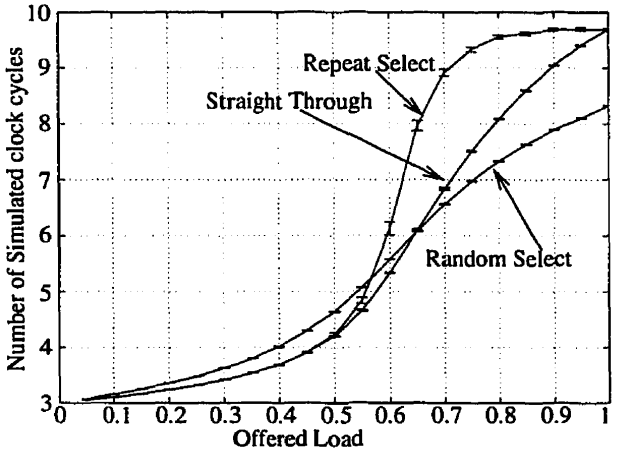
Figure 5.4 shows the throughput, delay and loss rate of System III simulated under the three selection strategies.

We expect Repeat Select System to have the best performance among the three systems because it always maximizes the use of buffers within a stage one switch. The performance of Straight Through System and Random Select System is expected to be not as good as Repeat Select System because each incoming packet will attempt to be transmitted to only one queue. If the queue is *full*, the packet will be *dropped* instead of attempting to be transmitted to other possibly empty queues, thus under-utilizing these queues. This behavior decreases the throughput and increases the loss rate. The results from the simulations satisfy the behavior just described.

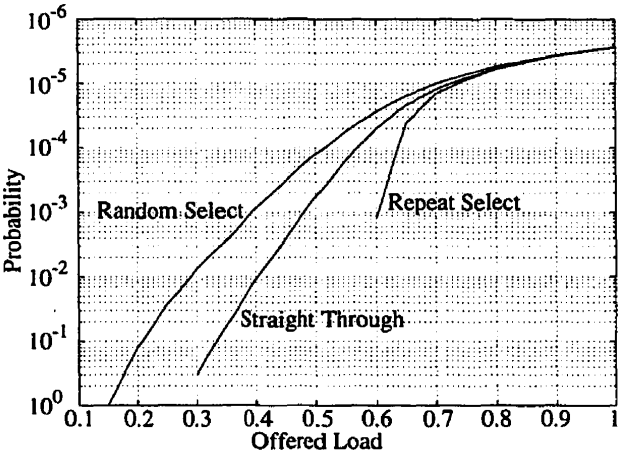
[8]Traffic is *uniform* when packets arrive at inputs of the network according to a Binomial process, each with parameter λ ($0 \leqslant \lambda \leqslant 1$).



(a) Throughput vs Offered load



(b) Delay vs. Offered load



(c) Loss rate vs. Offered load

FIGURE 5.4. Throughput, delay and loss rate of the different switching strategies for stage one.

5.3.1.1. *Throughput*

From Figure 5.4(a), it is observed that all of the three selection strategies have the same throughput when the load is light ($\lambda \leqslant 0.5$). This behavior is exhibited because the queues in stage one are not full at light loads, so there is no noticeable difference between the three selection strategies.

At higher loads ($\lambda \geqslant 0,5$), the average throughputs of Straight Through System and Random Select System start to degrade because some queues become full, and packets arriving at these queues are dropped. Also, it is observed that the rate of degradation of

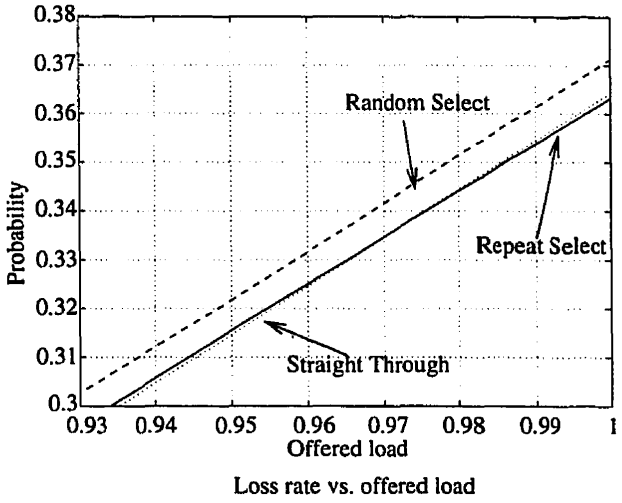


FIGURE 5.5. Loss rate of the different switching strategies for stage one.

Random Select System's throughput is faster than that of Straight Through System. This can be explained by the following. For Random Select System, some output queues in stage one may have packets arriving from more than one input. This increases the probability for a queue to overflow, leading to increasing packet loss. This situation becomes more serious as the load increases, resulting in faster degradation of the performance of Random Select System. Unlike Random Select System and Straight Through System, Repeat Select System maintains its rate of increase in throughput until $\lambda = 0.65$ because packets are only dropped when all buffers in the switch are occupied.

At high loads ($\lambda \geqslant 0.8$), average throughputs of Repeat Select System and Straight Through System are similar. This is because almost all queues in stage one are *full* in this case. Furthermore, as observed in Figure 5.4(a), at such loads, Random Select System never reaches the throughput of Repeat Select System and Straight Through System. This behavior, as we already explained, is a consequence of each queue receiving multiple packets in a single clock cycle, which causes overflow to increase. As such, the loss rate of Random Select System is approximately 1% higher than that of Repeat Select System and Straight Through System at full load, as shown in Figure 5.5 which is the enlargement of Figure 5.4(c) of offered load between 0.93 to 1. Also, the average throughput attained by Repeat Select System and Straight Through System is approximately 64%, and the throughput attained by Random Select System is roughly 63%, at full load.

5.3.1.2. *Delay*

The average delays are consistent with the throughput. At light loads ($\lambda \leqslant 0.5$), Random Select System has the highest average delay. This is because multiple packets may arrive at some queues causing these queues to hold up more packets than other queues. This imbalance in packet distribution among the queues increases the average delay. However, at high loads, ($\lambda \geqslant 0.65$), Random Select System has the lowest average delay because overflow of the *long* queues decreases the number of packets in the system, while the throughput is slowly increased (see Figure 5.4(a)). According to Little's Law [9], this translates into lower delays.

It is observed that, although multiple packets may also arrive at some queues for Repeat Select System, the delay is lower that that of Random Select System at light loads. This is because, when a packet arrives at a full queue in stage one, it will repeatedly select another output queue until it is forwarded or until it finds out that all queues are full, resulting in a more balanced distribution of packets in the queues. However, at high loads ($\lambda \geqslant 0.5$), since almost all buffers in the switches are at full capacity, the delay is much higher, compared to Random Select System and Straight Through System. At full load, the average delays for Repeat Select System and Straight Through System are similar because almost all queues are already full.

The maximum average delay for Repeat Select System and Straight Through System is approximately 9.6 clock cycles (1 clock cycle = 64 electrical cycles) and the maximum delay for Random Select System is 8.3 clock cycle.

5.3.1.3. *Loss Rate*

The average loss rates for the three systems are also consistent with the throughputs. Random Select System starts to experience detectable packet overflow at a very light load of $\lambda = 0.15$. As explained earlier, this behavior is caused by the *long* queues in stage one. Repeat Select System only experiences a detectable packet overflow when $\lambda \geqslant 0.6$. This behavior reflects the efficient use of buffers in its stage one. At full load, Repeat Select System and Straight Through System experience a similar amount of packet loss, whereas, Random Select System suffers a slightly more (approximately 1% higher) packet loss, as shown in Figure 5.5 on page 68. This additional loss is due to the unbalanced distribution of packets in the queues in stage one of the Random Select System.

[9] $Delay = \frac{Number\ of\ packets\ in\ system}{Throughput}$

5.4. Examples of the Backplane's Capacity

It would be interesting to use the simulated results for real world applications so that one can get an idea of how the systems perform in reality. Assuming each of the sixteen sources in a PCB has a bandwidth of 622 Megabits per second (Mbps), i.e.,the OC-12 telecommunication rate [10], each PCB will have a total bandwidth of 16 × 0.622 Gbps = 9.952 Gbps (where Gbps = Gigabit per second), and a backplane with sixteen PCBs will have a capacity of 16 × 9.952 Gbps = 159.23 Gbps. For example, in the first set of simulations, the maximum throughputs of System I, System II, and System III are 159.23 × 73.2% = 116.6 Gbps, 159.23 × 70.7% = 112.6 Gbps, and 159.23 × 63.5% = 101.2 Gbps, respectively.

If the OC-48 telecommunication rate [11] is used, each of the sixteen sources in a PCB will have a bandwidth of 2.488 Gbps, each PCB source will have a bandwidth of 16 × 2.488 Gbps = 39.8 Gbps, and the backplane will have a total capacity of 16 × 39.8 Gbps = 636 Gbps. For example, in the first set of simulations, the maximum throughputs of System I, System II, and System III are 636 × 73.2% = 465.8 Gbps, 636 × 70.7% = 449.8 Gbps, and 636 × 63.5% = 404.3 Gbps, respectively. Table 5.4 summarizes the maximum throughput of each system presented in this chapter, given that OC-12 and OC-48 telecommunication rates are used for busses in the backplane.

5.5. Summary

In this chapter, three sets of simulation experiments were conducted. The first set of simulations is to compare the performance of systems with different number of optical switches (1, 4, and 16) in the second stage; these systems are System I, System II, and System III, respectively. The second set of simulations is to compare two versions of System I, each with a different server speed; the one with the faster server speed is referred to as *Pruned System I* and the other one is referred to as *Original System I*. The third set is to compare three versions of System III, each using a different selection strategy, namely, *straight through*, *random select* and *repeat select*. We refer to these as *Straight Through System*, *Random Select System* and *Repeat Select System*, respectively. Finally, the throughputs obtained from the simulations are used for real world applications so that we can get an idea of how the systems perform in reality.

[10] OC-12 is a SONET optical carrier (OC) level with a bit rate of 622.08 Mbps [23].

[11] OC-48 is a SONET optical carrier (OC) level with a bit rate four times faster than that of OC-12's, i.e., 2488.32 Mbps [23].

Maximum throughputs (in Gigabits per second) of systems using the OC-12/OC-48 telecommunication rate.		
The Three Systems		
	OC-12	OC-48
System I	116.6 Gbps	465.8 Gbps
System II	112.6 Gbps	449.8 Gbps
System III	101.2 Gbps	404.3 Gbps
System I with Different Server Speeds		
	OC-12	OC-48
Original System I	116.6 Gbps	465.8 Gbps
Pruned System I	150.8 Gbps	602.3 Gbps
System III with Different Selection Strategies in Stage One		
	OC-12	OC-48
Straight Through System	101 Gbps	404 Gbps
Random Select System	100 Gbps	400 Gbps
Repeat Select System	101 Gbps	404 Gbps

TABLE 5.4. Maximum throughputs of systems using the OC-12/OC-48 telecommunication rate.

When comparing the performance of the three systems, it can been observed that, at high loads, the throughput of System I is around 2.5% higher than that of System II, and 9.7% higher than that of System III. The maximum throughput of System I is 73.2%. At light loads, the delay of Systems I and II are identical, which suggests that System II is as powerful as System I under such conditions. The delay of Systems I and II are also similar at full load.

When comparing the performance of the two versions of System I, Pruned System I was found to be better than Original System I. Pruned System I has a much higher average throughput and much lower average delay and average loss rate than Original System I. These results are due to the advantage of using faster servers.

When comparing the performance of System III with different selection strategies, we found that Repeat Select System has the highest average throughput and the lowest loss rate. This behavior is due to the efficient use of buffers within switches in stage one. However, this behavior also causes high average delays. Random Select System has the

worst performance among the three selection strategies because it creates an unbalanced distribution of arriving packets in queues of stage one.

CHAPTER 6

Conclusion

In this thesis, a software simulator has been built to provide a tool for the simulation study of the performance of the free-space optical backplane. We assume that the backplane is configured to support a 3-stage crossbar switching system, with electrical switches in the first and third stages, and optical switches in the second. In Chapter 2, the design of optical crossbar switches and the three main queueing strategies (input, output, and central queueing) that can be implemented in a switch were reviewed. We observed that output queueing offers performance advantages over input queueing and implementation advantages over central queueing. Therefore, output queueing was employed in the switching networks in this thesis.

In Chapter 3, the specification of the three 3-stage crossbar switching systems were discussed. These systems differ by the number and complexity of switches (1, 4, and 16) in the second stage, thereby making the interconnections of switches in each system different. Other than this, the characteristics (e.g., queueing discipline, routing strategy, and switches in stage one and three) of the three systems are identical. An in-depth description of the switch architecture in each stage, routing strategy, and queueing discipline of the system with four optical switches in the second stage was given.

In Chapter 4, the software simulator that was used to perform discrete event simulations of the three systems was described. In addition, the simulator model development approach, statistical information collection, and set-up of the simulation parameters were described. The pseudo-code of the software simulator, validation criteria of the software, and possible extensions of the software were also given.

Finally, in Chapter 5, three sets of simulation experiments were conducted, and their average throughputs, delays, and loss rates were presented and examined.

The aim of the first set of simulations is to compare the performance of the three systems (discussed in Chapter 3) that have equal queueing capacities in each switch in stage one and stage three, and equal aggregate queueing capacity in stage two. We illustrated that large optical switches in the second stage yield the highest performance, at the expense of relatively complex logic. We found that, at full load, the throughput of the system that uses one big optical switch in the second stage is only 2.5% higher than the throughput of the system that uses four medium size optical switches, and 9.7% higher than the throughput of the system that uses sixteen small optical switches. We also found that, at light loads, the performances of the system with one big optical switch and the system with four medium optical switches in the second stage are identical. This suggests that the system with medium optical switches is as powerful as the one with a big optical switch, at light loads. Moreover, we also found that these two systems have similar average delays at full load.

The aim of the second set of simulations is to compare the performance of a system with two different server speeds. We selected the system that has one big optical switch in the second stage for this set of experiments. We considered a version of the system that has sixteen times less servers but with each server having a service rate that is sixteen times faster than those in the original system. We found that the performance of the system with faster servers is far better than that of the original system. Its maximum throughput is approximately 20% higher than that of the original system, and its delay is much lower than that of the original system.

Finally, the last set of simulation experiments compares the three different routing strategies in first stage, namely, *repeat select*, *random select*, and *straight through*. We selected the system that has sixteen small optical switches in the second stage for this set of experiments. We found that the system that employs *repeat select* has the highest average throughput and the lowest loss rate. This behavior is due to the efficient use of buffers within switches in stage one. However, this behavior also causes high average delays. The system that uses *random select* has the worst performance among the three selection strategies because it creates an unbalanced distribution of arriving packets in queues of stage one.

6.1. Recommendations

Based on the analysis of the experimental results given in Chapter 5, the following suggestions are made.

In this thesis, we did not consider the details of the VLSI design of the smart pixel arrays and optical switches. It is recommended that one could consider the trade-off between the VLSI complexity of the optical switches which are to be embedded in the backplane and the throughput of the network achievable by implementing these switches.

From basic queueing theory, it is known that a system utilizing a small number of fast servers has higher throughput, lower delay, and lower loss rate than a system utilizing a large number of slower servers. In this thesis, the performance improvement has been quantified through the use of network simulation. The use of faster servers means a need for fast multiplexers and de-multiplexers. Since the details of VLSI design of multiplexers and de-multiplexers have not been considered in this thesis, it is recommended that one should consider the cost of faster multiplexers and de-multiplexers in the decision to replace a large number of slower servers with a small number of faster servers in order to achieve better throughput.

Finally, it is recommended that when deciding which selection strategies (*repeat select*, *random select*, and *straight through*) should be used by packets to select output links in electrical switches in stage one, one should consider the difference in the throughput, delay and loss rate that is achievable by each strategy and the processing time that is required. For example, using *repeat select* yields a high throughput and a low loss rate, but at the expense of high delay and processing time. Moreover, building an electrical switch with the *repeat select* strategy will require higher complexity (and probably more expensive) hardware.

6.2. Future Work

It may be worthwhile to explore the performance of the three systems under different routing strategies in stage one (*straight through, random select, and repeat select*) with non-uniform traffic. Systems with external input queues and/or unbuffered switches in stage two may also be a good exploration. A more realistic traffic model that uses multi-packet messages may also be worthwhile to consider.

REFERENCES

[1] Rad'ed Y. Awdeh and H.T. Mouftah, "Survey of atm switch architectures", *Computer Networks and ISDN Systems*, vol. 27, pp. 1567–1616, 1995.

[2] Dimitri Bertsekas and Robert Gallager, *Data Networks*, Prentice Hall, 2 edition, 1992.

[3] *Canadian Institute for Telecommunications Research Annual Report*, McGill University, Suite 753, Montreal, Queubec, Canada H3A 2A7. Phone: (514) 398-8140, Fax:(514) 398-3127, 1996, WWW: http::/www.citr.ee.mcgill.ca.

[4] Martin de Prycker, *Asynchronous Transfer Mode: solution for broadband ISDN*, Ellis Horwood, 1991.

[5] Joan Garcia-Haro and Andrzej Jajszczyk, "Atm shared-memory switching architectures", *IEEE Network*, July/August 1994.

[6] Ilias Iliadis and Wolfgang W. Denzel, "Analysis of packet switches with input and output queueing", *IEEE Trans. Commun.*, May 1993.

[7] Naim A. Kheir, *Systems Modelling and Computer Simulation*, Marcel Dekker, Inc., 1988.

[8] Tony T. Lee and Soung C. Liew, "Broadband packet switches based on dilated interconnection networks", *IEEE Transactions on communications*, vol. 42, no. 2/3/4, pp. 732–744, February/March/April 1994.

[9] Soung C. Liew, "Multicast routing in 3-stage clos atm networks", *IEEE Transactions on Communications*, vol. 42, no. 2–4, pp. 1380–1390, February–April 1994.

[10] M. Naraghi-pour, M. Hegde and B. Reddy, "Multiple shared memory switch", in *Proceedings of the Annual Southeastern Symposium on system theory 1996*, Piscataway, NJ, USA., 1996.

[11] M.H. MacDougall, *Simulating Computer Systems*, The MIT Press, 1987.

[12] Fred Maryanski, *Digital Computer Simulation*, Hayden Book Company, inc., 1980.

[13] M.J. Karol, M.G. Hluckyj, and S.P. Morgan, "Input versus output queueing on a space division packet switch", *IEEE Trans. Commun.*, vol. com-35, December 1987.

[14] I. Mitrani, *Simulation techniques for discrete event systems*, Cambridge University Press, 1982.

[15] Roger S. Pressman, *Software Engineering. A practitioner's Approach*, McGraw-Hill International Editions, 1992.

[16] Enrico Del Re and Romano Fantacci, "Performance evaluation of input and output queueing techniques in atm switching systems", *IEEE Trans. Commun.*, vol. 41, no. 10, October 1993.

[17] Reza Rooholamini and Vladimir Cherkassky, "Finding the right atm switch for the market", *IEEE Computer*, April 1994.

[18] Ted Szymanski, "Notes on the optical backplane", 1996, Internal memo.

[19] Ted Szymanski and C. Fang, "Randomized routing of virtual connections in an essentially non-blocking logn-depth network", *IEEE Transactions on Communications*, September 1995.

[20] Ted Szymanski and H. Scott Hinton, "Design of a terabit free-space photonic backplane for parallel computing", in *2nd International conference on Massively Parallel Processing using Optical Interconnects '95*, San Antonio, Texas, October 1995.

[21] Ted Szymanski and H. Scott Hinton, "Reconfigurable intelligent optical backplane for paralletl computing and communications", *Applied Optics*, vol. 35, no. 8, March 1995.

[22] Ted Szymanski and Scott Hinton, "Graph embeddings in a photonic hyperplane", in *International Conference on Applications of Photonic Technology (ICAPT-94)*, Toronto, Canada, June 21-23 1994.

[23] Ronald J. Vetter, "Atm concepts, architectures, and protocols", *Communications of the ACM*, vol. 38, no. 2, pp. 30–109, February 1995.

[24] William Mendenhall,Dennis D. Wackerly and Richard L. Scheaffer, *Mathematical Statistics with Applications*, PWS-KENT Publishing Company, 4 edition, 1990.

[25] Masayasu Yamaguchi and Kenichi Yukimatsu, "Recent free-space photonic switches.", *IEICE Transacations on Communications*, no. 2, pp. 128–138, February 1994.

[26] Yu-Shuan YehMichael G. Hluckyj and Anthony S. Acampora, "The knockout switch: A simple, modular, architecture for high-preformance packet switching", *IEEE Journal on Selected Areas in Communications*, vol. SAC-5, no. 8, October 1987.

[27] Yuji Oie, Masayuki Murata, Koji Kubota and Hideo Miyahara, "Performance analysis of nonblocking packet switch with input and output buffers", *IEEE Transaction on Communications*, vol. 40, no. 8, August 1992.

[28] Yuji Oie, Tatsuya Suda, Masayuki Murata and Hideo Miyahara, "Survey of the performance of nonblocking switches with fifo input buffers", *Supercomm ICC '90 Conference Record - International Conference on Communication. Publ by IEEE*, vol. 2, 1990.

APPENDIX A

System Configurations for the Experimental Results

Variables for simulating System I, II,and III			
System	I	II	III
Run-time variables			
System configuration	I	II	III
External input queue capacity	0	0	0
Debug mode	OFF	OFF	OFF
Compile-time variables			
ST_THRU	ON	ON	ON
RANDOM	OFF	OFF	OFF
REPEAT_SELECT	OFF	OFF	OFF
PRUNE	0	0	0
BUFFERED_2NDSTG	ON	ON	ON
SELECT_QSIZE	ON	ON	ON
STAGE1_OQUEUE_SIZE	4	4	4
STAGE2_OQUEUE_SIZE	2	2	2
STAGE3_OQUEUE_SIZE	4	4	4

TABLE A.1. Variables for simulating System I, II,and III (Section 5.1).

Variables for simulating PRUNED System I	
run-time variables	
System configuration	I
External input queue capacity	0
Debug mode	OFF
compile-time variables	
ST_THRU	OFF
RANDOM	ON
REPEAT_SELECT	OFF
PRUNE	1
BUFFERED_2NDSTG	ON
SELECT_QSIZE	ON
STAGE1_OQUEUE_SIZE	64
STAGE2_OQUEUE_SIZE	2
STAGE3_OQUEUE_SIZE	4

TABLE A.2. Variables for simulating Pruned System I.

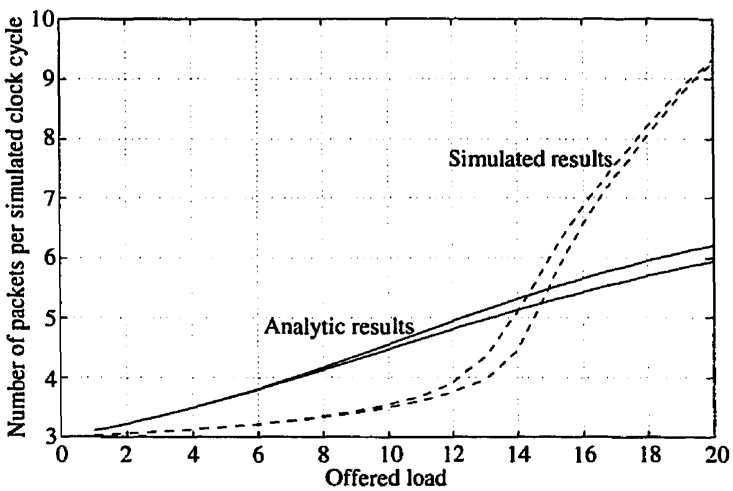


FIGURE A.1. Comparsion of results from the analytic model and the simulator for System I at light load.

Variables for simulating Different Selection Strategies for System III			
run-time variables			
	Straight Through	Random Select	Repeat Select
System configuration	III	III	III
External input queue capacity	0	0	0
Debug mode	OFF	OFF	OFF
compile-time variables			
	Straight Through	Random Select	Repeat Select
ST_THRU	ON	OFF	OFF
RANDOM	OFF	ON	OFF
REPEAT_SELECT	OFF	OFF	ON
PRUNE	0	0	0
BUFFERED_2NDSTG	ON	ON	ON
SELECT_QSIZE	ON	ON	ON
STAGE1_OQUEUE_SIZE	4	4	4
STAGE2_OQUEUE_SIZE	2	2	2
STAGE3_OQUEUE_SIZE	4	4	4

TABLE A.3. Variables for simulating different selection strategies.

APPENDIX B

Sample Output from the Simulator

B.1. Abbreviations used in the Simulator Standard Output

stats: statistics

N: total number of input ports and total number of output ports.u

u: utilization (offered load).

rxs: total number of packets received.

txs: total number of packets transmitted.

p(rx): probability that an input receives a packet per clock cycle.

p(tx): probability that an output transmit a packet per clock cycle.

nbw: net bandwidth.

age: average delay of a packet.

ext_hoq: average delay of a packet at the head of line of an external input queue.

extqs: average number of packet in an external input queue.

p(qof): probability of packet overflow.

Avg.: average.

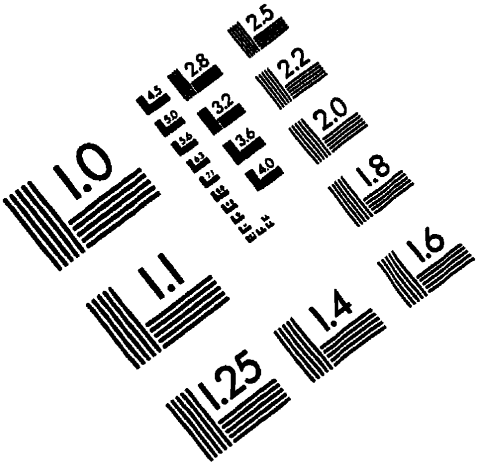
ci: confident intervals.

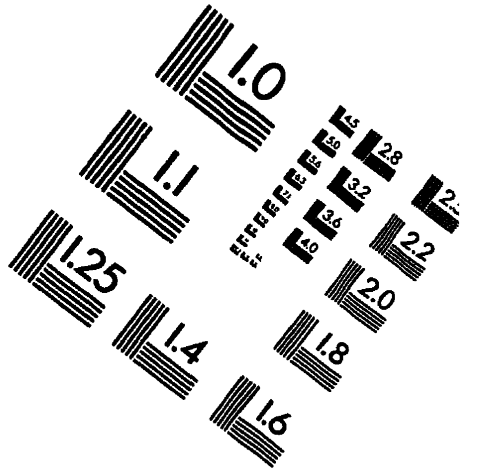
B.2. Sample Output from the Simulator

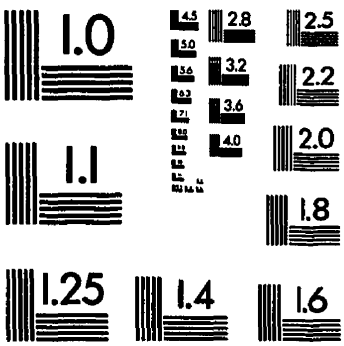
```
*************************************************************************
*************** 3-stage crossbar switching network simulator  *************
*************************************************************************


Enter system (1, 2 or 3): 1
Enter size of EXTERNAL QUEUE (0->no queue): 3
Enter debug (1=ON, 0=OFF): 0
******************** Switches info ***************************************

PRUNE OPTION:         OFF
STRAIGHT THROUGH:     ON
RANDOM SELECT:      OFF
REPEAT SELECT:        OFF
BUFFERED 2nd STAGE:   OFF

Number of switches in stage 1: 16
Number of switches in stage 2: 16
Number of switches in stage 3: 16

Degree of switches in stage 1: 16
Degree of switches in stage 2: 256
Degree of switches in stage 3: 16


STAGE 1 QSIZE: 4
STAGE 2 QSIZE: 2
STAGE 3 QSIZE: 4

*************************************************************************

==========================================
Output Q size in Node = 1
Input Q size = 3
==========================================

N 256
stats for N = 256, u = 0.050
trial# rxs     txs     p(rx) p(tx)  nbw     age    ext_hoq extqs p(qof)
     0 400002 400028 0.0500 0.0500 0.0500 3.026 1.0000  0.050 0.0000
     1 400003 399999 0.0500 0.0500 0.0500 3.026 1.0000  0.050 0.0000
     2 400010 400011 0.0500 0.0500 0.0500 3.027 1.0000  0.050 0.0000
     3 400007 400013 0.0500 0.0500 0.0500 3.026 1.0000  0.050 0.0000
     4 400005 400002 0.0499 0.0499 0.0499 3.026 1.0000  0.050 0.0000
     5 400008 400007 0.0500 0.0500 0.0500 3.027 1.0000  0.050 0.0000
     6 400005 400001 0.0499 0.0499 0.0499 3.026 1.0000  0.050 0.0000
     7 400005 400007 0.0500 0.0500 0.0500 3.026 1.0000  0.050 0.0000
     8 400004 400005 0.0499 0.0499 0.0499 3.026 1.0000  0.050 0.0000

Avg. nbw = 0.04998 +- 0.00004 (.95 ci)
Avg. age = 3.02620 +- 0.00028 (.95 ci)
Avg. age per stage = 1.00873 +- 0.00009 (.95 ci)
Avg. p(qof) = 0.00000 +- 0.00000 (.95ci)
Avg. ext_hoq  = 1.00000 +- 0.00000 (.95ci)

stats for N = 256, u = 0.100
trial# rxs     txs     p(rx) p(tx)  nbw     age    ext_hoq extqs p(qof)
     0 400008 400065 0.1002 0.1003 0.1002 3.055 1.0000  0.100 0.0000
     1 400015 400009 0.1000 0.1000 0.1000 3.056 1.0000  0.100 0.0000
     2 400003 400014 0.1002 0.1002 0.1002 3.055 1.0000  0.100 0.0000
     3 400017 399999 0.1001 0.1001 0.1001 3.056 1.0000  0.100 0.0000
     4 400014 400022 0.0999 0.0999 0.0999 3.055 1.0000  0.100 0.0000
     5 400022 400018 0.1000 0.1000 0.1000 3.055 1.0000  0.100 0.0000
     6 400013 400017 0.1001 0.1001 0.1001 3.055 1.0000  0.100 0.0000
     7 400009 400014 0.1000 0.1000 0.1000 3.055 1.0000  0.100 0.0000
     8 400004 400000 0.1001 0.1001 0.1001 3.055 1.0000  0.100 0.0000
```

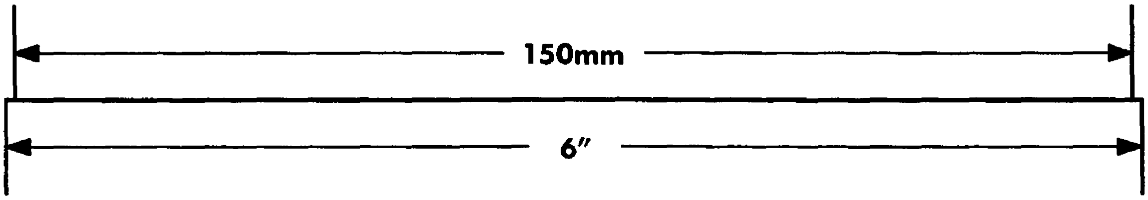
```
Avg. nbw = 0.10004 +- 0.00007 (.95 ci)
Avg. age = 3.05525 +- 0.00026 (.95 ci)
Avg. age per stage = 1.01842 +- 0.00009 (.95 ci)
Avg. p(qof) = 0.00000 +- 0.00000 (.95ci)
Avg. ext_hoq  = 1.00000 +- 0.00000 (.95ci)
                          .
                          .
                          .

 (u = 0.1500 .. 0.9500 are skipped in this print out)
                          .
                          .
                          .

stats for N = 256, u = 1.000
trial# rxs     txs    p(rx) p(tx)  nbw    age   ext_hoq extqs p(qof)
     0 400171 401586 0.7250 0.7276 0.7263 11.655 1.3733  2.978 0.2710
     1 400130 400162 0.7297 0.7298 0.7297 11.696 1.3704  3.000 0.2702
     2 400112 400092 0.7266 0.7266 0.7266 11.732 1.3762  3.000 0.2734
     3 400007 400009 0.7291 0.7291 0.7291 11.710 1.3717  3.000 0.2709
     4 400128 400122 0.7294 0.7293 0.7293 11.699 1.3710  3.000 0.2707
     5 400119 400104 0.7259 0.7259 0.7259 11.731 1.3772  3.000 0.2741
     6 400102 400093 0.7286 0.7286 0.7286 11.717 1.3728  3.000 0.2714
     7 400077 400124 0.7286 0.7287 0.7286 11.714 1.3725  3.000 0.2713
     8 400014 399946 0.7268 0.7266 0.7267 11.726 1.3759  3.000 0.2734

Avg. nbw = 0.72808 +- 0.00100 (.95 ci)
Avg. age = 11.71557 +- 0.00950 (.95 ci)
Avg. age per stage = 3.90519 +- 0.00317 (.95 ci)
Avg. p(qof) = 0.27192 +- 0.00102 (.95ci)
Avg. ext_hoq  = 1.37346 +- 0.00181 (.95ci)
```

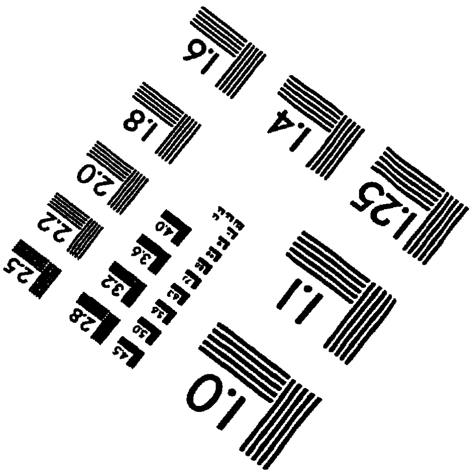
TEST TARGET (QA-3)

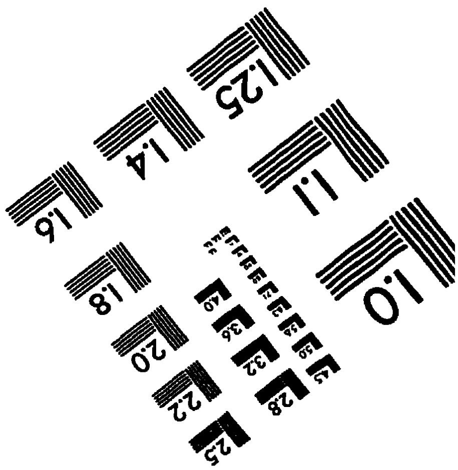












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989