Switching Fabrics for Broadband Networks

A thesis submitted to
The Hong Kong University of Science and Technology
in partial fulfillment of the requirements for
the Degree of Master of Philosophy in
Electrical and Electronic Engineering

By
Lei ZAN
Department of Electronics
B.S., July 1998
Peking University, Beijing, CHINA

Dec. 2000
Switching Fabrics for Broadband Networks

By

Lei ZAN

Approved by:

Prof. Chin-Tau LEA
Thesis Supervisor

Dr. Jia-Nan QU
Thesis Examination Committee Member (Chairman)

Dr. Ho-Chi HUANG
Thesis Examination Committee Member

Prof. Philip Ching Ho Chan
Head of the Department

Department of Electrical and Electronic Engineering
The Hong Kong University of Science and Technology

Dec. 2000
Acknowledgements

First and foremost I would like to express my special gratitude to my advisor, Professor Chin-Tau Lea, for his insightful guidance, incentive encouragement and support throughout my study. Working with him is a valuable experience for me.

I give my sincere appreciation to Dr. Jia-Nan QU and Dr. Ho-Chi HUANG for reviewing this thesis and providing valuable comments and pertinent suggestions for improvement.

I am also grateful to all the members of the Computer Networks and System Integration Lab, especially Lai Wangyang, Hei Xiaojun and Shen Hongxia, Patrick Chan, Steven Wong, for their friendly support and assistance.

My deep love and thanks go to my family for their constant encouragement and support. I would like to especially mention of Mr. SHE Peiman, James, for his help from the beginning to the end. His enthusiasm and indefatigable personality stimulates me much.
Switching fabrics for broadband networks

By

Lei Zan

For the degree of
Master of Philosophy in Electrical and Electronic Engineering
At the Hong Kong University of Science and Technology
Dec. 2000

Abstract

Advances in switching technologies are crucial to achieve the goal of current broadband ISDN. In my thesis, much work has been done on both electronic and photonic switching.

In the photonic switching category, $\log_2(N,m,p)$ networks are examined and the methods to construct nonblocking multicast $\log_2(N,m,p)$ networks are discussed. The results presented allow the determination of the best design tradeoffs among some important parameters such as attenuation, crosstalk and the total amount of hardware, when used for directional-coupler-based photonic switching systems.

In the electronic switching category, input organization and scheduling algorithm are our main focus to improve the performance of input-queued switches. A scheme with queue-splitting and random selection is deployed. Performance has been evaluated not only for single-stage and multi-stage switches with unicast traffic, but also for multicast switches. Simulation results show that great improvement in performance can be achieved if our approach is used. The most significant features of this scheme are that it can eliminate the degradation of traffic correlation and is
applicable to both unicast and multicast traffic. Therefore, we conclude that this scheme is a good candidate for input-queued switch architectures.
# TABLE OF CONTENTS

## Chapter 1. Introduction

1.1 Introduction to broadband switching........................................1
1.2 Photonic switching.....................................................................4
1.3 ATM switching..........................................................................6
  1.3.1 ATM switching architecture..................................................7
  1.3.2 Queueing strategies..............................................................9
1.4 Thesis organizations.................................................................11

## Chapter 2. Multicast Log$_2$N networks

2.1 Introduction..............................................................................12
2.2 Our proposed method for multicast Log$_2$(N,m,p) networks..........15
  2.2.1 Splitting a multicast connection into sub-connections.............15
  2.2.2 Definition and characteristic of sub-connections..................18
  2.2.3 Worst-case traffic pattern....................................................19
2.3 Results and Concluding remarks.............................................22

## Chapter 3. Scheduling algorithms for crossbars

3.1 Introduction.............................................................................24
3.2 Simulation model and performance methodologies..................26
  3.2.1 Simulation model...............................................................26
  3.2.2 Performance methodologies...............................................28
3.3 iSLIP algorithm with queue-splitting....................................29
Chapter 4. Multicast packet switching

4.1 Introduction.................................................................................57
4.2 Multicast packet switch with queue-splitting scheme..................59
4.3 Performance evaluation of multicast switch with queue-splitting scheme.................................................................61
  4.3.1 Performance of Policy 1.........................................................62
  4.3.2 Performance of Policy 2.........................................................66
  4.3.3 Performance of Policy 3.........................................................69
4.4 Conclusion....................................................................................72

Chapter 5. Conclusion and future work

References
List of Figures

1.1 A crossbar switch fabric.................................................................8
1.2 A 3-stage space-division switch fabric.................................................9
2.1 Topologically equivalent self-routing networks: Banyan, baseline and shuffle networks.................................................................13
2.2 (a) The bipartite graph representation..............................................16
2.2 (b) The crossbar representation of banyan network and bipartite graph representation of baseline network with size 8..............................................16
2.3 The input and output window for $RS=3$, $IWS=8$, $OWS=4$..................17
2.4 The redundancy graph of the $Log_2(16,1,1)$ network............................18
2.5 The left tree and right tree of $Log_2(16,1,1)$ network for $RS=3$..........19
2.6 The left tree and right tree of $Log_2(16,1,1)$ network for $RS=2$............19
2.7 (a) Input 1 and output 1 combined will block only half plane..............20
2.7 (b) Now input 1 and output 1 combined block one plane......................20
2.8 (a) The tagged connection is $<0,\{0\}>$.................................................22
2.8 (b) The tagged connection is $<0,\{0,4\}>$.................................................22
2.8 (c) The worst-case tagged connection.................................................22
3.1 Illustration of HOL blocking..............................................................25
3.2 An input-queued switch with "Virtual Output Queueing"...........................26
3.3 Diagram of simulation model..............................................................27
3.4 Example of the three steps of RRM matching........................................30
3.5 Average delay versus offered load for 16x16 network with iSLIP, queue-split size=4, uncorrelated traffic.........................................................35
3.6 Average delay versus offered load for 16×16 network with iSLIP, queue-split size=16, uncorrelated traffic .................................................................36

3.7 Cell loss versus buffer size for 16×16 network with iSLIP, queue-split size=4, uncorrelated traffic .................................................................37

3.8 Cell loss versus buffer size for 16×16 network with iSLIP, queue-split size=4, correlated traffic .................................................................38

3.9 Cell loss versus buffer size for 16×16 network with iSLIP, queue-split size=16, uncorrelated traffic .................................................................38

3.10 Cell loss versus buffer size for 16×16 network with iSLIP, queue-split size=16, correlated traffic .................................................................39

3.11 Cell loss versus buffer size for 16×16 network with iSLIP, speedup=2 ........39

3.12 Average delay versus offered load for 16×16 network single-stage network with queue-splitting size=4 .......................................................45

3.13 Cell loss versus buffer size for 16×16 network single-stage network with queue-splitting size=4, uncorrelated traffic ..................................46

3.14 The topology of 16×16 two-stage network switch with switching element 4 ....48

3.15 Average delay versus offered load for 64×64 two-stage network with queue-splitting size=4 .................................................................49

3.16 Average delay versus offered load for 64×64 two-stage network with queue-splitting size=8 .................................................................50

3.17 Cell loss versus buffer size for 64×64 two-stage network with queue-splitting size=4, uncorrelated traffic .......................................................50

3.18 Topology of three-stage 64×64 switches with SE 4 ..................................51
3.19 Average delay versus offered load for 64×64 three-stage network with queue-splitting size=4........................................................................................................52
3.20 Cell loss versus buffer size for 64×64 three-stage network with queue-splitting size=4, uncorrelated traffic.................................................................53
3.21 Multi-stage network with d×d switching element..............................................53
4.1 4×4 switch architecture with queue-splitting size 2..........................................59
4.2 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=4, uncorrelated traffic, policy 1.........................64
4.3 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=16, uncorrelated traffic, policy 1..............................64
4.4 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=4, correlated traffic with mean pk-length=5, policy 1..................................................................................................................65
4.5 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=16, correlated traffic with mean pk-length=5, policy 1..................................................................................................................65
4.6 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=4, uncorrelated traffic, policy 2.........................67
4.7 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=16, uncorrelated traffic, policy 2.........................67
4.8 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=4, correlated traffic with mean pl-length=5, policy 2..................................................................................................................68
4.9 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=16, correlated traffic with mean pk-length=5, policy 2…………………………………………………………………………………………………………………………...68

4.10 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=4, uncorrelated traffic, policy 3………………………………………70

4.7 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=16, uncorrelated traffic, policy 3…………………………………70

4.8 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=4, correlated traffic with mean pl-length=5, policy 2………71

4.9 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=16, correlated traffic with mean pk-length=5, policy 2………71
List of Tables

2.1 The values of $p$ required for the $(N,m,p)$ networks to be strictly nonblocking for different value of $N$ and $m$.................................................................14
2.2 The values of $p$ required for multicast multi-$\log_2 N$ networks be to strictly nonblocking.................................................................14
2.3 The value of $p$ required for multicast $(N,m,p)$ networks to be strictly nonblocking for different value of $N$ and $m$.................................................................23
3.1 Maximum throughput of iSLIP for uncorrelated traffic.................................34
3.2 Maximum throughput of iSLIP for correlated traffic........................................34
3.3 Maximum throughput of window scheme.........................................................44
3.4 Maximum throughput for single-stage switch with queue-splitting scheme.......44
3.5 Maximum throughput for single-stage switch with queue-splitting, correlated traffic with mean pk-length=10.................................................................45
3.6 Maximum throughput for two-stage network with random scheduling, uncorrelated traffic.................................................................48
3.7 Maximum throughput for two-stage network with random scheduling, correlated traffic with mean pk-length=5.................................................................49
3.8 Maximum throughput for three-stage network with random scheduling, uncorrelated traffic.................................................................52
3.9 Maximum throughput for three-stage network with random scheduling, correlated traffic with mean pk-length=5.................................................................52
3.10 Maximum throughput for input-queued single-stage switch.............................55
3.11 Analytical results and simulation results for maximum throughput for multi-stage input-queued switch.................................................................56
4.1 Maximum throughput for 32×32 network with multicast cell prob.=1.0, policy
1.............................................................................................................63

4.2 Maximum throughput for 32×32 network with multicast cell prob.=1.0, policy
2.............................................................................................................66

4.3 Maximum throughput for 32×32 network with multicast cell prob.=1.0, policy
3.............................................................................................................69
Chapter 1 Introduction

This chapter provides the necessary background knowledge for understanding the materials in the rest of this thesis. Broadband switching, optical switching architecture and general functions of ATM switch will be discussed in the following sections. At the end of this chapter, the thesis organization will be given.

1.1 Introduction to broadband switching

In the late 1970s all communication networks could be divided into two categories: telephone networks and computer networks. They were designed for different purposes and were incompatible with each other. Later on, researchers found that it was beneficial to combine the existing networks with one universal network for all communication service. The concept of integrated services digital network (ISDN) was then proposed and CCITT adopted it in 1984. Shortly after the launch of ISDN, it was realized that ISDN was not suitable and efficient in handling high bandwidth applications. Later, an extended version called broadband ISDN (B-ISDN) was then under study. B-ISDN has the ability to accommodate all existing and future communication applications with a wide range of bandwidth and latency requirements. The development of B-ISDN has made advances made in switching technologies necessary in order to achieve the goal of broadband integrated communication networks.

Switching techniques in current telecommunication networks can be classified into two types: circuit-switched and packet-switched. Because these switching techniques have their strengths and weaknesses, they are useful for different kinds of frameworks. They will be discussed in the following paragraphs. In particular, ATM
is introduced since it is widely accepted as a proper candidate to provide high-speed packet switching.

Circuit switching is mainly used in telephone networks. A particular path is reserved for the whole duration of a call. So signals transmitted through this switching system do not experience any delay once the connection is set up. However, this allocation strategy reveals two disadvantages of circuit switching, namely the unfeasibility of tailoring the bandwidth allocated to the instantaneous requirements of the call and poor utilization of bandwidth. In addition, circuit switched systems are incapable to serve a larger bandwidth. So we need another switching technique.

The idea behind a packet-switching network was to create a network, the sole function of which is to transport digital data traffic. At the source, the data is divided into group of bits called packets. There are two parts of information in a packet. The first part, the header field, contains information such as a packet’s source, destination and its priority. The second part is the content of the information to be transported over the network. A great advantage of packet switching is its inherent characteristic if statistical multiplexing that lowers the cost of data transmission. Moreover, it allows the bursty traffic to be combined into aggregate flows that can be accommodated economically by connections.

ATM was proposed to create a broadband (high-speed) packet switching network capable of transporting a wide variety of services in an integrated manner. These services include voice, data and video communication. In an ATM network, all packets (also called cells) have a fixed length. All cells are 53 bytes long with a 48-byte payload and a 5-byte header. Since ATM must be capable of supporting a wide range of applications, it is likely that ATM networks will interconnect with the existing data networks [18][19]. In particular, the TCP/IP internetworking protocol
has been successful in connecting a large variety of computers with a great diversity of sizes, vendors and operating systems. Its widespread success has motivated the transmission of TCP/IP over ATM networks.

ATM presents some challenges in supporting data traffic. In particular, ATM employs a connection-oriented service with small fixed-size cells, whereas data networks employ a connectionless service with large variable-size packets (or datagrams). Before transmission in ATM networks, a data message is first segmented into packets, each of which is further disassembled into cells. The transport layer located over the ATM layer handles the transmission based on packets. The function of the transport layer is to manage end-to-end communication and to perform error control and traffic control. If a cell is lost within the network, the destination cannot resemble the packet to which the cell is originally belonged. The source needs to retransmit the packet. The ATM adaptation layer (AAL), which is the upper sub-layer of the ATM, transforms a packet received from the transport layer into cells at the source node and vice versa at the destination. For example, AAL5, one of the AAL protocols, directly divide a packet into 48-byte payload of the ATM layer. In addition, an ATM-layer-user-to-ATM-layer-user (AUU) parameter is provided in the payload-type indication (PTI) field of the ATM cell header, which indicates whether or not the cell is the last cell of a packet. If the AUU of a cell is set to be one, this represents the fact that the cell is the last one of a packet. As a result, in case of AAL5, the switch can recognize the last cell of a packet simply by monitoring the header of cells, without checking the actual payload.

Extensive work has been done on electronic switching. As the demand for high bandwidth continues to increase, photonic switching has attracted much research attention due to its many important characteristics needed in future networks.
supporting different forms of data, for example, it offers high-speed data rate/format transparency and configurability. Photonic switching will be covered in the following section.

1.2 Photonic switching

As the rapid growth of Internet and the increasing number of multimedia application, the capacity demand has increased in recent years. This demand for high bandwidth will continue to exert pressure on conventional electrical architectures and create opportunities for the introduction of optical architecture. The concept of wavelength-division-multiplexing (WDM) has provided researchers with an opportunity to multiply the network capacity. Current optical switching technologies allow the rapid delivery of the enormous bandwidth of WDM networks. WDM networking has been launched by the concept of wavelength routing. The principle is that high-speed data flows, which consist of many time-division multiplexed channels, are associated with specific optical wavelengths. Thus, they are routed through the optical network by means of their wavelengths, without necessarily being opto-electronically converted, demultiplexed, and electronically routed. This concept allows the realization of all-optical routers, which can handle many WDM channels simultaneously. The lower hierarchies are naturally processed by an electronic cross-connect that possibly interoperates with the optical cross-connect (OXC). Thus, wavelength routing consents the realization of optical add-drop multiplexers (OADM's) and OXC's working in a semi-permanent way. The capacity to aggregate optical bandwidth provides the means to cope with diverse Internet traffic.

The main feature of WDM optical network lies in the possibility of performing these operations directly in the optical domain without requiring costly high-speed
electronic equipment, and in its transparency, that is, the possibility of making those functions independent, to some extent, of the signal format. Actually, it is possible to define several degrees of transparency. In fact, absolute transparency is the property of a network in which any signal travels along the network independent of its transmission format, speed (bit rate in case of pulse code modulated, PCM, signals) and so on. However, due to physical limitations of fiber propagation and the physical nature of optical devices traversed by the signal, absolute transparency can never be reached. Thus, it is more useful to specify a certain level of transparency. The simplest degree of transparency is in digital signals (independence of bit rate, format, and protocol). Furthermore, it is possible to define transparency to intensity-modulated signals (both analogue and digital). Full transparency would require that a network be transparent to any optical signal, regardless of its amplitude, phase, or frequency modulation.

In practical networks, transparency will allow the handling of different types of data flows simultaneously. In fact, wavelengths can carry either synchronous digital hierarchy/optical network SDH/SONET) streams, ATM streams, or other possible transport formats.

The main issue when designing optical networks for Internet application is the right mode of transport from IP packets. Actually, several transport options have been proposed in the literature, such as IP over ATM over WDM and IP over SDH/SONET over WDM; and recently, a lot of literature has proposed IP over WDM.

The rate of change of technology also impacts the selection of core network technology. For instance, today's implementation of IP networks makes use of different transport techniques, embodying IP, frame relay, ATM, SDH/SONET, and WDM. If one minimizes network elements (e.g., IPoWDM), it may be more cost
effective, but there is more risk of obsolescence of investments. However, such a risk may be small if IP still remains the predominant traffic type.

Since major carriers around the world have installed WDM equipment on each end of the fiber, the next logical step is to build a WDM cross-connect to route multiple-wavelength signals. One of the technologies for a photonic cross-connect is directional coupler. Like an optical fiber, a directional coupler can pass multiple-wavelength signals, which makes it ideal for a WDM cross-connect.

An electro-optical directional coupler is physically implemented by fabricating two waveguides close to each other. It consists of two optical inputs, two optical outputs and one electrical control input. In a simplified view, the voltage of the control input puts the device in either one of the states: the bar state or the cross state. In the bar state, the upper (lower) input is coupled to the upper (lower) output. Directional couplers are used as 2x2 switching elements (SE) and are connected by optical waveguides (links) to construct large optical switches. Here we introduce a class of switches called \( \text{Log}_2(N,m,p) \) networks. Many characteristics of \( \text{Log}_2(N,m,p) \) networks, \( \text{Log}_2N \) stages, regular topology and fault tolerance, are attractive for directional-coupler-based photonic switching technologies. Much research work to construct non-blocking \( \text{Log}_2(N,m,p) \) networks has been carried out. In chapter 2, we will present our finding of building up non-blocking \( \text{Log}_2(N,m,p) \) networks for multicast traffic.

1.3 ATM switching

The asynchronous transfer mode (ATM) technique has been standardized and widely accepted as a basis for transporting and switching the user's information in broadband-ISDN.
1.3.1 ATM switching architecture

Many ATM switch architectures have been proposed [20] and can be basically classified into two main categories: time-division architecture and space division architecture. Due to the problems of time-domain architecture such as scalability, here we only introduce space-domain architectures in detail.

Switches in this category provide better performance than that of time-division architecture. The switch fabric of space-division architecture is usually composed of small building blocks called switching elements (SEs). These SEs are interconnected by internal links according to different topologies to form different switching networks. Each incoming cell will pass through a number of these SEs and finally reach their desired output port. Cell routing inside the switch fabric can be achieved by a centralized or distributive controller, but the latter is always preferred because of its capability of scalability. According to the different network topologies of the switch fabric, this category can be further divided into single-stage and multi-stage switches.

- Single-stage Space Division Switches

In a single-stage space-division switch, the switch fabric is just a single SE. The well-known example of this type is the crossbar switch (see Fig. 1.1). Crossbar switches are widely used for switched backplanes because of their simplicity. A N×N crossbar switch has N rows and N columns. At the intersection of each row and column, a crosspoint is present. Each crosspoint can be either in the cross state or the bar state. Cell routing is performed by modifying the state of the crosspoints.
Since each input and output pair is joined by a crosspoint, a path can always be established between any input-output pair. The crossbar switch can satisfy any connection request when all incoming cells have unique destinations. Owing to this property, the crossbar switch is said to be nonblocking. Blocking only happens when two or more cells are destined to the same output, called destination blocking. Under this circumstance, only one of the cells is transmitted and the others are blocked. The crossbar switch has numerous advantages, e.g. simple in structure, internal nonblocking, modular and high performance. However, its hardware complexity restricts its applications. The hardware complexity of a crossbar switch is proportional to the square of its size. This makes it impractical for large size switches.

- Multi-stage Space Division Switches

Multi-stage space division switches were proposed to overcome the problem of hardware complexity in crossbar switches. In a multi-stage switch, the switch fabric is formed by many small identical non-blocking SEs arranged into stages with internal links connecting the adjacent stages (See Fig. 1.2). Because of this, the number of crosspoints is much less than that of a crossbar switch with same size. The internal links act as shared paths serving all cells moving from one SE to another. This design reduces the hardware complexity with the tradeoff of performance. When
more than one cell access the same internal link, some of them are blocked. As a result, even cells destined to an idle output may be blocked at the internal links. This is called internal blocking. A multi-stage switch suffers from both internal blocking and destination blocking, that leads to lower performance than that of a non-blocking switch.

![Diagram of a 3-stage space division switch fabric](image)

**Fig. 1.2** A 3-stage space division switch fabric

### 1.3.2 Queueing Strategies

As discussed above, blocking can occur in any kind of space-division switches. Blocked cells are discarded unless buffers are provided for temporary storage. The allocation of buffers in ATM switches is an important consideration to the overall performance. There are three classes of buffering strategies: input buffering, output buffering and internal buffering. These three different classes can be used in combination, e.g. the input-output buffering.

- **Input Queueing**

With an input queueing switch, the arriving cell enters a buffer located at its input port unless buffer space is not available. A special phenomenon called head of line (HOL) blocking occurs in this kind of switches. This happens if the input buffer
employs first-in-first-out (FIFO) discipline, in any time slot, while a cell is waiting for its turn to be transmitted, other cells behind may be blocked despite the fact that their destinations are idle at that moment. Note that HOL blocking limits the maximum throughput of input queueing switch to 0.586.

- **Output Queueing**

  Similar to input queueing switches, output queueing switches have buffers situated after the switch fabric and in front of the output links. These buffers are essential for switches with switch fabric capable of transmitting more than one cell to each output port per time slot. This can be achieved by speeding up the switch fabric by a factor, say N.

  Although output queueing has a much better performance, it is usually more complex and, thus, difficult to design. In addition, for the speed-up method, the output memories must have an N-time speed of the external links for the speedup factor equals N. As a result, output queueing switches do not scale well. On the other hand, memory speed is not a big problem for input queueing switches. In our research, we deployed input queueing and concentrated our effort on improving the performance of input-queued switches.

- **Combined Input-Output Queueing**

  In order to achieve an acceptable performance while avoiding the complexity problem associated with pure output queueing, many switch designs employ a combined input-output queueing strategy. This can be achieved by speeding up a switch by a factor of T, where 1<T< N. By doing so, up to T cells can be switched to any output port during one time slot. The performance is worse than that of a pure output queueing switch, but better than that of a pure input queueing switch [21]. In
[21], it is reported the maximum throughput equal 0.885, 0.976 and 0.996 for T=2,3 and 4 respectively, assuming infinite buffers at input and output ports.

1.4 Thesis Organizations

The rest of the thesis is composed of four chapters. The method for constructing nonblocking multicast Log$_2$(N,m,p) networks is given in Chapter 2. After investigating different scheduling algorithms, particularly iSLIP, we present our scheme, the performance of which was explored for single-stage and multi-stage networks with unicast traffic in Chapter 3. The same method applied to multicast switches is discussed in chapter 4. Finally, in chapter 5 the conclusion and remarks about possible work to be done in future is presented.
Chapter 2 Multicast $\log_2 N$ networks

2.1 Introduction

The deployment of fiber optics as a transmission media has created an imbalance between transmission speed and switching speed. This speed mismatch has incited new efforts in switching architecture research. Recently, self-routing networks have attracted many researchers’ attention due to their two characteristics: 1) self-routing and 2) $O(\log_2 N)$ stages between each inlet-outlet pair. The self-routing capability can remove the time and space bottleneck in conventional RAM-controlled switching systems [9]. The small number of switching element ($O(\log_2 N)$ stages compared to $O(N)$ stages in a crossbar) between an inlet-outlet pair makes self-routing networks attractive for photonic switching systems based on nonlogic photonic switching devices where signal attenuation is proportional to the number of devices a light signal passes through. There are three topologically equivalent self-routing networks widely used, banyan, baseline and shuffle networks, as shown in Fig. 2.1.
Apart from all the advantages stated above, one problem of many self-routing networks is that they are blocking in the sense that a call can be blocked even if both the inlet and outlet are idle. It is shown in [9] that strictly nonblocking and rearrangeable nonblocking [10] can be created by vertically stacking these self-routing networks. These networks are called multi-Log₂N networks. Later it is found that if extra stages are added on to the back of the basic self-routing Log₂N networks, the total amount of hardware needed to make a rearrangeable nonblocking network can be reduced. Therefore, the design principle can be considered as a combination of horizontal decomposition and vertical stacking.

[7] extended the extra-stage approach to the strictly nonblocking network based on the vertical stacking scheme. The proposed networks are named Log₂(N,m,p) network. The sufficient and necessary conditions for the Log₂(N,m,p) network to be strictly nonblocking are presented in [7]. The result listing in Table 2.1 in [7] indicates that there is an optimal value of m which can minimize the hardware cost for the Log₂(N,m,p) strictly nonblocking network.
Table 2.1 THE VALUES OF p REQUIRED FOR THE (N,m,p) NETWORKS TO BE STRICTLY NONBLOCKING FOR DIFFERENT VALUE OF N AND m

<table>
<thead>
<tr>
<th>m/N</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td>15</td>
<td>23</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results in the above table are only applicable to one-to-one connection (i.e. unicast). The research attention of extending the traffic to one-to-many (i.e. multicast) connections has been considered in [8]. The authors in [8] presented their method of splitting a one-to-many connection uniquely into subconnections and using subconnection-intersection graph to represent the blocking among the connections in a Log₂N networks. Results for making multicast multi-Log₂N network strictly nonblocking are given in formula in [8]. Here we also list them in a table (see Table 2.2) so that they ca be easily compared with the above results and our results quoted in Section 2.3.

Table 2.2 THE VALUES OF p REQUIRED FOR MULTICAST MULTI-Log₂N NETWORKS TO BE STRICTLY NONBLOCKING

<table>
<thead>
<tr>
<th>m/N</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>25</td>
<td>33</td>
<td>65</td>
</tr>
</tbody>
</table>

As stated above, methods for constructing unicast (one-to-one) nonblocking Log₂N self-routing networks have been extensively studied [6-7]. But, the results of multicast (one-to-many) are still underway. Ref [8] recently presented a method to
build nonblocking multicast networks using multiple Log_2 N networks vertically stacked together. In this part of the thesis, we investigate a more general class of networks, called Log_2(N,m,p) networks, where N denotes the number of inputs (or outputs) with N a power of 2 (let n=\log_2 N), m denotes the number of extra stages cascaded on to the Log_2 N networks, and p denotes the needed number of planes vertically stacked for nonblocking. Since Log_2 N networks are only a special case of Log_2(N,m,p) networks, our results provide a more powerful tool than that given in [8] for building a multicast network with the O(Log_2 N)-stage characteristic.

The remaining of this chapter is organized as follows. In section 2.2, the definitions of multicast Log_2(N,m,p) networks based on our proposed method are presented. This is followed by an analysis of the worst case traffic pattern. Results and concluding remarks are presented in Section 2.3.

2.2 Our Proposed Method for multicast Log_2(N,m,p) networks

2.2.1 Splitting a Multicast Connection into Sub-connections

In [8], the authors broke up a multicast connection into sub-connections and treated each sub-connection independently. This successfully led to a nonblocking multicast network with a reasonable complexity. We will use a similar approach. However, we face a more difficult problem because the optimal way to break a multicast connection is not easy to find in a Log_2(N,m,p) network. In the analysis, we use the bipartite graph topology convention [2], in which each link is represented by a vertex and the edges between the vertexes represent switching elements. We use the baseline topology shown in Fig. 2.2 for our discussion.
The topology of a $\log_2(N,m,p)$ network is that the extra $m$ stages are the mirror stages of the first $m$ ones. In our study, we number the stages from 0,1, ..., $n+m$. Multiple paths exist in a $\log_2(N,m,p)$ network. The graph formed by all the paths between any input and output pair is called the **Path Redundancy Graph (PRG)**. The PRG of a $\log_2(16,1,1)$ network is shown in Fig. 2.4. Let us arbitrarily pick up a vertex of one path in the PRG for illustration and without loss of generality, we use the first path. The selected vertex is called the **Reference Vertex** and the stage of this vertex is called the **Reference Stage (RS)**. All the inputs and outputs that can be reached from the reference vertex form a double binary-tree (see Fig. 2.5). Let OWS be the number of outputs in the right binary-tree (output tree). We then split the outputs into groups and each group has OWS outputs. We call each group an **Output Window** (OWS stands for the output window size). That is, output window $g$ will be the output set $\{OWS \times g, OWS \times g + 1, ..., OWS \times g + (OWS - I)\}$. Similarly we define IWS as the number of inputs in the left tree (input tree). We group the consecutive IWS inputs into an
input window. Thus the input window $f$ will be the input set \{\text{IWS} \times f, \text{IWS} \times f+1, \ldots, \text{IWS} \times f+(\text{IWS}-1)\}. An example is shown in Fig. 2.3 for RS=3.

Our approach of splitting a multicast connection into sub-connection is based on the output windows. Suppose the destinations of a multicast connection belong to $k$ output windows. We then break the multicast connection into $k$ sub-connections and each sub-connection contains all the outputs belonging to the same output window. Let $s, D$ be a multicast connection from input $s$ to a set of outputs $D$. In Fig. 2.3, we show an example of $<0,\{0,2,6,15\}>$ multicast connection. Using our breaking up approach, the multicast connection can be divided into three sub-connections---$<0, \{0,2\}>$, $<0, \{6\}>$, $<0, \{15\}>$---each output set belonging to a different output window. Shifting the reference stage to the right, as can be easily seen, will reduce the output window size and increase the input window size simultaneously. Shifting the reference stage to the left side will have the opposite effect. Finding the optimal reference stage is a crucial step in our study.

![Log$_2$(16,1,1) Network](image)

Fig. 2.3 The input and output window for $RS=3$, $IWS=8$, $OWS=4$
2.2.2 Definition and Characteristics of Sub-Connections

Let us call the new connection to be set up the tagged connection. Assume the reference stage is fixed. For a specific tagged connection, we need to find the maximum number of planes that can be blocked by the existing connections. Unlike the unicast networks, the worst case scenario of a multicast switch depends on the type of the tagged connection to be set up-unicast or multicast.

Define the weight of a vertex $w_i$ as the fraction of the corresponding PRG that will be blocked if that vertex is blocked (illustrated in Fig. 2.4). For example, the weight of a first stage vertex is $\frac{1}{2}$ since half of the PRG will be blocked if the vertex is already occupied by some existing connection. Since $w_i$ is symmetric to the middle stage, we can summarize $w_i$ into the following formula:

$$w_i = \begin{cases} 2^{-i}, & 1 \leq i \leq m \\ 2^{-m}, & m \leq i \leq (n+m)/2 \end{cases}$$

if $n+m$ is even (Eq. 2.1)

$$w_i = \begin{cases} 2^{-i}, & 1 \leq i \leq m \\ 2^{-m}, & m \leq i \leq (n+m-1)/2 \end{cases}$$

if $n+m$ is odd

When a multicast connection intersects with a vertex, this connection may correspond to several sub-connections due to the break-up rule we use. The multiplicity of an $i$-th stage vertex in the PRG, denoted by $m_i$, is defined as the
maximum number of sub-connections that can be derived from an intersecting multicast connection. Obviously, \( m_i \) depends on the reference stage we choose (illustrated in Fig. 2.5 and Fig. 2.6).

From the bipartite graph, it is a direct matter to derive \( m_i \) as the following:

\[
m_i = \begin{cases} \min(2^{n+m-i}, 2^n) / OWS, & i \leq j \\ 1, & i > j \end{cases} \quad \text{(Eq. 2.2)}
\]

where \( OWS = \min(2^{n+m-j}, 2^n) \) if the reference stage is \( j \).

### 2.2.3 Worst-Case Traffic Pattern

As pointed out previously, the type of tagged connection, unicast or multicast, will affect the result. We will begin the discussion with a unicast tagged connection. We then change it to a multicast connection. The contrast between them will reveal what constitutes the worst case scenario.

The discussion is conducted by fixing the reference stage value first. Once the worst case scenario is identified and the maximum number of planes blocked by existing connections is computed, we then shift the reference stage from 1 to \((n+m-\)

19
to find the optimal value of reference stage which leads to the minimum $p$ for making network nonblocking.

- **Unicast tagged connection**

  Assume that the reference stage is fixed and that the tagged connection is unicast. Without loss of generality, we assume that the connection is from input 0 and output 0. Since each path in the PRG has the same property, we only show one path as shown in Fig. 2.7. An intersecting connection can intersect with more than one vertex, for example, connection $<1,1>$ (see Fig. 2.7(a)). However, to maximize the number of blocked planes (the worst case scenario), the intersecting connection should intersect only one vertex, thus releases some inputs (or outputs) to block more planes (see Fig. 2.7(b)).

Comparing the two cases in Fig. 2.7, it is found that output 1 can add blocking to the tagged connection. The result is summarized below.

![Diagram](image)

Fig 2.7. (a) Input 1 and output 1 combined will block only half plane (b) Now input 1 and output 1 combined block one plane.

**Property 1.** In the worst case scenario, an intersecting connection will only intersect the PRG with one vertex.
Under Property 1, for a particular vertex of the PRG, all the intersecting paths inputs and outputs form a double binary-tree (see Fig. 2.7). Because each connection must have one input and output, the number of intersecting paths of an i-th stage vertex in the PRG, denoted by $s_i$, will be the smaller cardinal of the two sets (the cardinal of a set is defined as the number of elements of a set). $s_i$ can be easily derived from the topology of the network into the following equation:

$$s_i = \min(2^{i-1}, 2^{n+m-1-i}) \quad (\text{Eq. 2.3})$$

Given $w_i$, $m_i$, and $s_i$, we can compute the total number of blocked planes, denoted by $q$, under the worst case scenario as:

$$q = \left\lceil \frac{n+m-1}{\sum_{i=1}^{n+m-1} s_i \times m_i \times w_i} \right\rceil \quad (\text{Eq. 2.4})$$

- **Multicast Tagged Connections**

We change the tagged connection to a multicast connection. Note that because of our break-up rule, we only need to consider the case in which the outputs of the tagged connection fall within one output window.

Let us add one output, say output 4, to the tagged connection in the previous unicast example (see Fig. 2.8(a) and Fig. 2.8(b)). Note that outputs 0 and 4 must be put on the same plane according to our break-up rule. As a result, $s_4$ is reduced by 4, but $s_5$ and $s_6$ are increased by 2 and 1. Since $w_6 = 1/2$, $w_5 = 1/4$ and $w_4 = 1/8$, the total impact of blocking is thus increased. Following the same reasoning and comparing the difference between reducing $w_i$ and increasing $s_i$, we can conclude the multicast tagged connection shown in Fig. 2.8(c) will generate the maximum blocking impact that corresponds to the worst case scenario.
Once the worst case tagged connection is identified, the last step in our computation is to find the optimal reference stage that leads to the minimum number of blocked planes. This can be achieved by shifting the reference stage from 1 to \((n+m-1)\). Adding 1 to the result listed in Eq. 2.4, we have the number of planes needed for making a nonblocking multicast switching network.

### 2.3 Results and Concluding Remarks

Table 2.3 tabulates the number of planes required for nonblocking multicast Log\(2(N,m,p)\) networks. Adding extra stages, as shown in Table 2.3, does not always reduce the number of planes for making a nonblocking multicast network. Another observation is that the optimal reference stage for a particular Log\(2(N,m,p)\) network
does not appear to follow a regular pattern. Also note that the reference stage value determines how we break up a multicast connection. This will affect the connection setup time. Compared to the results in Table 2.2, we can see that better results were obtained in some cases than those in [8] because we used dynamic reference stages but [8] fixed the window size for any N.

Table 2.3 THE VALUES OF p REQUIRED FOR MULTICAST (N,m,p) NETWORKS TO BE STRICTLY NONBLOCKING FOR DIFFERENT VALUE OF N AND m

<table>
<thead>
<tr>
<th>m/N</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2(RS=1)</td>
<td>3(RS=1)</td>
<td>5(RS=2)</td>
<td>8(RS=2)</td>
<td>13(RS=3)</td>
<td>20(RS=3)</td>
<td>33(RS=4)</td>
<td>48(RS=4)</td>
</tr>
<tr>
<td>1</td>
<td>2(RS=2)</td>
<td>3(RS=2)</td>
<td>4(RS=2)</td>
<td>7(RS=2)</td>
<td>10(RS=3)</td>
<td>17(RS=3)</td>
<td>24(RS=4)</td>
<td>40(RS=4)</td>
</tr>
<tr>
<td>2</td>
<td>3(RS=3)</td>
<td>4(RS=3)</td>
<td>5(RS=3)</td>
<td>9(RS=4)</td>
<td>11(RS=4)</td>
<td>21(RS=5)</td>
<td>25(RS=5)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4(RS=4)</td>
<td>6(RS=4)</td>
<td>9(RS=4)</td>
<td>14(RS=5)</td>
<td>20(RS=5)</td>
<td>32(RS=6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7(RS=5)</td>
<td>11(RS=5)</td>
<td>16(RS=6)</td>
<td>24(RS=6)</td>
<td>36(RS=7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>11(RS=6)</td>
<td>16(RS=7)</td>
<td>25(RS=7)</td>
<td>37(RS=8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>16(RS=8)</td>
<td>26(RS=8)</td>
<td>38(RS=9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>26(RS=9)</td>
<td>38(RS=10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>38(RS=11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Log₂(N,m,p) networks will play an important role in a future WDM optical network since they are the most suitable architecture for photonic switches based on optical directional couplers which can pass WDM signals. The results provided in this part of the thesis allow us to determine the best design tradeoffs among some important parameters such as attenuation (related to how to break a multicast connection), crosstalk (related to the number of stages), and the total amount of hardware.
Chapter 3
Scheduling algorithms for crossbars

3.1 Introduction

The Internet requires fast switches and routers to handle the increasing congestion. One emerging strategy to achieve this is to merge the strengths of ATM and IP: building IP routers around high-speed cell switches. Each of these high-speed switches and routers is built around a crossbar switch that is configured using a centralized scheduler, and each uses a fixed-size cell as transfer unit. Variable length packets are segmented as they arrive, transferred across the central switching fabric, then reassembled again into packets before they depart. A crossbar switch is widely used because it is simple to implement and is internally non-blocking: it allows multiple cells to be transferred across the switch fabric simultaneously.

A switch consists of three parts: (i) input queues to buffer cells arriving at the input links; (ii) output queues to buffer cells going out on output links; (iii) switch fabric to transfer the cells to the desired outputs. The switch fabric operates under a scheduling algorithm that arbitrates among the cells from different inputs destined to the same output. A lot of work has been done in designing these three parts of a switch; Each has its own set of advantages and disadvantages. It is well known that output-queued switches requires the output-queues and the internal interconnect have a bandwidth equal to N times the link rate. Since memory bandwidth is not increasing as fast as the demand for network bandwidth, this architecture becomes impractical for very high-speed switches. Moreover, numerous papers have indicated that by using non-FIFO input queues and by using good scheduling polices, much higher throughputs are possible [23,24]. Therefore, input-queued switches are of increasing
interest to the research and development community. The performance of input-queued switch depends very much on scheduling scheme, and especially depends how the input queues are organized. Therefore, to improve switch performance, our research focus has been the input buffer organization, and the scheduling algorithm.

If the input queue is a simple, single FIFO queue, there is a popular perception that input-queued switches suffer from low performance due to the head of line (HOL) blocking, illustrated in Fig. 3.1. In a FIFO input queue, a cell destined to an output that is free may be held in line behind a cell that is waiting for an output that is busy. Even with benign traffic, it is well-known that the HOL can limit the throughput to \(2 - \sqrt{2} \approx 58.6\%\) [12]. However, if the input queue is organized as multiple queues, for example, as one per destination port, the so-called Virtual-Output-Queue (VOQ) (see Fig. 3.2), then we come closer to the concept of an output queueing scheme, even though these queues are physically located at the input side. As is widely known, output queueing provides an ideal performance. The extent to which the performance of input queueing with multiple queues per input measures up to that of output queueing largely depends on how the input queues are controlled and scheduled.

![Diagram of HOL blocking](image)

HOL blocking: even the path for the second cell destined for output 3 is available, it cannot be sent since the first cell has not been sent yet.

Fig. 3.1. Illustration of HOL blocking
The remaining part of this chapter is organized as follows. First, the simulation model is presented in Section 3.2. What follows in Section 3.3 is the introduction of a scheduling algorithm called iSLIP [13], which deploys the VOQ scheme and can achieve 100% throughput. However, the complexity of the scheduling algorithm is very high. In addition, it is not practical for large-scale sized switches. Then, in Section 3.4, we investigate a novel scheme in which queue splitting and a simple scheduling algorithm, random selection is deployed. Conclusions are presented in section 3.5.

3.2. Simulation Model and Performance Methodologies

3.2.1 Simulation Model

We choose conventional discrete-event simulation for modeling ATM switches. Fig.3.3. shows the diagram of our event-driven simulation model.
The simulation model and simulation environment is specified as follows:

1. Time is slotted and the slot size is the same as the cell transmission time of the incoming link, it is also the time of unit 1 in our simulation model.

2. Space parallelism is related to internal blocking of switch. Let us define $p$ as the probability that a cell can reach the destination is $p$ and $1 - p$ as the probability that a cell is blocked inside the switch. Since a crossbar is used as our space switch, here $p = 1$, that is, crossbar switch is internally nonblocking.

3. The uncorrelated input traffic and correlated input traffic arrive at the input ports by independent and identically distributed Poisson process with offered load $\alpha$. Actually, a packet is a set of cells to be sent from the same source to the same destination, which forms correlation among each other. Here we assume that the packet length is of geometric distribution with mean length of $l$.

4. The virtual arrival queue at each input port, as shown in Fig. 3.3, is used to store the arriving cells. The cells are then scheduled from the virtual arrival queue to the real input buffer at the speed of one cell per one time slot. A cell is lost here if the input buffer overflows.

5. After cells enter the input buffer, they will become the head-of-line cells based on the FIFO discipline. When a cell becomes a head-of-line (HOL) cell, the transfer
process is initiated. If there are more than one HOL cells competing for the same destination, at most $T$ ($T$ is the speedup factor of the switch fabric, $1 \leq T \leq N$) cells can be selected for transmission within one time slot. The others will be blocked. Here scheduling algorithms will greatly affect the performance of the switch. We will investigate different scheduling schemes here, such as iSLIP and random scheduling.

### 3.2.2 Performance Methodologies

In our simulation, we chose three typical parameters to examine the performance of the switch fabric. All of the parameters refer to steady-state conditions.

- **Switch throughput $\rho$ ($0 < \rho \leq 1$):** switch throughput is defined as the probability that a cell received on an input link is successfully switched and transmitted to the addressed switch output. The maximum throughput $\rho_{\text{max}}$ indicates the offered load=1, that is, input traffic is saturated.

- **Average cell delay $D$ ($D \geq 1$):** average number of slots it takes for a cell received at a switch input to cross the network and to be transmitted downstream to the addressed output; The minimum value of $D=1$ indicates that the exact cell transmission time. $D$ takes into account the input queueing delay plus the cell transmission time.

- **Cell/Packet loss rate $\gamma$ ($0 < \gamma \leq 1$):** probability that a cell/packet received at the switch input is lost due to the buffer overflow. Since in our thesis, we mainly focus on the maximum throughput in different cases, the cell loss rate is appropriate to be chosen as the measurement. Therefore, a cell is either lost or be
successfully transmitted to its destination in the steady state. The equation \( p + \gamma = \alpha \) holds.

Our simulation monitored throughput, delay and cell loss. Simulations were run for different scheduling algorithms, switch structure (single-stage crossbar and multi-stage crossbar), buffer size, traffic types and offered load.

### 3.3 iSLIP Algorithm with queue-splitting

#### 3.3.1 iSLIP with Single-iteration

In this section, we describe and evaluate the iSLIP algorithm [13], concentrating on the behavior of iSLIP with just a single iteration per cell time. Later, we will discuss the iSLIP with multiple iteration.

As we stated before, iSLIP employs a virtual output queue (VOQ) scheme to achieve better performance of input-queue switch fabric since VOQ can dramatically eliminate the HOL blocking. In terms of scheduling, it is a variation of the basic round-robin matching algorithm (RRM). RRM is perhaps the simplest and most obvious form of iterative round-robin scheduling algorithms, comprising a two-dimensional array of round-robin arbiters: cells are scheduled by round-robin arbiters at each output, and each input. As it shall be seen, RRM does not perform well; But it helps to demonstrate how iSLIP performs, so we start here with a description of RRM. The RRM algorithm consists of three steps. As shown in Fig. 3.4, each round-robin schedule contains \( N \) ordered elements in an \( N \times N \) switch. The three steps of arbitration RRM are:

**Step 1. Request.** Each input sends a request to every output for which it has a queued cell.
Step 2. Grant. If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request is granted. The pointer $g_i$ to the highest priority element of the round-robin schedule is incremented (module $N$) to one location beyond the granted input.

Step 3. Accept. If an input receives a grant, it accepts the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The pointer $a_i$ to the highest priority element of the round-robin schedule is incremented (module $N$) to one location beyond the accepted output.

![Diagram of the RRM matching process]

a) Step 1: Request. Each input makes a request to each output for which it has a cell.
Step 2: Grant. Each output selects the next requesting input at or after the pointer in the round robin schedule. Output arbiters are shown here for output 2 and 4. Input 1 and 3 both request output 2, since $g_2 = 1$, output 2 grant input 1. $g_2$ and $g_4$ are updated to favor the input after the one that is granted.

![Diagram of the RRM matching process]

b) Step 3: Accept. Each input selects at most one output. The input arbiter for input 1 is shown. Since $a_1 = 1$, input 1 accept output 1. $a_1$ is updated to point to output 2.

c) When the arbitration has completed, a matching size of tow has been established. It is not the maximum matching

Fig. 3.4 Example of the three steps of RRM matching
However, the RRM matching algorithm has poor performance when the traffic load is heavy. The reason for this fact lies in the rules for updating the pointers at the output arbiters. The synchronization phenomenon greatly reduces the performance. Therefore, in order to improve the performance of RRM, synchronization should be avoided or at least eliminated.

Then comes iSLIP that improves RRM by reducing the synchronization of output arbiters. iSLIP achieves this by not moving the grant pointers unless the grant is accepted by an input. iSLIP is identical to RRM except for condition placed on updating the grant pointers. The three steps of iSLIP are stated as follows:

**Step 1. Request.** Each input sends a request to every output for which it has a queued cell.

**Step 2. Grant.** If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer $g_i$ to the highest priority element of the round-robin schedule is incremented (module N) to one location beyond the granted input *if, and only if, the grant is accepted in Step 3.*

**Step 3. Accept.** If an input receives a grant, it accepts the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The pointer $a_i$ to the highest priority element of the round-robin schedule is incremented (module N) to one location beyond the accepted output.

### 3.3.2 iSLIP with Multiple-iteration
With more than one iteration, the iterative iSLIP algorithm improves the performance: each iteration attempts to add connections not made by earlier iterations. When multiple iterations are used, it is necessary to modify the iSLIP algorithm. The three steps of each iteration operate in parallel on each input and output as follows:

**Step 1. Request.** Each **unmatched** input sends a request to every output for which it has a queued cell.

**Step 2. Grant.** If an **unmatched** output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer $g_i$ to the highest priority element of the round-robin schedule is incremented (module N) to one location beyond the granted input **if, and only if, the grant is accepted in Step 3 of the first iteration.**

**Step 3. Accept.** If an **unmatched** input receives a grant, it accepts the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The pointer $a_i$ to the highest priority element of the round-robin schedule is incremented (module N) to one location beyond the accepted output.

### 3.3.3 iSLIP with different queue-splitting size

In [13], iSLIP employs a virtual output queue (VOQ) mechanism in which each input is assigned $N$ sub-queues, one for each output port. In this way, correlation and HOL blocking are greatly reduced compared to single input queue per port. iSLIP can work, under some circumstances, with the same effect as output-queued scheme to some extent. But the extreme division of input queue to $N$ sub-queue brings about the complexity problem and causes a bottleneck in the design. Our consideration of whether there was a tradeoff of VOQ scheme led us to propose the queue-splitting
scheme in which the input queue is split into multiple sub input queues, say 2 or 4, not necessarily N. It can improve the throughput of input-queued switches for both uncorrelated traffic and correlated traffic. Which of the sub queues the coming cell is assigned to depends on some bits of its destination. Suppose in a two-queue scheme, which queue the cell is assigned to depends on the last bit of its destination. The cells can be divided into non-overlapped sets and the out of sequence problem can be avoided since cells belonging to one particular packet will be assigned to same queue consecutively.

Our queue-splitting scheme only modifies the input organization from VOQ to a number of sub-queues per input buffer ranging from 2 to N. The scheduling procedures are the same as [13]. So we call it the iSLIP with queue-split scheme. The performance of the iSLIP with queue-split is studied in Section 3.3.4.

3.3.4 Performance evaluation of iSLIP

- Maximum throughput

The Simulation results for uncorrelated traffic and correlated traffic with the mean packet length 10 are listed in Table 3.1 and 3.2. Maximum throughput was obtained under heavy traffic load and with an infinite buffer size.

There are several points that can be derived from the two tables. First of all, as expected, maximum throughput that can be achieved increases as the queue-splitting size and the number of iteration increase. For example, when queue-splitting size and number of iterations is from 2 to 16, the maximum throughput can reach from around 0.72 to 0.99. Secondly, it is also well accepted that performance degrades when the switch size increases. Thirdly, by comparing Table 3.1 with Table 3.2, it can be seen that correlation will inevitably cause performance degradation.
Let take the case of queue-split size as N for more consideration. We can achieve as good throughput as almost 100%, shown in Table 3.1. However, under low load, iSLIP's performance is not good; arriving cells usually find empty input queues, and on average, there are only a small number of inputs requesting a given output. As the load increases, we can expect the pointers to move away from each other, making it more likely that a large matching will be found quickly in the next cell time. In fact, under uniform 100% offered load, the iSLIP arbiters adapt to a time-division multiplexing scheme, providing a perfect match. It is demonstrated in the tables that for an N×N switch, it takes about \( \log_2 N \) iterations for iSLIP to converge, as pointed out in [13].

<table>
<thead>
<tr>
<th>Switch size</th>
<th># of iteration</th>
<th># of queue split</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td></td>
<td>64.5%</td>
<td>72.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>66.4%</td>
<td>82.2%</td>
<td>84.5%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>71.4%</td>
<td>85.8%</td>
<td>93.0%</td>
<td>93.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td>98.8%</td>
<td>98.9%</td>
<td>99.0%</td>
<td>99.1%</td>
<td>99.3%</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td></td>
<td>63.4%</td>
<td>71.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>64.2%</td>
<td>80.8%</td>
<td>82.9%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>65.1%</td>
<td>84.8%</td>
<td>90.8%</td>
<td>90.8%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td>70.8%</td>
<td>88.1%</td>
<td>95.9%</td>
<td>96.0%</td>
<td>96.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>98.4%</td>
<td>98.8%</td>
<td>99.0%</td>
<td>99.1%</td>
<td>99.2%</td>
<td>99.3%</td>
</tr>
</tbody>
</table>

Table 3.1. Maximum throughput of iSLIP for uncorrelated traffic

<table>
<thead>
<tr>
<th>switch size</th>
<th># of iteration</th>
<th># of queue split</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td></td>
<td>60.7%</td>
<td>67.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>64.9%</td>
<td>79.0%</td>
<td>81.3%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>67.9%</td>
<td>85.9%</td>
<td>91.0%</td>
<td>91.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td>96.4%</td>
<td>97.5%</td>
<td>97.8%</td>
<td>97.9%</td>
<td>97.9%</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td></td>
<td>59.5%</td>
<td>65.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>62.7%</td>
<td>76.1%</td>
<td>78.4%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>63.7%</td>
<td>82.7%</td>
<td>87.4%</td>
<td>87.4%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td>66.5%</td>
<td>85.6%</td>
<td>92.7%</td>
<td>92.7%</td>
<td>92.7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>84.5%</td>
<td>91.0%</td>
<td>95.6%</td>
<td>95.6%</td>
<td>95.6%</td>
<td>95.6%</td>
</tr>
</tbody>
</table>

Table 3.2. Maximum throughput of iSLIP for correlated traffic
- Delay

Fig. 3.5 and Fig. 3.6 show the average cell delay as a function of offered load for the cases with queue-splitting size 4 and 16.

In each figure, the performance improvement is illustrated as the number of iterations is increased from 1-iteration to 4-iteration. This indicates that multiple iterations of iSLIP significantly increase the size of matching and therefore reduce the queueing delay.

A comparison of Fig. 3.5 and 3.6 shows that a larger queue-splitting size leads to better performance. When the queue splitting size reaches $N$, perfect performance in terms of throughput and delay function can be achieved.

![Graph showing average delay versus offered load](image)

Fig. 3.5 Average delay versus offered load for 16*16 network with iSLIP, queue-split size=4, uncorrelated traffic
Fig. 3.6 Average delay versus offered load for 16*16 network with iSLIP, queue-split size=16, uncorrelated traffic

- **Cell loss**

  The effect of the buffer size to cell loss is illustrated in Fig. 3.7 to 3.11. Fig. 3.7 and 3.8 show the case of queue-splitting size 4, for uncorrelated traffic and traffic with correlation correspondingly. Similarly, Fig. 3.9 and 3.10 illustrate the case of queue-splitting size 16. Fig. 3.11 illustrates the impact of speedup factor.

  Note that when the buffer size increases, the cell loss drops. Multiple iteration increases the number of matching and contributes to higher throughput and lower cell loss.

  For uncorrelated traffic, the cell-loss curves drop sharply and then tends to be constant. On the other hand, for traffic with high correlation, the demand for buffer
size is relatively large in order to get the cell loss drop perceptibly. As expected, correlation leads to performance degradation.

With a larger queue-splitting size and more iteration, lower cell loss can be obtained. There are several ways to reduce the degradation caused by traffic correlation. One way is to increase the speedup factor of the switch fabric. From Fig. 3.11, it can be seen that both the uncorrelated traffic and correlated traffic perform perfectly if speedup=2. With a buffer increase, the throughput quickly tends to 100%. The cost is that the switch fabric needs to be speeded up, which is quite expensive.

![Graph showing cell loss versus buffer size](image)

Fig. 3.7 Cell loss versus buffer size for 16*16 network with iSLIP, queue-split size=4, uncorrelated traffic
Fig. 3.8 Cell loss versus buffer size for 16*16 network with iSLIP, queue-split size=4, correlated traffic.

Fig. 3.9 Cell loss versus buffer size for 16*16 network with iSLIP, queue-split size=16, uncorrelated traffic.
Fig. 3.10 Cell loss versus buffer size for 16*16 network with iSLIP, queue-split size=16, correlated traffic

Fig. 3.11 Cell loss versus buffer size for 16*16 network with iSLIP, speedup=2
3.4 Random scheduling with queue-splitting

In the previous section, we introduced iSLIP algorithm, an iterative algorithm that employs virtual output queueing and can achieve high throughput. Using round-robin arbitration, iSLIP provides fair access to output lines. By careful controlling the round-robin pointers, the algorithm can achieve 100% throughput for uniform traffic. When the traffic is non-uniform, the algorithm can quickly adapt to an efficient round-robin policy among the busy queues. However, there is a bottleneck in this scheme. The complexity of the scheduling algorithm is very high since one input port is divided into N sub-queues and each will send a request each cell time. So, the scheduling processor will have to process $N^2$ requests from $N^2$ queues. This is not practical when the size of the switch increases. And, from our simulation results, it can be seen that to achieve a better performance for both the uncorrelated traffic and correlated traffic, a large buffer is needed for each input port. The iSLIP algorithm is also sensitive to traffic pattern and switch size. Under heavy loads, it can achieve good performance. When the traffic load is low, its performance is not much better than random scheduling.

In this Section, we focus on a simple scheduling algorithm, random scheduling. Its complexity is much lower than that of the iSLIP, but the throughput is low. However, we can increase the throughput from the perspective of input queue organization. One of its most salient advantages is that it is applicable to both unicast and multicast traffic. In addition, to make a large size switch, multiple stages, cascaded together, can be used. Therefore, random scheduling is the best candidate of the existing scheduling algorithms for multi-stage switch.

Much work has been done to improve the throughput of input-queued switches from the input queue organization perspective. In [14], a window policy is
investigated and evaluated. It relaxes the strict FIFO queueing discipline of the input buffer to allow each input to scan multiple cells competing for transmission, starting from the HOL cell, up to a fixed number of cells, called window size. An approximate analysis of maximum throughput as a function of the window size has been carried out in [15]. Input expansion policy has been proposed in [16], in which each input port is expanded into $s$ cells before cells enter an asymmetric $N_s \times N$ switch, in a time slot, up to $s$ cells from each input queue can be presented to the switch fabric for contention. Another scheme called the cell-dropping policy has also been proposed in [16]. In this scheme, whenever $k$ cells are addressed to the same output port in a time slot, only one can be switched to the output port. The remaining $k-1$ cells are discarded. The result of this scheme shows that the destination correlation does not affect the throughput. The maximum throughput of a space-division packet switch with correlated destination is studied in [17]. It has been shown that, under FCFS discipline, the correlation decreases the throughput from 0.59 to 0.51 when ten consecutive cells are sent to the same destination.

As random scheduling is used, queue-splitting is still deployed for input queueing organization. Instead of assigning one separate queue for each output port at an input side which makes it the bottleneck of iSLIP [13], we assign multiple queues per input port, say 2 or 4 not necessarily $N$. We claim that the performance can be greatly improved with this scheme. Performance will be evaluated in more detail in the following sections.
3.4.1 Performance of single-stage network with queue-splitting scheme

The iSLIP employs a virtual output queue (VOQ) mechanism, in which each input is assigned $N$ sub-input-queues, one for each output port. Because of this, correlation and HOL blocking are greatly reduced compared to single input queue per port. The iSLIP can work, under some circumstances, with the same effect to some extent as output-queued scheme. But the extreme division of the input queue to N sub-queue brings about a complexity problem and becomes the bottleneck of the design. So we considered whether there was a tradeoff of VOQ scheme.

We propose dividing the input queue into multiple sub input queues, say 2 or 4, not necessarily $N$. This will improve the throughput of the input-queued switches for both uncorrelated traffic and correlated traffic. We assign the coming cell into one of these queues using some bits of its destination. If two-queue is employed, the queue the cell belongs to depends on the last bit of its destination. If this is done, the cells can be divided into non-overlapped sets and this avoids the out of sequence problem since cells belonging to one particular packet will be assigned to same queue consecutively. The simplest scheduling algorithm is random selection. In order to improve the throughput further more, iteration can also be practiced in our scheme.

To achieve fairness to each input port, there is one input pointer at each input port indicating which queue is started from in the current time slot. In each time slot, the queue that current input pointer points to is checked if it has HOL cell requesting to contend the access to the switch fabric in the first iteration. If the cell is selected to transmit, all of the other queues' requests are terminated in the remaining iterations. If no cell is selected, in the next round, the next queue is checked and attends the contention for the remaining idle outputs until the end of the last iteration or until
there is a cell selected to transmit. At the end of this time slot, the input pointer is updated to next available queue.

We discuss the single-stage network first. For a single-stage crossbar, the switch fabric itself is non-blocking. In this circumstance, blocking occurs when multiple cells are destined to one particular output, and there is also the well-known HOL blocking.

- Maximum throughput

Before we present the maximum throughput for random scheduling with queue split, we would first like to outline the traditional window scheme.

A higher throughput can be achieved by relaxing the strict FIFO discipline at the input buffers. Each input still sends, at most, one cell input the switch fabric per time slot, but does not necessarily send the first cell in its queue, and no more than one cell is allowed to pass through the switch fabric to each output. Consider a scenario in which, at the beginning of each time slot, up to the first $w$ cells (called window size) in each input queue sequentially contend for access to the switch outputs. The cells at the heads of the input queues contend first for access to the outputs. Those inputs not selected to transmit their first cells in their input queues then contend with their second cells for access to any remaining idle outputs, and so on. A window size of $w=1$ corresponds to input queueing with FCFS discipline. We implement the traditional window scheme based on our simulation model. Our simulation results of the maximum throughput for different window size and the analysis result quoted from [16] are listed in Table 3.3. It can be seen that our simulation results agree well with the analysis results in [16].
<table>
<thead>
<tr>
<th>W</th>
<th>w=1</th>
<th>w=2</th>
<th>w=3</th>
<th>w=4</th>
<th>w=6</th>
<th>w=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.60</td>
<td>0.71</td>
<td>0.77</td>
<td>0.80</td>
<td>0.85</td>
<td>0.87</td>
</tr>
<tr>
<td>32</td>
<td>0.59</td>
<td>0.70</td>
<td>0.76</td>
<td>0.80</td>
<td>0.84</td>
<td>0.87</td>
</tr>
<tr>
<td>Analysis</td>
<td>0.59</td>
<td>0.69</td>
<td>0.77</td>
<td>0.81</td>
<td>0.85</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Table 3.3 The maximum throughput of window scheme

The window scheme clearly provides higher throughput than the FIFO scheme because it decreases the effect of HOL blocking. The disadvantage of the window selection policy scheme is that it involves a serious out of sequence problem since FIFO discipline is not maintained.

The maximum throughput for 16×16 crossbar and 32×32 crossbar is listed in Table 3.4 and Table 3.5 respectively. Intuitively, the window scheme and our queue-split scheme with multi-iteration should show a similar improvement in throughput, provided the window size equals the number of the queue-split size and the number of iterations. In addition, with queue-split organization of the input queue, our scheme should demonstrate relatively better performance than the window scheme, since we divide the cells based on their destination into non-overlapping sub-queues while the window scheme does not have this property. Our simulation results demonstrated this point.

<table>
<thead>
<tr>
<th>switch size</th>
<th># of iteration</th>
<th># of split</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1</td>
<td></td>
<td>60.2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>61.8%</td>
<td>72.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>65.5%</td>
<td>75.8%</td>
<td>83.0%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>75.0%</td>
<td>81.0%</td>
<td>87.1%</td>
<td>91.4%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td>96.3%</td>
<td>96.5%</td>
<td>96.5%</td>
<td>96.7%</td>
<td>96.9%</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td></td>
<td>59.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>60.1%</td>
<td>71.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>61.8%</td>
<td>73.6%</td>
<td>81.7%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>65.5%</td>
<td>75.8%</td>
<td>84.1%</td>
<td>89.5%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td>74.9%</td>
<td>80.9%</td>
<td>87.2%</td>
<td>92.0%</td>
<td>94.9%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>97.0%</td>
<td>97.1%</td>
<td>97.2%</td>
<td>97.4%</td>
<td>97.5%</td>
<td>98.0%</td>
</tr>
</tbody>
</table>

Table 3.4 Maximum throughput for single-stage with queue splitting scheme
The maximum throughput for correlated traffic with a mean packet of 10 is also given below. Unsurprisingly, the correlation between traffic leads to a lower throughput. However, the throughput can be greatly increased with queue-splitting size and number of iteration growing. For example, without any effort, correlation with a mean packet length of 10 decreases the throughput to about 0.51, as pointed out in [17] and with the queue-splitting size and the number of iterations growing from 2 to 8, it can be upgraded from about 0.65 to 0.87.

<table>
<thead>
<tr>
<th>switch size</th>
<th># of iteration</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td># of queue split</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>52.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>54.4%</td>
<td>66.6%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>58.2%</td>
<td>71.1%</td>
<td>79.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>67.3%</td>
<td>76.6%</td>
<td>84.7%</td>
<td>89.5%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>95.8%</td>
<td>95.8%</td>
<td>96.0%</td>
<td>96.3%</td>
<td>97.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td># of iteration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>51.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>52.2%</td>
<td>65.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>54.2%</td>
<td>68.5%</td>
<td>77.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>57.8%</td>
<td>71.1%</td>
<td>81.0%</td>
<td>87.2%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>67.3%</td>
<td>76.5%</td>
<td>84.7%</td>
<td>90.6%</td>
<td>93.6%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>94.7%</td>
<td>94.8%</td>
<td>95.1%</td>
<td>95.6%</td>
<td>96.9%</td>
<td>97.9%</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5. Maximum throughput for single-stage with random scheduling, correlated traffic with mean=10

- Delay

Fig. 3.12 indicates the average cell latency curve as a function of the offered load is improved with multiple iteration used.
- Cell loss

When the buffer size is finite, there will be cell loss due to buffer overflow. The cell loss can be captured as the function of buffer size for different number of iterations. For each number of iteration, the cell loss first drops sharply when the buffer size increases and then keeps almost constant. That is to say, it reaches a steady state.
We have studied the performance of our proposed queue-splitting scheme plus random scheduling for a single-stage crossbar switch. We can improve the performance of input-queued switches significantly in terms of maximum throughput, average cell latency and cell loss. The strength of our scheme compared to iSLIP is that we probably do not need to maintain a separate queue for each output at each input port to eliminate the HOL blocking. In addition, iSLIP is not practical for constructing large size switches and is not applicable to mixed unicast and multicast traffic. On the contrary, these are the advantages of our scheme. We will apply the scheme to multi-stage crossbar switches and, later, to multicast packet switching networks.

3.4.2 Performance of multi-stage network with queue-splitting scheme

As introduced in Chapter 1, a multi-stage switch fabric is formed by many small identical non-blocking SEs arranged into stages with internal links connecting the adjacent stages. The number of crosspoints is much less than that of a crossbar switch of the same size. The internal links act as shared paths serving all the cells moving from one SE to another. This design reduces the hardware complexity with the tradeoff of performance. When more than one cell accesses the same internal link, some of them are blocked. As a result, even cells destined to an idle output may be blocked at the internal links. This is called internal blocking. A multi-stage switch suffers from both internal blocking and destination blocking. This leads to lower performance than that of non-blocking switches.
1. two-stage switches

The topology of a 16×16 two-stage crossbar with SE equal to 4 is presented in Fig. 3.15.

![Fig. 3.14 the topology of 16×16 two-stage switch with switching element 4](image)

- Maximum throughput

Since multi-stage switches experience both internal blocking and destination blocking, the performance gets worse as the number of stages increase. For example, for two-stage switches, only about 45.2% throughput can be obtained for pure input-queued switch, compared to about 58.6% for single-stage switches. In addition, a larger switch size leads to a poorer performance. For example, compare results for 64×64 with 256×256. But, with our proposed queue-splitting scheme, especially multiple iteration is exerted, the maximum throughput can be increased dramatically.

<table>
<thead>
<tr>
<th>Switch size</th>
<th># of iteration</th>
<th># of queue split</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1</td>
<td></td>
<td>45.2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>48.1%</td>
<td>64.2%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>49.9%</td>
<td>67.9%</td>
<td>78.8%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>61.7%</td>
<td>73.5%</td>
<td>82.6%</td>
<td>88.5%</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td></td>
<td>43.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>44.5%</td>
<td>62.2%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>46.0%</td>
<td>64.2%</td>
<td>77.1%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>49.7%</td>
<td>66.7%</td>
<td>79.0%</td>
<td>86.5%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td>57.7%</td>
<td>70.4%</td>
<td>80.4%</td>
<td>87.6%</td>
<td>91.8%</td>
</tr>
</tbody>
</table>

Table 3.6. Maximum throughput for two-stage with random scheduling, uncorrelated traffic
<table>
<thead>
<tr>
<th>Switch size</th>
<th># of iteration</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1</td>
<td>40.6%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49.3%</td>
<td>59.2%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>51.3%</td>
<td>64.1%</td>
<td>75.1%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>51.9%</td>
<td>65.1%</td>
<td>76.9%</td>
<td>85.6%</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td>38.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>39.5%</td>
<td>57.1%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>41.1%</td>
<td>59.6%</td>
<td>72.6%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>43.7%</td>
<td>63.7%</td>
<td>74.7%</td>
<td>81.4%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>49.0%</td>
<td>64.5%</td>
<td>76.2%</td>
<td>84.0%</td>
<td>85.8%</td>
</tr>
</tbody>
</table>

Table 3.7. Maximum throughput for two-stage with random scheduling, correlated traffic

- Delay

Fig. 3.15 and 3.16 presents the delay curves as a function of the offered load. It can been seen from these figures that multiple-iteration improves the maximum throughput, and meanwhile enhances the delay function. In contrast, a larger queue-splitting size and iteration number, a better performance ensures.

![Fig. 3.15 average delay versus offered load for 64*64 two-stage network with queue-split size=4](image-url)
Fig. 3.16 average delay versus offered load for 64*64 two-stage network with queue-split size=8

- Cell loss

The same effect as that of single-stage is demonstrated in Fig. 3.17. Multiple iteration significantly decreases the cell loss.

Fig. 3.17 cell loss versus buffer size for 64*64 two-stage network with queue-split size=4, uncorrelated traffic
2. Three-stage crossbar

The following figure gives the representation of three-stage 64×64 switches with a switching element equal to 4. The large size of switch can be constructed easily by horizontally cascading multiple stages, or by vertically increasing switching elements. If the SE is increased from 4 to 8, still 3 stages are kept, 512×512 switch is obtained.

![Figure 3.18 Topology of three-stage 64x64 switches with SE 4](image)

- maximum throughput

When the results of single-stage and two-stage are compared, it can be noted that when more stages is cascaded, there is a higher probability of internal blocking and lower performance obtained. For example, the maximum throughput of a single-stage, two-stage and three-stage switch ranges from around 0.58 to 0.43 and then 0.36. Despite of this, higher throughput can still be obtained by using queue-splitting scheme. This can be demonstrated in Table 3.8.
<table>
<thead>
<tr>
<th>switch size</th>
<th># of queue split</th>
<th># of iteration</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1</td>
<td>39.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>42.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>48.9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1</td>
<td>36.0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37.1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>39.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>44.9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.8 Maximum throughput for three-stage with random scheduling, uncorrelated traffic

<table>
<thead>
<tr>
<th>switch size</th>
<th># of queue split</th>
<th># of iteration</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1</td>
<td>35.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>43.6%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>1</td>
<td>32.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>33.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>35.6%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>39.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.9. Maximum throughput for three-stage with random scheduling, correlated traffic

- delay

Refer to Fig. 3.19 for the average delay as a function of the offered load. Similar conclusion can be drawn as that drawn in respect to single-stage and two-stage switches.

Fig. 3.19 average delay versus offered load for 64x64 three-stage network with queue-split size=4
• cell-loss

From Fig. 3.20, it can be seen that cell loss is quite severe for three-stage switches. Multiple iteration is deployed, cell loss can be greatly eliminated.

![Graph showing cell loss versus buffer size](image)

**Fig. 3.20** cell loss versus buffer size for 64×64 three-stage network with queue-split size=4, uncorrelated traffic

### 3.4.3 Approximate analysis of maximum throughput

Let us make some approximate analysis about the throughput of multi-stage switches, referring to [25]. An analysis of our queue-splitting scheme seems quite complicated, and it requires further effort to develop a precise model for the analysis.

![Diagram of multi-stage network](image)

**Fig. 3.21** Multi-stage network with d×d switching element
Assume the size of each switching node is $d \times d$ and that a setup request reaches at an inlet of the node is $\rho_i$ during the setup phase. The probability $p_j$ that there are $j$ requests generated at a node is given as:

$$ p_j = \binom{d}{j} \rho_i^j (1 - \rho_i)^{d-j} $$

Given that $j$ requests are generated at their inlet, the probability, denoted by $f_j$, that a particular output fails to receive one is the same as the probability that all the $j$ requests are destined for the remaining $d-1$ outlets. Thus we get

$$ f_j = (1 - \frac{1}{d})^j $$

Given $p_j, f_j$ we can then calculate $\rho_{i+1}$

$$ \rho_{i+1} = 1 - \sum_{j=0}^{d} p_j f_j $n
$$

$$ = 1 - \sum_{j=0}^{d} \binom{d}{j} \rho_i^j (1 - \rho_i)^{d-j} (1 - \frac{1}{d})^j $$

$$ = 1 - (1 - \frac{\rho_i}{d})^d \text{(Eq. 3.1)} $$

Using Equation 3.1, the throughput of unbuffered networks of various sizes can be calculated. It can be seen that $\rho_{i+1}$ quickly approaches $1 - e^{-\rho}$ as $d$ increases.

Let us verify our simulation results. For single-stage switch, the maximum throughput can be obtained by letting $\rho_0 = 1$, as equation 3.2.

$$ \rho_{1\text{max}} = 1 - \left(1 - \frac{\rho_0}{N}\right)^N \Bigg|_{\rho_0 = 1} = 1 - \left(1 - \frac{1}{N}\right)^N \text{ (E.q. 4.2)} $$

where $N$ is the switch size, $\rho_{1\text{max}}$ quickly approaches $1 - e^{-\rho_0}$ as $N$ increases.
By letting $\rho_0=1$, it leads to $\rho_{1_{\text{max}}}=0.632$. The above analysis ignores the correlation between the consecutive cells. Taking the correlation among consecutive cells into account, a more accurate analysis shows that $\rho_{1_{\text{max}}}=0.586$. Simulation results for different switch size are listed in Table 3.10. Conclusion can be obtained that as $N$ increases, the throughput converges closely to 0.586. The simulation results agreed with our analysis.

<table>
<thead>
<tr>
<th>N</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>512</th>
<th>N-&gt;∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.586</td>
</tr>
<tr>
<td>Simulation</td>
<td>0.618</td>
<td>0.602</td>
<td>0.593</td>
<td>0.590</td>
<td>0.587</td>
<td>0.586</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.10 The maximum throughput of input-queued single-stage switch

Applying the same analysis as that for a single-stage switch, the approximate analysis results for multi-stage switch can be derived. If we consider the correlation between consecutive cells, the maximum throughput of the first stage is about 0.586, that is, $\rho_1=0.586$. Then we can get an approximation for the second stage and third stage as follows:

$$\rho_{2=1} \cdot e^{-0.586} = 0.444$$

$$\rho_{3=1} \cdot e^{-0.444} = 0.359$$

If we totally ignore the correlation, then

$$\rho_{2=1} \cdot e^{-0.632} = 0.469$$

$$\rho_{3=1} \cdot e^{-0.469} = 0.374$$
<table>
<thead>
<tr>
<th></th>
<th>2-stage</th>
<th></th>
<th>3-stage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64×64</td>
<td>256×256</td>
<td>N→∞</td>
</tr>
<tr>
<td>Analysis</td>
<td>0.456</td>
<td>0.450</td>
<td>0.444</td>
</tr>
<tr>
<td>Simulation</td>
<td>0.452</td>
<td>0.437</td>
<td>----</td>
</tr>
</tbody>
</table>

Table 3.11 Analytical results and simulation results of maximum throughput for multi-stage input-queued switch

3.5 Conclusion

In this chapter, we first discussed a scheduling algorithm, iSLIP, in which VOQ is used. Almost 100% throughput can be achieved with this algorithm. Stimulated by its queueing organization scheme, VOQ, we probe what the performance would be if multiple queues, say 2 or 4, are maintained at each input port, not necessarily N. A performance study shows that queue-splitting scheme can improve the performance of input-queued switch steadily.

Although iSLIP is perfect in terms of throughput, it is complex and does not scale well for large-size switches. Therefore, a simple scheduling algorithm, random selection was deployed with the same queue-splitting approach. The performance under this method for single-stage and multi-stage switches with unicast traffic was discussed. Simulation results showed that higher throughput, lower cell latency and cell loss could be achieved. We conclude that the queue-splitting scheme combined with random selection is a good candidate for input-queued architecture.
Chapter 4
Multicast Packet Switching

4.1 Introduction

In broadband ISDN, a wide range of services can be provided. It is expected that video services such as video-on-demand and teleconferencing will be the most important types of traffic in B-ISDN. Such video applications usually involve the transmission of information from one source to several destinations simultaneously, that is, multicasting. Therefore, multicasting is a required feature for such applications. As unicasting is a special case of multicasting, it is desirable to have a single switch capable of both unicasting and multicasting.

A number of different architectures and implementations have been proposed for multicast switching [26,27,28]. However, since we were interested in the design of very high-speed switches, we restricted out attention to input-queued architectures. There have been several input access schemes proposed for input-queued multicast switching, which can improve the performance of input-queued multicast switches to some degree. [31] introduces a scheme in which multicast and unicast packets of each input port were separately queued and all multicast queues took priority over unicast queues. A generalized model for the scheduling polices using FIFO queueing called TRTRA was presented in [32]. TRTRA is a greedy approach that tries to pack the HOL cells in a way similar to the block-packing game Tetris. The scheduling process is modeled using a 2-D tetris box. Each slot in the box can hold an output cell (which is a copy of multicast input cell). Column j of the tetris box holds cells destined to output port j. When a new cell advances to HOL, output cells are dropped onto the
columns in the tetris box corresponding to the destination set of the HOL cell. A departure date (DD) is assigned to the HOL cell specifying how long, from the current cell time, the input cell will be held at HOL before being discharged. The DD is equal to the largest row number among the output cells. The new output cell is dropped to the lowest possible level in the appropriate column, without getting ahead of another cell whose DD is less than, or equal to, its own. The bottom row of the tetris box specifies the transmission schedule for the current cell time. The scheduling algorithm is outlined below:

(i) At the end of time $t$, all output cells in the bottom row of the tetris box are discharged. Input cells with DDs=1 are completely served and removed from the input queue.

(ii) At the beginning of time $t+1$, all residue cells (the set of all output cells that lose contention and remains at the HOL of the input queues at the end of each cell time) drop down one level and their DDs are decremented by one.

(iii) New input cells advancing to HOL are then scheduled according to the order of their input port number.

In Chapter 3, we proposed a scheme in which the queue-splitting method was deployed to organize input queue and scheduling algorithm used random selection. Much higher throughput for both single-stage and multi-stage switches with unicast traffic can be achieved. We have stated that one of the advantages of our approach is it is applicable to both unicast and multicast traffic. Therefore, in this chapter, we apply our proposed scheme to multicast traffic and evaluate its performance in terms of throughput and latency. Our scheme outperforms any of the schemes described before.
4.2 Multicast Packet Switch with Queue-Splitting Scheme

The switch architecture we used was N inputs and N outputs and each input maintains several FIFO sub-queues for the arriving cells, no matter unicast or multicast (see Fig. 4.1). For multicast cell, we refer to the number of copies as fanout, which is defined to be the maximum number of outputs that can be connected to one particular input. The fanout of a multicast cell can be any number between 1 and N. Therefore, a unicast cell can be treated as a multicast cell with a fanout of 1. If the fanout equals N, it is one-to-all connection.

![Fig. 4.1 4x4 switch architecture with queue-splitting size 2](image)

Here we propose three policies to put the multicast cell into the input queues. Policy 1 is for the switch which has multicast capability, that is, each input port can transmit several cells simultaneously. When the cell arrivals, a sub-queue is randomly assigned to it and when the cell becomes HOL, all the copies generated. All of the copies are able to attend the contention of transmission. Multiple copies can be selected to transmit from same input port since the switch has the multicast capability. The copies selected by the scheduler transmit and others are still held in the HOL waiting for next round.
Policy 2 is for the switch which itself has no multicast capability. When a multicast cell arrives, all of the copies are generated and each is put into one of the sub-queues based on their destination. In other words, in this policy, each copy from a multicast cell is treated as an independent unicast cell.

Policy 3 is similar as Policy 1 except that the switch is a unicast switch, and in each time slot only one cell can be transmitted. When a cell arrives, it is randomly assigned into one of the sub-queues. When it is HOL cell, copies are generated one by one attending the contention for access to the switch fabric.

Regarding the above three polices, it can be seen that there are two general ways to service a multicast cell. One is replicating the cells over multiple cell times, generating one per cell time, as Policy 2 and 3. However, this approach has two disadvantages. First each input must be copied multiple times, and this increases the required memory bandwidth. Second, input cells contend for access to the switch multiple times, reducing the bandwidth available to other traffic at the same input. Higher throughput can be attained if, instead, it is assumed that one multicast cell can be copied to any number of cells in a single cell time for which there is no conflict. Therefore, Policy 1 and 2 should outperform Policy 3.

After describing the input queue organization schemes, we will introduce the scheduling algorithm. The multi-iteration random scheduling algorithm is deployed. There is a pointer per input port indicating which queue is started first in this time slot. In the first round of this time slot, the scheduler will check the HOL cell at this sub-queue first. If there are multiple requests destined to same output, one is selected randomly. This one will be transmitted and all the others are still held up in HOL position until they are chosen to access the switch fabric. If no cell is selected in this round, next non-empty queue will be checked in the second round, and so on. If the
switch has no multicast capability, the scheduler will stop checking once there is a cell at this port selected or if it finishes all of the iterations. If the switch is a multicast switch, the scheduler will keep on checking until all of the iterations are finished. Then in the next time slot, the pointer at each input port will be updated to the next sub-queues.

There are two different service disciplines that can be used. Following the description in [29], the first is no fanout-splitting in which all of the copies of a multicast cell must be sent during the same cell time. If any of the output cells loses contention, none of the output cells are transmitted and the cell must try again in the next cell time. The second discipline is fanout-splitting, in which copies of a multicast cell may be delivered to output ports over any number of cell times. Only those copied cells that are unsuccessful in one cell time continue to contend for output ports during the next cell time. Because fanout-splitting is work conserving, it enables a higher switch throughput [30] for little increase in implementation complexity. Therefore, we employ fanout-splitting discipline in our three policies, copies belonging to one multicast cell can be transmitted within any number of cell times.

4.3 Performance evaluation of multicast switch with queue-splitting scheme

The same simulation model as that in Chapter 3 is used. The simulation environment is:

1. The Poisson arrivals are assumed.

2. Traffic is evenly distributed.

3. The switch size is $32\times32$. 
The offered load $\rho_o$ is defined as the probability that an incoming cell arrives during a time slot. Since an incoming multicast cell can use multiple time slots to generate cells, the load of a inlet is higher than $\rho_o$. Here we define the load of input as the mean equivalent offered load $\rho_e$, which is:

$$\rho_e = (1-p) \times \rho_o + p \times f \times \rho_o \quad \text{(Eq. 3.1)}$$

where $p$ is the probability that an incoming cell is a multicast cell and $f$ is the mean fanout value of a multicast cell. For example, suppose $p$ equals 1, and mean fanout of the cells are 4, when the offered load $\rho_o$ is 0.2, the mean equivalent offered load is actually 0.8. All of the figures presented later showing the average latency versus offered load are based on the mean equivalent offered load.

The performance of the queue-splitting scheme is evaluated in terms of maximum throughput, delay. Our simulated switch is assumed to have infinite buffers at the input ports so that no cells are dropped due to a lack of buffer space.

### 4.3.1 Performance of Policy 1

This policy is for switches that have multicast capability. That is, multiple cells can be transmitted from the same input port provided their destinations are available at that time. Therefore, it has the best performance of the three policies.

- **Maximum throughput**

  In Table 4.1, the maximum throughput of 32×32 network for different mean fanout values of both uncorrelated and correlated traffic, are given in the next page.
<table>
<thead>
<tr>
<th># of queue-splitting size (# of iteration)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncorrelated traffic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fanout=4</td>
<td>80.1%</td>
<td>88.7%</td>
<td>94.0%</td>
<td>96.8%</td>
<td>98.5%</td>
</tr>
<tr>
<td>fanout=16</td>
<td>92.3%</td>
<td>95.9%</td>
<td>97.8%</td>
<td>99.0%</td>
<td>99.6%</td>
</tr>
<tr>
<td>Correlated traffic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fanout=4</td>
<td>70.1%</td>
<td>80.8%</td>
<td>88.4%</td>
<td>93.1%</td>
<td>96.4%</td>
</tr>
<tr>
<td>fanout=16</td>
<td>84.2%</td>
<td>90.2%</td>
<td>94.1%</td>
<td>97.0%</td>
<td>98.8%</td>
</tr>
</tbody>
</table>

Table 4.1 The maximum throughput for 32×32 network with multicast cell prob.=1.0,policy 1

Firstly, it can be seen that higher throughput can easily be achieved under such a policy since the switch has multicast capability. Note that the larger mean fanout value, the higher throughput we can achieve. This can easily be understood since larger fanout values can provide more chance of maximum matching. Our simulation verifies that correlation can degrade the performance. But with our proposed scheme under policy 1, we can greatly improve the throughput for correlated traffic since the queue-splitting scheme with iterations can significantly eliminate the correlation between the consecutive traffic.

- Average cell latency

From Fig. 4.2 and Fig. 4.3, it can be seen that under this policy, a higher throughput can be easily achieved with very low latency for both uncorrelated traffic and correlated traffic, although there is some degradation for correlated traffic.

In [32], there are several scheduling algorithms, such as TATRA introduced in earlier, WBA, Concentrate algorithm and Distribute algorithm, proposed for multicast input-queued switches. Among them, TATRA and Concentrate algorithms are said to have a better performance, especially compared to random scheduling (for simple input-queued switch without queue-splitting). Therefore, we will compare the performance of our queue-splitting and random scheduling scheme with them. Simulation results show that when the queues-splitting size and the number of...
iteration is equal or more than 4, we can achieve pretty better performance than any one of them, regardless of uncorrelation traffic or correlated traffic.

Fig. 4.2 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=4, uncorrelated traffic, policy 1

Fig. 4.3 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0, mean fanout=16, uncorrelated traffic, policy 1
Fig. 4.4 Average cell latency versus offered load for 32x32 network with multicast cell prob.=1.0, mean fanout=4, correlated traffic with mean pk-length=5, policy 1

Fig. 4.5 Average cell latency versus offered load for 32x32 network with multicast cell prob.=1.0, mean fanout=16, correlated traffic with mean pk-length=5, policy 1
4.3.2 Performance of Policy 2

- Maximum throughput

Under this policy, all the copies are generated simultaneously and then each of them is placed into one of the sub-queues. In this approach, each multicast cell is treated as a set of unicast cells. Therefore, a throughput for multicast traffic, similar to that for unicast traffic, can be obtained. The disadvantage of this method is that more buffers are required to store all of the copies.

<table>
<thead>
<tr>
<th># of queue-splitting size (1)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncorrelated traffic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fanout=4</td>
<td>59.8%</td>
<td>71.9%</td>
<td>81.9%</td>
<td>89.6%</td>
<td>94.9%</td>
</tr>
<tr>
<td>fanout=16</td>
<td>60.1%</td>
<td>72.4%</td>
<td>82.5%</td>
<td>90.2%</td>
<td>95.3%</td>
</tr>
<tr>
<td>Correlated traffic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fanout=4</td>
<td>57.7%</td>
<td>68.7%</td>
<td>78.9%</td>
<td>87.9%</td>
<td>94.2%</td>
</tr>
<tr>
<td>Fanout=16</td>
<td>60.0%</td>
<td>72.0%</td>
<td>81.6%</td>
<td>89.1%</td>
<td>94.6%</td>
</tr>
</tbody>
</table>

Table 4.2 Maximum throughput for 32×32 network with multicast cell prob.=1.0, policy 2

- Average cell latency

The conclusion can be drawn that the mean fanout value has no significant effect on the performance, as demonstrated from both the maximum throughputs shown in Table 4.2 and the delay performance in the following figures. There is slight difference between the performance of a small fanout and that of a large fanout.
Fig. 4.6 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0 and mean fanout=4, uncorrelated traffic, policy 2

Fig. 4.7 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0 and mean fanout=16, uncorrelated traffic, policy 2
Fig. 4.8 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0 and mean fanout=4, correlated traffic with mean pk-length=5, policy 2

Fig. 4.9 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0 and mean fanout=16, correlated traffic with mean pk-length=5, policy 2
4.3.3 Performance of Policy 3

Simulation results show that the performance of Policy 2 is almost indistinguishable from that of Policy 3, although the performance of Policy 2 is a little bit better than that of Policy 3.

Although the performance of Policy 2 and 3 are not as good as that of Policy 1, compared with the algorithms in [32], we can still achieve a similar, or better, performance when the queue-splitting size and the number of iteration is large. For example, for uncorrelated traffic, when the queue-splitting size and the number of iterations are equal to 8, results similar to the TATRA or Concentrate algorithm can be obtained. Once the number is larger than 8, we can outperform any algorithm in [32]. For correlated traffic, the results are even better. When the number of queue-splitting size and the number of iteration equals 4, our scheme already exceeds the performance of the algorithms in [32].

<table>
<thead>
<tr>
<th></th>
<th># of queue-splitting size (# of iteration)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncorrelated traffic</td>
<td>Fanout=4</td>
<td>59.8%</td>
<td>71.3%</td>
<td>81.3%</td>
<td>88.4%</td>
<td>93.4%</td>
</tr>
<tr>
<td></td>
<td>Fanout=16</td>
<td>60.2%</td>
<td>71.6%</td>
<td>81.3%</td>
<td>88.6%</td>
<td>93.4%</td>
</tr>
<tr>
<td>Correlated traffic</td>
<td>Fanout=4</td>
<td>58.0%</td>
<td>69.9%</td>
<td>79.7%</td>
<td>87.3%</td>
<td>92.7%</td>
</tr>
<tr>
<td></td>
<td>Fanout=16</td>
<td>60.2%</td>
<td>71.5%</td>
<td>81.1%</td>
<td>88.5%</td>
<td>93.4%</td>
</tr>
</tbody>
</table>

Table 4.3 Maximum throughput for 32×32 network with multicast cell prob.=1.0, policy 3
Fig. 4.9 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0 and mean fanout=4, uncorrelated traffic, policy 3

Fig. 4.10 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0 and mean fanout=16, uncorrelated traffic, policy 3
Fig. 4.11 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0 and mean fanout=4, correlated traffic with mean pk-length=5, policy 3

Fig. 4.12 Average cell latency versus offered load for 32×32 network with multicast cell prob.=1.0 and mean fanout=16, correlated traffic with mean pk-length=5, policy 3
4.4 Conclusion

We have studied the performance of our queue-splitting scheme with random scheduling under three different polices. The performance of Policy 1 for switches that have multicast capabilities is the best. High throughput and low latency can be easily achieved with this policy. One important point is that, compared with the TATRA or Concentrate algorithm in which simple FIFO input-queued with relatively complex scheduling is deployed, the performance of our scheme is superior when the queue-splitting size and number of iteration reaches some number.

Policy 2 and policy 3 are both for unicast switches. Performance improvement for multicast traffic can greatly obtained with the queue-splitting size and number of iteration increases. Although they are almost indistinguishable, Policy 2 has relatively better performance than Policy 3 with the pay of larger buffer occupancy.

After extensive performance study of our queue-splitting and random scheduling scheme, it can be concluded that performance can be dramatically improved for both uncorrelated and correlated traffic; this scheme can be applied to multi-stage switches to build up a large-scale switch size which still has a good performance; the most important and useful point is it is applicable to both unicast and multicast traffic. Therefore, our queue-splitting scheme with random scheduling is a good candidate for input-queues switch architecture.
Chapter 5 Conclusion and future work

The existing switching architectures can be classified into two general categories: electronic and photonic switching. We have done much work on both categories. Among the work done in these two categories, the multicast feature was our main concern.

$\log_2N$ network is the most suitable architecture for optical directional-coupler-based photonic switching systems. Since optical directional couplers can pass WDM signals, $\log_2N$ networks will play an important role in the future WDM optical network. In the first part of this thesis, we investigated a more general class of networks, called $\log_2(N,m,p)$ networks and presented a method to build up nonblocking multicast switches. The results obtained allow us to determine the best design tradeoffs among some important parameters, such as attenuation, crosstalk and the total amount of hardware, when used for directional-coupler-based photonic switching systems.

Secondly, we concentrated on the parts of the input buffer organization and the scheduling algorithms to improve the performance of input-queued switches. A discrete-event driven simulation model was developed for performance study and queue splitting and random scheduling with multiple-iteration scheme was investigated. We first applied the scheme to single-stage and multi-stage switches for unicast traffic. Results showed that performance, in terms of maximum throughput, average cell latency and cell loss, could be greatly improved as queue splitting and multiple-iteration is executed. Furthermore, correlation could be eliminated as the number of queue-splitting size increases. Following that, we performed the same method to multicast traffic. Three different policies have been proposed, one of which was for switch that has multicast capability, two were for unicast switches. Great
performance improvement was also achieved for multicast switches, regardless of whether the traffic was with or without correlation. Compared with some existing multicast scheduling algorithms such as TATRA, it was demonstrated that our scheme could achieve much better performance when the queue-splitting size and the number of iterations reached a number. Therefore, it could be concluded that our scheme is a good candidate for input-queued switches, since it is applicable for both unicast and multicast switches, better performance can be achieved for both uncorrelated traffic and traffic with correlation.

What may be the future work is to build up an analytical model. Extensive analytical studies on pure input-queued switches for single-stage and multi-stage have been done. Regarding the scheme in which multiple sub-queues are maintained at each input port and multiple-iteration of random selection is deployed, it will be quite helpful if some analytical model could be built up and theoretical results obtained.
REFERENCES


