Providing Quality of Service over High Speed Electronic and Optical Switches

by

Can Emre Koksal

B.S., Electrical Engineering
Middle East Technical University (1996)

M.S., Electrical Engineering and Computer Science
Massachusetts Institute of Technology (1998)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

Author .....
Department of Electrical Engineering and Computer Science
September 3, 2002

Certified by .....
Robert G. Gallager
Professor of Electrical Engineering
Thesis Supervisor

Certified by .....
Charles E. Rohrs
Research Scientist, Lab. for Information and Decision Systems
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Providing Quality of Service over High Speed Electronic and Optical Switches

by

Can Emre Koksal

Submitted to the Department of Electrical Engineering and Computer Science
on September 3, 2002, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In a network, multiple links are *interconnected* by means of *switches*. A switch is a device with multiple input and output links, and its job is to move data from the input links to the output links. In this thesis, we focus on a number of fundamental issues concerning the quality of service provided by electronic and optical switches. We discuss various mechanisms that enable the support of quality of service requirements. In particular, we explore fundamental limitations of current high speed packet switches and develop new techniques and architectures that make possible the provision of certain service guarantees. We then study optical wavelength switches and illustrate how similar ideas can be applied in a manner consistent with the current state of optical switching technology.

First, we focus on providing rate guarantees over packet switches. We develop a method called *rate quantization* which converts the set of desired rates into a certain discrete set such that the quality of service guarantees can be greatly improved with a small *resource speedup*. Moreover, quantization simplifies rate provisioning for dynamically changing traffic demands since it allows service opportunities for different input output link pairs to be scheduled with minimal dependence. We illustrate an isomorphism between packet switch schedulers and Clos networks to develop such schedulers.

Next, we evaluate the amount of resource speedup necessary for single stage switches to support multicast rates. This speedup limits the scalability of a single stage multicast switch a great deal. We present an in depth study of multistage switches and propose a number of architectures, along with associated routing and scheduling algorithms. We illustrate how the presence of multiple paths between input output pairs can be exploited to improve the performance of a switch and simplify the scheduling algorithms. Some of our architectures are capable of providing multicast rate guarantees without a need for a resource speedup.

We extend our results on switch schedulers and use them for providing service guarantees over optical wavelength switches. We will take the limitations of the optical crossconnects and unavailability of optical memory technology into account, and modify the procedure we developed for electronic switches to make them suitable for various optical wavelength switches. These results will provide understanding of when to move optical switching closer to the end users for an efficient utilization of resources in networks with both optical and electronic technologies.

Thesis Supervisor: Robert G. Gallager
Title: Professor of Electrical Engineering

Thesis Supervisor: Charles E. Rohrs
Title: Research Scientist, Lab. for Information and Decision Systems

Acknowledgments

I wish to thank my thesis advisors Prof. Robert Gallager and Dr. Charles Rohrs for their guidance and support throughout my graduate education at MIT.

Professor Gallager taught me how to do engineering research and provided me with the opportunity to pursue my own ideas. His fundamental and creative thinking, and clarity of communicating his ideas were truly inspiring. He always finds a simple way to look at any given problem which seems almost magical to me. I enjoyed every moment of our discussions. His perfectionism gave me the motivation to challenge myself more than I would have otherwise. I feel very lucky to have been a student of his.

I have been inspired by the fundamental understanding that Dr. Rohrs posesses on all conceptual and practical problems in the area of networking. His insights influenced me every stage of this thesis. I also enjoyed our non-research discussions and I would like to thank him for his friendship outside the school, as well as inside.

I would like to thank Professor Hari Balakrishnan for his enthusiasm and showing a genuine interest in my work. Our discussions, both on my thesis and on other areas were very fruitful to me. I am very thankful to Professor Eytan Modiano for showing an interest in my work and being in my thesis committee.

I am grateful to Professor Vincent Chan and Dr. Rick Barry for their advice, encouragements and support. They both were major figures during my studies.

Many other people have contributed to this thesis in various ways. Among them, I would like to thank Professor Vahid Tarokh and the members of Networks and Mobile Systems Group at the Laboratory for Computer Science.

The social and intellectual atmosphere of the Laboratory for Information and Decision Systems was ideal for me. Many thanks to my friends, Ibrahim Abou-Faycal, Erik Anderson, Randy Berry, Aaron Cohen, Diana Dabby, Anand Ganti, Angelia Geary, Shane Haas, Shan-Yuan Ho, Chung-Yao Kao, George Kotsalis, Julius Kusuma, Mike Neely, Alex Wang, Edmund Yeh, Won Yoon and Murtaza Zafer. But especially I would like to thank Thierry Klein, Hisham Kassab and Masha Ishutkina for their invaluable fellowship. In addition I wish to thank Doris Inslee, Kathleen O'Sullivan and Marilyn Pierce for making my life easier at MIT by taking care of the administrative issues that came across my path.

I would like to thank Asu for her warmth, support and encouragement. I am also thankful

to my best friends in Boston, Oguz, Erdem, Alp, Cagri Etemoglu, Cagri Savran, Onur, Volkan, Goksel, Tarkan, Murat and my cousin Baris with whom I shared a lot of fun.

Most importantly, I want to express my gratitude to my parents for their unconditional love and always being there for me. They always believed in me and supported me with devotion to pursue my dreams. This thesis is dedicated to them.

This research was partially sponsored by Lockheed Sanders under Grant QK9932, by DARPA under Grants BX-7036 and BX-7276 and by NSF under Grant NCR-9702015.

Contents

Chapter 1

Introduction

While the Internet has served as a vehicle for research and education for more than a decade, recent years have witnessed its tremendous growth and great potential to provide a wide variety of services. By any measure, this growth is remarkable on all fronts: the number of hosts, the number of users, the amount of traffic, the number of links, or the growth rates of Internet Service Provider (ISP) networks. It is predicted that Internet traffic will continue to grow at high rates, although probably not as high as predicted one or two years ago. There had been some overestimation for the need for high rate applications, and this induced a sudden and unorganized growth. Hence, we may not experience further growth in the short term due to the slowdown in the economy caused by this overexpansion, but we expect recovery in the long run. Regardless of the condition of financial markets ([1]), the Internet is widely considered as the most reachable platform for the next-generation information infrastructure.

One challenge to the success of the Internet lies in the deployment of very high speed packet switches (IP, ATM, Gigabit Ethernet or Frame Relay) to meet the growth of multimedia and data traffic. Coupled with these high rates, there is a great demand for the Internet to provide quality of service (QoS) guarantees for a wide variety of applications that will be part of our lives in this century. Hence, there is an urgent need for the design of scalable and high speed switches/routers that can provide QoS guarantees, e.g., bounded delay.

Similarly, networks that multiplex thousands of sessions and provide transport to them at the backbone transport networks have steadily grown more complex as a consequence of more sophisticated customer needs, the rate of traffic growth and the conditions imposed by external markets. Traditionally, transport has been viewed just as a data carrier. Tied with the emergence

of QoS traffic, there is a need for availability and automated provisioning within the backbone network.

In most current networks, the dominant technology in the backbone network is optical technology. Even though the optical layer has the potential to provide very high bandwidths, it is not as agile as the electronic layer. That is, resource assignments/reservations must be made for longer durations of time and larger chunks of data, and it is not possible to make updates as frequently as in electronic networks. Given this limitation, an important challenge is to have the optical network provide the services desired by the users sharing the resources, and at the same time harness the bandwidth in an efficient manner. How quickly these services can be provided and how efficiently those services can be supported will be key differentiators for future optical networks. This challenge is not just technological. It also involves consideration of network architecture and algorithm design issues.

Also, with increasing rates, bringing optical transport services closer to the high end customer appears to be economically attractive; as optical transport gets closer to the customer, the value of providing optical services that better match the customer application increases.

In this thesis, we focus on a number of fundamental issues concerning the quality of service provided by electronic and optical switches. We discuss various mechanisms that enable the support of quality of service requirements: In particular, we explore fundamental limitations of current high speed packet switches and we develop new techniques and architectures that make possible the provision of certain service guarantees. We, then study optical wavelength switches and illustrate how similar ideas can be applied in a manner consistent with the current state of optical switching technology.

The rest of this chapter is organized as follows. First, we give an overview of the architecture of current networks. Next, we present an introduction to electronic and optical switching technology. In the sections following that, we discuss the issues to be treated later in the thesis. We wrap up with a summary of this thesis.

1.1 Network Architecture

The architecture of a typical wide area network is as illustrated in Fig. 1-1. There are multiple levels of aggregation and switching. The end users lie at the periphery, where the first level of traffic aggregation and routing is performed. That is, the traffic generated by end users is collected,

combined and transferred over the links toward its destination. Closer to the backbone, speeds and the level of aggregation grow. The second level of aggregation and transport of massive amounts of data is performed at the backbone, where most of the switching is in optical domain. Thus, the optical network can be considered as a functional layer providing certain services to the electronic network, which lies above it hierarchically.

Actually, this illustration of the network architecture is rather simplistic. In general, there is no sharp transition between these layers and technologies. Almost all of the optical switches have some electronic processing. Namely, some portion of the optical signal is dropped by means of *add drop multiplexers*, processed electronically, and added back to optical fibers. Also, most of the routers have some optical processing. Therefore, in practice it is very hard to make the separation between an optical and an electronic switch. There are two reasons why we make this distinction. First, we will treat electronic switching and optical switching separately. Second, the optical switches we consider will be *all-optical*, i.e., no opto-electronic conversion and electronic processing.

The first level of aggregation is done at the edge. Data and real-time traffic generated by the end users injected into the core is very bursty and hence its dynamics vary in short time scales (e.g., few milliseconds). For a more detailed treatment of the characterization of multiscale multimedia and data traffic, see [42] and [43] respectively.

A router is of critical importance since it forms a bridge between the end users and the optical network. It provides quality of service that end user applications desire, and it multiplexes incoming traffic into optical channels to be transferred to the optical network. We assume electronic switching at the router level, but the rates go up to optical channel rates (i.e., transmission may be optical) and the switch sizes need to be large. There are many technical issues to be considered in such high speed routers. We will discuss the technological limitations of these routers and study different ways of providing service guarantees.

Transfer between routers are made either over a direct link, if one exists, or through optical switches. Optical switches can handle multiple optical channels simultaneously by Wavelength Division Multiplexing (WDM) technology. The principle of WDM is as follows. Since the rates at the router level go as high as optical channels, the signal at each output link of a router can be associated with a specific optical wavelength. Then these links are multiplexed on to the same fiber, and 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

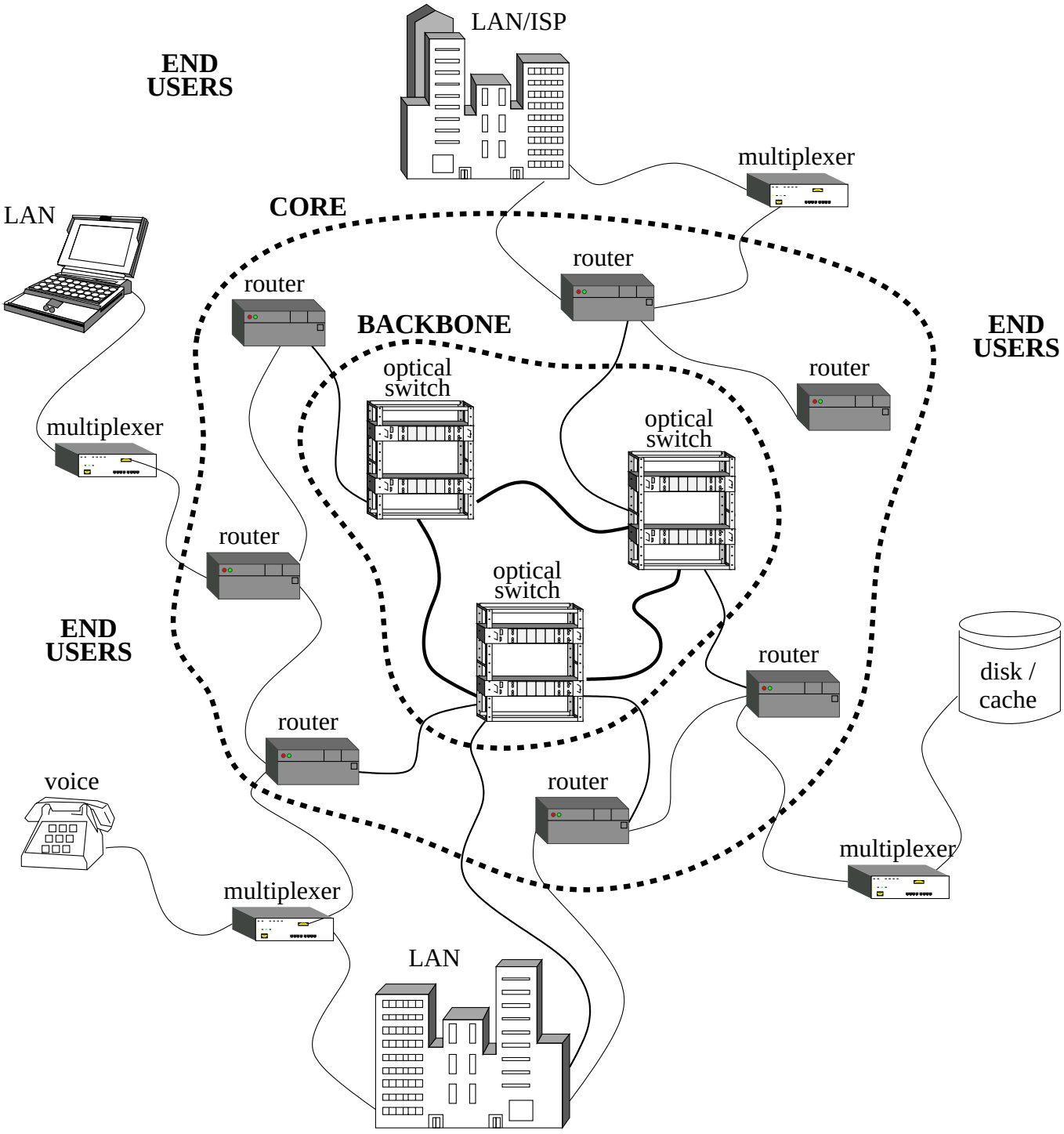

Figure 1-1: Structure of the entire network is illustrated. At the high order backbone lie the optical switches; the traffic generated by different applications are multiplexed and transferred to their ultimate destinations through a network of routers and optical switches in the core.

the realization of all-optical routers, which can handle many WDM channels simultaneously without the need for very high-speed electronics.

Having an all-optical WDM layer will enable a huge amount of "through" traffic to be switched in the optical domain. As a result, this layer can not only eliminate electronic bottlenecks, but also create high speed communication pipes that are transparent to bit rate and signal format. Namely, end to end optical paths with no electronic conversion are set up between sources and destinations (e.g., routers) via optical wavelength switches. Such paths are also known as lightpaths. As viewed by routers, lightpaths act as dedicated links that connect them with other routers. They want these lightpaths to be set up, and resource assignments to be updated in an on-demand basis as the traffic requirements of the end users change. Therefore, within the backbone there is a need for automated provisioning, a feature which current optical switches lack. The structure of optical switches and how rate guarantees can be provided by them will be studied later in this thesis.

1.2 Electronic Packet Switching

In this section, we present an introduction to packet switching. We give a basic model for packet switches, and then we briefly discuss what we will be studying in this thesis. We also provide a literature survey.

1.2.1 Basics

Consider a link with a constant transmission capacity that is shared by a number of users to transmit data. In old telephone networks, data at each input link was broken into frames of equal size. If transmission capacity is shared according to a prespecified format which repeats in each frame, the system is called Time Division Multiplexed (TDM) and the (fixed) capacity assigned to a user is called a circuit.

In a network, multiple links are *interconnected* by means of *switches*. A switch is a device with a number of input and output links, and its job is to move data from the input links to the output links. A set of connections between input-output pairs of a switch at a specific point in time is called a *configuration*. In all the switches we will consider, input and output links have identical capacity. If the link capacities are shared in a TDM manner, the switch is called a *circuit switch*. The position of a circuit in a frame determines the output link to which this circuit is transferred; hence, no extra overhead containing routing information is necessary. Note that TDM is not the

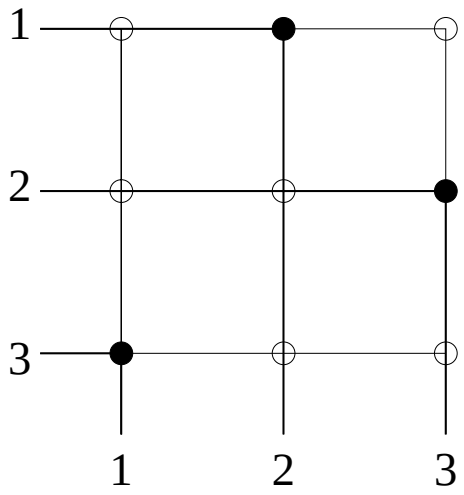

Figure 1-2: A 3×3 crossbar fabric. Crosspoints are set to connect the lines to enable end to end connections. For instance, input 1 and output 2 are connected through the corresponding crosspoint.

only alternative for circuit switching. For instance, in a wavelength switch, the output port and the output wavelength to which an incoming signal is routed at any given time is determined solely based on the input wavelength and input port carrying the signal. Accordingly, wavelength routing can be considered as a form of circuit switching.

Circuit switching is ideal for voice traffic since voice requires constant rate and a typical connection lasts for a long time. On the other hand, since the circuits are held fixed for long time periods, circuit switching may not be appropriate for data and multimedia applications due to their bursty nature. The long lasting rate assignment of circuit switching can be avoided by labeling each segment of data with routing information using a header. These segments are known as *packets*. Since each packet has routing information included, there is no need for a prior circuit assignment. Since resources are not necessarily reserved beforehand as in circuit switching, there is the necessity for queueing to avoid loss of data. Interconnection of packet multiplexed links is done by *packet switches*.

The core of a packet switch is composed of a switch fabric and memory elements. The function of the fabric is to set up connections between the input and the output links. A very important class of fabrics is the non-blocking class. A fabric is non-blocking if a connection can always be set up between any free input link and any free output link. A link is free if it is not a part of any existing connection. The most popular non-blocking fabric is the crossbar. A crossbar fabric can be thought of as a set of lines, and crosspoints that connect these lines. In general, connections

between inputs and outputs are made synchronously in a crossbar as illustrated in Fig. 1-2. Hence, packets are fragmented at the input of crossbar switches into equal size cells that are transferred to the output side synchronously. The most important limitation of a crossbar is the so called "crossbar constraint:" At any point in time, only one input can be matched with a given output, and only one output can be matched with a given input[1]. For example, in Fig. 1-2, the first input is connected to the second output and hence no other connection can be made from the first input or to the second output. Note that the condition for non-blocking is more general in that it allows more than one input to be matched with an output and more than one output to match with an input.

In a packet switch, packets that are destined for the same output link may arrive simultaneously at various input links. Some switches (e.g., crossbar switches) may only be able to immediately transfer one of these contending packets to the destined output link; the others must be enqueued for later transmissions. This form of congestion is unavoidable in a packet switch and dealing with it often represents the greatest source of complexity in the switch architecture.

There are different queueing schemes which provide ways to buffer the incoming packets. Depending on the physical location of the queues, switches can be classified as output queued (OQ), input queued (IQ) or combined input and output queued (CIOQ). Today, most of the deployed commercial switches and routers employ output queueing, i.e., all of the queueing is done at the output side of the fabric and no storage is present at the input. Thus, every packet must be placed directly into its desired output queue upon arrival. Since packets destined for the same output link may arrive simultaneously from many input ports, each output buffer needs to enqueue traffic at a higher rate than the line rate. In particular, the rate required inside the fabric is proportional to both the number of ports and the line rate. Output buffering has long been considered an ideal way of constructing packet switched devices because of its theoretical performance: Output queued switches can send out packets in any order after they arrive at the inputs, so that OQ switch can provide the QoS of a multiplexer. Hence, an ideal OQ switch equipped with infinite buffer space can be *work conserving* under any traffic pattern (arrival process). For a switch, work conserving means that, at any point in time, if there exists a packet destined to an output link, then that output link is busy transmitting some packet at that time. However, with ever increasing link rates, it is simply no longer possible to find random access memories (RAMs) with sufficiently fast access

[1]Note that this constraint assumes that a crossbar does not have broadcast capability. We will also deal with crossbars with broadcast capability later in this thesis.

times to build OQ switches. Hence, an OQ switch gets more and more unfeasible as the switch sizes increase, i.e., it is not *scalable*.

Input queueing does not have the scaling limitations of output queueing. In the input queueing architecture, the fabric can run at a single line rate with one read and one write operation per incoming packet. However, an input buffered architecture also presents some technical difficulties due to the limitations of the fabrics, such as the crossbar constraint. Input buffered architectures are employed in many high speed applications, e.g., [5]-[9].

Combined Input and Output Queueing is a good compromise between the performance and scalability of both output and input queued switches. For input queued switches, at most one packet need be delivered to an output port in one unit of time, and for an output queued switch, up to N packets need to be delivered to an output in the same amount of time. Using CIOQ, instead of choosing one of these two extreme choices, we can choose a reasonable value in between 1 and N. This can be achieved by having buffers at both the input and output ports.

1.2.2 Past Work and Our Contribution

In this section, we discuss the following topics: Providing service guarantees over single crossbar packet switches, isomorphism between crossbar switch schedulers and Clos networks, multicast support, and multistage switches. For each of these sections we first give an introduction to the topic, then the previous literature, and then an outline of our contributions.

Providing Service Guarantees over Single Crossbar Packet Switches

In high speed switches and routers, to achieve high performance, an incoming packet is usually fragmented into equal size cells. We define a *time slot* as the time it takes for one cell to be transmitted over an input or an output link (it is also referred to as the *cell slot* in the literature), i.e., each input receives at most one cell and each output transmits at most one cell in a time slot. These cells are dispatched to the outputs in a manner synchronous across input lines. Let us define *service slot* as the time it takes one cell to be transferred from an input to an output of the crossbar. We view the time over which a crossbar switches from one configuration to another as included in the service slot. Note that a service slot is not necessarily identical to a time slot. For instance, the crossbar may transfer S, $S \geqslant 1$ cells per time slot. The factor, S, represents the

amount of resource speedup where S is not constrained to be an integer. Hence,

$$\text{service slot} = \frac{\text{time slot}}{\text{speedup}} \tag{1.1}$$

Since more than one cell can be forwarded to the same output port, switches with speedup must have buffers at the output. If the crossbar operates with a speedup of S, all the memory elements must be able to operate S times as fast as the links. Note that all of the results we present will be for a switch of size $N \times N$ but we believe that generalization to an asymmetric ($N \times M, N \neq M$) switch is straightforward.

With the above definitions, the crossbar constraint translates into the following: In a service slot, each output can receive data from only one input and each input can send data to only one output. This constraint makes it very difficult to provide rate guarantees. Indeed, Karol et.al. [10] showed that when there is a single FIFO queue at each input, the throughput[2] of a switch for large N is limited to 58.6% for uniform random arrivals[3] of input traffic. This degradation of throughput is due to head-of-the-line (HOL) blocking. HOL blocking is caused by the situation in which multiple inputs each containing a cell at the head of its queue are destined to the same output. Since only one of them can use the crossbar in a time slot, all the others have to wait even if they have cells in their queues that are destined to idle outputs.

One brute force approach to battle HOL blocking is to introduce some speedup. There have been a number of studies (see e.g., [11]) that show that the performance of a crossbar switch can be improved remarkably with some constant speedup. However, all these studies assume a certain (simplistic) statistic for the arrival processes in order to deal with average delay and throughput guarantees. Thus, they lack generality.

Another way of eliminating HOL blocking is to change the queueing structure at the input. Instead of keeping all the packets in a FIFO queue, a separate queue can be maintained for each source-destination pair as illustrated in Fig. 1-3. This eliminates HOL blocking. This scheme is known as virtual output queueing (VOQ). This queueing scheme overcomes the HOL blocking associated with FIFO input queueing while keeping its scalability advantage.

One key factor in achieving high performance using VOQ switches is the scheduling algorithm

[2]Throughput is defined to be the average fraction of time that any given output is busy sending packets down the output line.

[3]Arrivals at each input are independent identically distributed (IID) and the destination of each packet is uniformly distributed over all outputs.

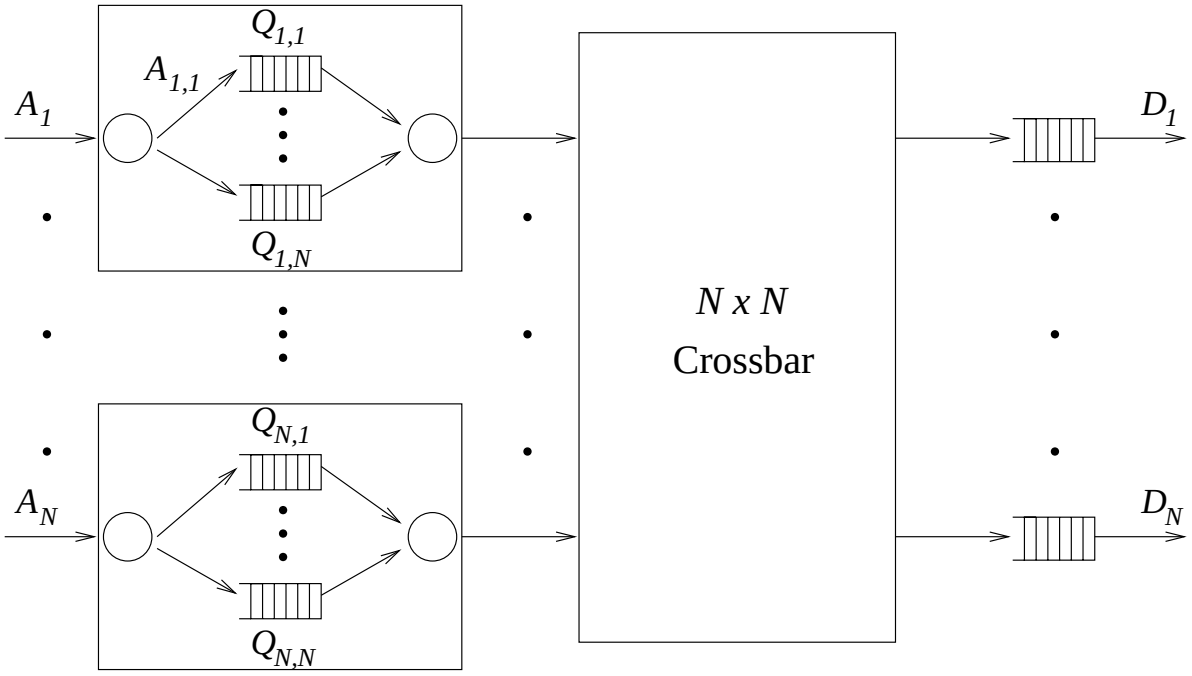

Figure 1-3: The switch keeps a per-output queue at each of its inputs. Service opportunities are provided to a packet at the head of any one of the per-output queues.

that is responsible for the selection of packets to be transmitted from the input links to the output links in each service slot. The VOQ switch architecture is receiving much attention from the research community, and many commercial and experimental switches based on this technique have already been built.

There have been two fundamentally different approaches to this scheduling problem depending on the level at which the scheduling is done: cell scheduling and rate reservation based scheduling. Before we study these approaches, we believe that it is important to discuss the two major alternatives for the *service agreement* between users and a switch: connection oriented or connectionless, since each one of these approaches to the scheduling problem is suitable for only one type of service agreement. Note that we are dealing with switches that handle multiple (e.g., thousands at the router level) end to end sessions between each input output pair. We study service guarantees between input output pairs in a switch, and we call these agreements between the switch and a group of users that share the same input output pair, a *contract*. Basically, a contract is defined as a set of unicast rates as given in the following two paragraphs.

A connection oriented contract has a duration (in number of time slots) associated with it. Also, the number of cells to be transferred within the lifetime of the contract is specified. If a

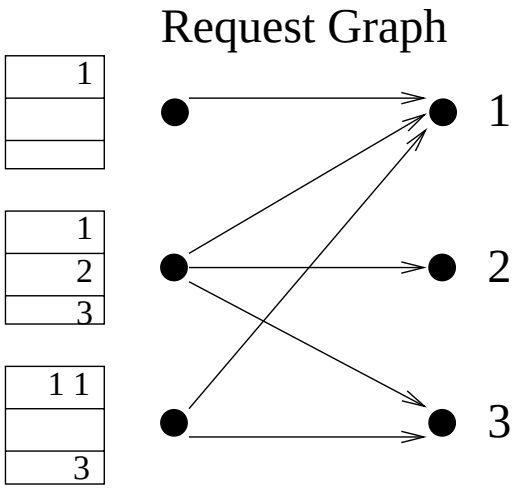

Figure 1-4: A typical request graph is illustrated. Edges are assigned priorities (possibly identical) that are functions of the state of the system. Then a connect graph is found according to these priorities subject to the crossbar constraint.

switch is dealing exclusively with connection oriented contracts, a rate, R_{ij}, can be associated with the input-output pair (i, j) as the ratio of the number of cells to be transferred between the pair to the lifetime of the contract. Hence, R_{ij} takes on values from the set $[0, 1]$. Note that it is only with connection oriented service that rate reservation based scheduling algorithms can be developed since, as shall be discussed, such algorithms require the prior knowledge of the desired rates between I-O pairs. In connection oriented contracts, an admission controller makes sure that no input or output link is oversubscribed, i.e., $\sum_i R_{ij} \leqslant 1$ for all j and $\sum_j R_{ij} \leqslant 1$ for all i. Succinctly, we can represent each R_{ij} as the (i, j) entry of a matrix, R. The two admission control inequalities imply that R is a *doubly sub-stochastic* matrix.

In connectionless service, no rates are specified by the users. Congestion avoidance and control mechanisms are necessary to prevent buffers inside the network from overflowing. As shall be apparent, cell scheduling is an appropriate alternative for connectionless service.

In the approaches based on cell scheduling ([5]-[8], [18]-[22]), the problem of finding the appropriate connections between the inputs and the outputs of a crossbar is posed as a matching problem in a bipartite graph. Every service slot, a request graph is generated as illustrated in Fig. 1-4. A request graph is composed of the edges, one for each non-empty VOQ. A scheduling algorithm chooses which edges shall be used for transmission of cells in the service slot. Edges are assigned priorities (possibly identical) that are functions of the state of the system. They are updated every time slot and the new matching which maximizes some objective function is found subject to the

crossbar constraint. The objective may be achieving a stable marriage match ([5], [18]), achieving a maximal match ([19], [22]) or maximizing the sum of edge weights which are in the connect graph ([6], [7]). The edge weights (for a VOQ) are usually chosen to be the queue size ([5]-[7]), the delay experienced by the packet at the head of the queue ([5], [7]) or even identical edge weights.

Cell scheduling algorithms tend to be aggressive in the sense that they look for some useful matching at each slot. Therefore, they adapt to some extent to dynamically varying traffic patterns. They rely on congestion avoidance and control to avoid buffer overflow, and most of them depend on the use of traffic shapers for fairness[4]. Other than for admission control purposes, they do not require any apriori information about the arrival processes. It was shown ([7]) that 100% throughput can be achieved with VOQ switches for all possible cell arrival processes in which all the input links are fully utilized, and no output link is oversubscribed. It has also been shown that when a VOQ crossbar has a speedup of 2, certain QoS guarantees can be achieved ([18], [19], [5]).

There are two different types of delay at the input of a packet switch. The first is due to the randomness in the packet arrival process. This kind of delay is unavoidable and queue sizes depend on how bursty the arrival processes are. The second is due to the imperfections of the schedulers used in the switches. For instance, a switch that provides connection oriented service is responsible for providing the desired number of service opportunities specified in the contract between each I-O pair. If the lifetime of the contract is T, then the switch will go through ST configurations and at least TR_{ij} of them should connect input i to output j for the terms of the contract to be met. We call such a schedule of configurations, a *switch schedule*. Switch schedules for connection oriented services are generated by rate reservation (RR) based scheduling algorithms. Suppose, for some switch schedule, the switch goes through configurations such that input i and output j are connected for $D_{ij}(t)$ time slots, by time $t \leqslant T$. We call the difference, $tR_{ij} - D_{ij}(t)$, the service lag for I-O pair (i, j) at time t.

Note that service lag is directly tied with delay. Indeed, for an I-O pair

$$\text{maximum cell delay} = \frac{\text{maximum service lag}}{\text{rate}} \tag{1.2}$$

This can be shown as follows. Suppose the maximum service lag for I-O pair (i, j) is L, i.e., $tR_{ij} - D_{ij}(t) \leqslant L$ for all t. If there is a rate controller at the input of the switch, then $A_{ij}(t) \leqslant tR_{ij}$ for all t where $A_{ij}(t)$ is the number of (i, j) cell arrivals. This implies that $A_{ij}(t) - D_{ij}(t) \leqslant L$

[4] In almost all of cell scheduling algorithms, bursty flows hurt non-bursty flows without traffic shaping.

for all t. Hence, the number of (i,j) cells in the switch does not exceed L. Therefore, the delay experienced by a cell cannot exceed L/R_{ij}; otherwise more than L (i,j) cells accumulate in the switch.

Rate reservation based algorithms were originally proposed for circuit switches in traditional voice networks to provide constant bit rate (CBR) guarantees for voice traffic that is rather static in nature. In that case, rate is reserved for very long durations. This kind of switching is also known as *multirate* circuit switching if desired rates between input output pairs (I-O pairs) are picked from a set of possible rates (e.g., certain fractions of link capacity rather than $\{0,1\}$ as in full circuit switching).

Several statistics and forecasts have already been reported in the literature comparing the evolution of data traffic to that of traditional telephone traffic. As usual, such predictions can fail in the details; however, it is clear that data traffic will play a dominant role in the future of telecommunications. Thus, the challenge here is that to provide service guarantees not only over the long term but also for the much shorter time scales of more dynamic and bursty data and multimedia traffic since the service requirements change fairly quickly with such sources.

Another reason why it may be desirable to keep a contract short is that the quality of service is better with shorter contracts. Let us clarify this statement. Consider a contract with infinite duration. For a switch to meet the terms of the contract, it is sufficient that the service lag remains bounded: If $tR_{ij} - D_{ij}(t)$ is bounded over all t and all (i,j), then

$$\lim_{t\to\infty} \frac{D_{ij}(t)}{t} = R_{ij}$$

and the rates desired between all the pairs are met successfully even if $tR_{ij} - D_{ij}(t)$ may be very large. For instance, the switch may provide no service opportunities to some I-O pair for a long time, and provide a bunch of opportunities in a burst and still be able to meet the rate requirement for that pair. On the other hand, for contracts with short durations, it is not sufficient that the service lag be bounded for all I-O pairs. It is also necessary that the bound be small for satisfactory service quality. The switch cannot wait for a long time to start providing service opportunities since the contract duration is short. The opportunities need to be provided smoothly (i.e., small service lag) for satisfactory quality of service with short contracts. For this reason, even for voice traffic, for which the QoS requirements do not change rapidly, shorter contracts are desirable since voice applications are delay sensitive.

A number of RR based scheduling algorithms have been proposed to provide guaranteed performance over shorter time scales. An early approach which is called the parallel iterative matching (PIM) algorithm was presented in [12]. The algorithm is complicated, and limited in its application. Later, weighted probabilistic iterative matching (WPIM) was proposed in [13]. This is simpler than PIM and allows flexible allocation of rate among different links. However, both PIM and WPIM can provide probabilistic guarantees only. Indeed, the service lag is not necessarily upper bounded with these algorithms.

The BATCH-TSA algorithm proposed in [14] is a RR based scheduling algorithm that guarantees bounded service lag. The algorithm treats the switch as a time division multiple access (TDMA) network and the problem of providing rate guarantees is translated into the time slot assignment problem. Each flow is first stored in a queue for a period, say T time slots. If at most T packets arrive at each input and at most T packets are destined to each output in the first time period, it was shown that all these packets can be transmitted in the next period. The idling weighted round robin (WRR) algorithm [14] is fundamentally the same as BATCH-TSA but the method in which the packets are scheduled within a frame is different. In both of these approaches, the frame size is initially chosen by the algorithm. A large frame size implies a large service lag, while a small frame size implies the set of rates for which the switch can provide bounded service lag is very limited. As a result, these algorithms fail to provide uniform service guarantees for all non-uniform traffic.

A more general approach to RR based scheduling is the Birkhoff-von Neumann decomposition ([15], [16]). We will consider some of the details of this approach later in this thesis. This approach eliminates the problem of choosing the frame size and provides uniform service guarantees for all admissible traffic. However, bounds on the service lag can be very high, and as we shall show, these bounds are tight. Therefore, a higher (possibly much higher) rate than the long term average rate of a bursty, delay sensitive traffic stream must be allocated in order to satisfy its delay requirement. As a result, for variable bit rate data and multimedia sources, the Birkhoff-von Neumann approach may not be satisfactory.

In Chapter 2, we study the fundamental properties of reservation based scheduling. We present some examples to show that the bounds given in [15] for the service lag are indeed tight. Then we discuss how to use speedup to improve the quality of rate guarantees for them to be suitable for variable bit rate sources. We introduce *rate quantization*, and propose an algorithm for efficient rate

quantization. Basically, rate quantization converts the set of desired rates into a certain discrete set which can then be used as an input to a RR based scheduling algorithm. We will show that for an $N \times N$ switch, along with some (small) speedup, rate quantization improves the service lag by a factor of order O(N) even with simple schedulers.

Rate quantization also guarantees the presence of certain points in time where the service lag is non-positive for all I-O pairs simultaneously. Such events occur frequently enough to enable synchronous short term service provisioning (rate updates occur simultaneously), which is of critical importance for multimedia and other data sources. We also show that, rate quantization makes it possible to update rates asynchronously without rerunning the entire decomposition algorithm. Without rate quantization, the scheduler must generate the entire schedule before the switch can configure the first connection.

We will also show that the complexity of the scheduler is significantly reduced by rate quantization. Moreover, our scheduler and the switch can operate simultaneously; thus the complexity is spread over a long time period, rather than being incurred as a one time cost.

As an insight into rate quantization, consider the following simple example. Suppose a link is shared by three users, U_1, U_2 and U_3, each of which has packets to be transmitted. Let every packet have identical size, and the link capacity be 1 packet/second. Users also specify weights, $w_i,\ i \leqslant 3$, which signify the fraction of the link capacity needed for user i. Consider the case where $w_1 = 0.53,\ w_2 = 0.17,\ w_3 = 0.3$. The link can transmit only one packet a second, starting at time $t = 0$. Let $D_i(t)$ be the number of (full) packets served for user i by time t. Since

$$\sum_{i=1}^{3} D_i(t) = \lfloor t \rfloor$$

regardless of the scheduler used, there will always be some user, i, for which $D_i(t) < w_i t$ for all $t < 100$. That is, there exists a user for which the service lag $(w_i t - D_i(t))$ is positive for all $t < 100$.

Now, suppose we expand the link capacity so that 11 packets can be transmitted every 10 seconds. If in the first 10 seconds, U_1, U_2 and U_3 are given 6, 2 and 3 service opportunities respectively, then at the end of 10th second, $D_i(10) \geqslant 10w_i$, for all i. We can repeat the schedule to have this property once every 10 seconds. In the first scenario, if the users made a contract for some period of time less than 100 seconds, the link would not be able to meet the terms of at least one contract. With rate expansion, this period is cut down to 10 seconds. This enables the users to update the terms (e.g., the rate) of their contracts more frequently, and thus a larger set

of sources (e.g., bursty sources whose statistics vary in shorter time scales) can be accommodated. In the case of a crossbar switch, we will show how to divide some given extra capacity (speedup) among the I-O pairs to achieve a similar improvement.

In the switches we consider, we assume that there are rate controllers at the input of the switch for traffic policing (the rate controllers can be present either explicitly as a separate unit, just like admission controllers, or the schedulers may inherently function as rate controllers without the need for separate units). This confines the variations of the traffic at the input queues of the switch so that the impact of these variations does not propagate to the output. In that sense these rate controllers act as a traffic shaper in a switching network.

Isomorphism between Crossbar Switch Schedulers and Clos Networks

As described in the previous section, chapter 2 discusses service guarantees for connection based service contracts. There a contract is defined as a set of unicast rates and a duration that represents the lifetime of the contract. Once a contract matures, i.e., at the end of its lifetime, another one with a new duration and set of rates are negotiated. The switch has to calculate a new schedule, and crossbar configurations are set according to the new schedule.

In chapter 3, we first show that there is a one to one correspondence between the service provided by a crossbar that alternates over a number of configurations in a time division multiplexed (TDM) manner and that by a three stage Clos network composed of crossbars that have fixed configurations.

We study the Slepian-Duguid algorithm originally developed for Clos networks (see [37] for an in depth treatment). With this algorithm rate updates can be made with minimal modification to the existing schedule in a simple and efficient way. We show that rate quantization is necessary for this approach to be successfully implemented, and discuss certain trade-offs. Then, we evaluate the necessary speedup that enables single crossbar switches to schedule contracts independently of each other. Our purpose is to accommodate rate updates of an I-O pair without changing the existing schedule of configurations. The main motivation for such an effort can be given as follows.

For a switch of size $N \times N$, there are N^2 input output (I-O) pairs. In practice, the desired rates between different I-O pairs may change independently of each other. Suppose each I-O pair updates its rate every T units of time on the average independently of other I-O pairs. This corresponds to N^2/T changes per unit time. This has a significant impact on the implementation complexity of the scheduling algorithms. Indeed, if rates are updated one at a time, even with rate quantization,

a given set of rates must be kept for about N time slots for satisfactory service quality with the type of schedulers we introduce. This corresponds to $O(N^3)$ time slots for maturity of an individual contract.

Multicast Support over a Single Crossbar Switch

In chapter 4, we study *multicast support* over crossbar switches. Many applications and traffic sources may ask for the same information to be sent to multiple points in the network, a situation called multicast. An increasing proportion of traffic on the Internet is multicast. Instead of packets being duplicated with a separate copy sent to each destination, the source should send a single packet to a *multicast address*. The network, then should form a *multicast tree* so that the packet can be sent to its appropriate destination in a more efficient manner. This requires the switches and the routers in the network to have the capability of copying a packet in an input link onto multiple output links. While multicast is not supported by many of the routers in the Internet, wide area multicast is made available via the Multicast backbone (Mbone), [32], [33]. The Mbone is a logical internet layered over the top of the current Internet. It consists of multicast-enabled routers that tunnel to each other through the existing Internet. Regular routers between two multicast enabled routers process only unicast headers and never have to worry about multicast addresses.

One trivial solution to multicast support is implemented by duplicating multicast packets upon arrival to a switch and treating each one as a separate unicast packet. However, higher throughput can be attained if we take advantage of the natural multicast properties of switching fabrics. For instance, the crossbar can easily copy one input cell to any number of outputs for which there is no conflict in a single cell time[5]. In Fig. 1-5, crosspoints (1,1) and (1,2) are connected. This enables the crossbar to copy the same cell at the first input link onto outputs 1 and 2 simultaneously rather than being sent at different times.

A number of different architectures and implementations have been proposed for multicast switches (see e.g., [4], [34]). Algorithms used in all of these implementations are based on cell scheduling rather than rate reservation. Due to the complicated nature of the multicast traffic, it is very hard to quantify the quality of service provided. All of these past papers focus on specific examples and present simulations that illustrate the performance for these examples. To our knowledge, there are no rate reservation based scheduling algorithms developed in the context

[5]Earlier, we stated the crossbar constraint such that a crossbar was incapable of making broadcast connections. In this part, we will get rid of that condition and adapt a *broadcast enabled* version of the crossbar constraint.

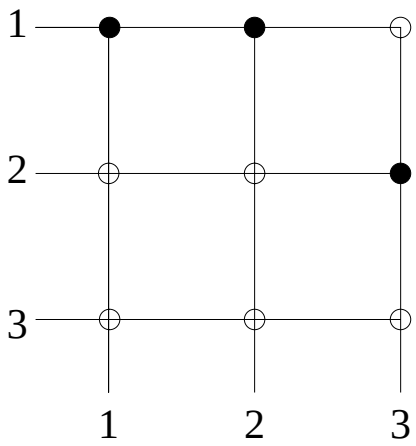

Figure 1-5: Crosspoints (1,1) and (1,2) are connected. This enables the crossbar to copy the same packet at the first input link onto outputs 1 and 2 simultaneously.

of multicast.

In this chapter, we assume a crossbar is capable of connecting an input to multiple outputs simultaneously. We call such crossbars to be *broadcast enabled*. We will show that, even with broadcast enabled crossbars, *support* of multicast rates is not possible without speedup. That is, even if the crossbar is capable of sending a copy of a cell to multiple outputs simultaneously, in the presence of *multicast multirate* connection requirements, the switch may be unable to provide the desired rates between its inputs and outputs. Indeed, we will show that, to support all admissible multicast traffic, some speedup is necessary, We then derive this necessary speedup. We then present other switch architectures that are capable of providing rate guarantees for all admissible multicast rates.

Multistage Switches

Despite the difference in their approaches to the problem of providing quality of service, the switch model considered in a majority of papers in the literature is almost identical: A single crossbar fabric and input and output buffers running as fast as the fabric[6]. This model is common to almost all the papers published in this literature.

The main limitation of the schedulers in the classical architecture is the crossbar constraint. Due to this constraint, at the input of a crossbar switch, a packet competes not only with other packets that are destined to the same output, but also with those sharing the same input. Given

[6]Even with no speedup, buffers at the output is desired in general since one may want to control the delay jitter of a flow at an output link rather than possibly sending its packets in bursts.

that the connection fabric is a single crossbar, there is no way to avoid this constraint. In Chapters 2 and 3, we study such algorithms and associated complexities.

There are a number of studies on different architectural choices as well. In a recent work by Iyer et.al. [35], it is shown how to simply modify the classical architecture to make use of the extra capacity of the links when the memories run slower than the line rate. The basic idea is to divide each pipe into multiple, say k, pipes by means of demultiplexers and use k switches in the middle stage before multiplexing packets into a single link again.

Another interesting modification to the classical architecture is proposed by Stephens et.al. [36] to overcome the contention between the cells sharing the same input. Instead of keeping the virtual output queues at the input, they are pushed inside the crossbar next to their corresponding crosspoints as illustrated in Fig. 1-6. For instance, the queue at input i that holds the packets destined to output j is moved to the crosspoint, (i, j). When a packet arrives with an input-output pair, it is directly forwarded inside the crossbar fabric, placed into the buffer located at the corresponding crosspoint. This way, multiple crosspoints in the same input (e.g., $(i, j_1), (i, j_2), \dots$) can be connected simultaneously and an input can send packets simultaneously to different outputs since the queues are physically separate. Hence, packets with different destinations do not contend at the input for the crossbar, even though contention between packets destined to the same output is still not eliminated. Each output can separately apply round robin scheduling between the queues located at the crosspoints which connect the inputs to this output. The disadvantage of this architecture is that it is drastically different from the traditional crossbar and hard to manufacture. Also, since the N^2 buffers in an $N \times N$ crossbar are physically separate, the advantages associated with statistical multiplexing are lost. Note that, in the regular VOQ scheme, we do not need to keep a separate queue at each input to implement virtual output queueing. We view the queues as linked lists, i.e., we can store all the cells arriving at an input in a single buffer, and assign a pointer to each cell to keep track of the output it is destined to.

Turner ([38]) considered the Benes architecture for packet switching, and showed that 100% throughput can be achieved over a Benes architecture, given that the packets between an input-output pair do not necessarily follow the same path. However, Turner did not focus on performance issues, such as packet delay and queue sizing.

In Chapter 5, we present an in depth study of multistage switches. In the first section, we develop the required mathematical tools, and using the intuition developed for the general structure

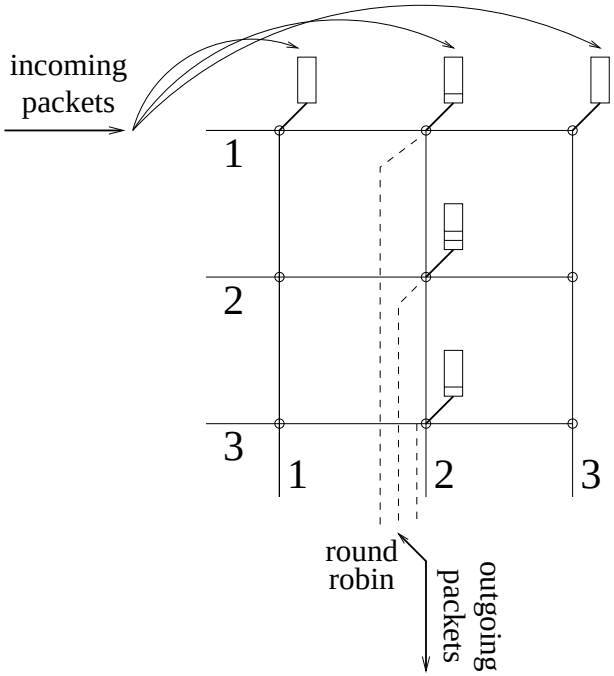

Figure 1-6: Virtual output queues are moved inside the fabric next to the corresponding crosspoints. Incoming packets are placed into these queues upon arrival. Each output link can then independently implement a round robin schedule among the those queues which have packets destined to itself.

of multistage switches, we propose a number of architectures along with associated routing and scheduling algorithms. In the second section, we will analyze the quality of service provided by these algorithms. Our algorithms and architectures illustrate how the presence of multiple paths between input output pairs can be exploited to improve the performance of a switch and simplify the scheduling algorithms. Also, we show that some of our architectures are capable of providing rate guarantees for all admissible multicast rates which would not be possible with a single stage broadcast enabled crossbar switch.

1.3 Optical Switches

In Chapter 6, we study a number of issues concerning service guarantees over optical networks. Most of our results are extensions of the results for electronic switches, but we need to account for different kinds of technological constraints in optical networking. To have a better grasp of the challenges involved in the problem of QoS over optical networks, we need an understanding of the current technologies and the structure of networks and internetworks. In the first part of this

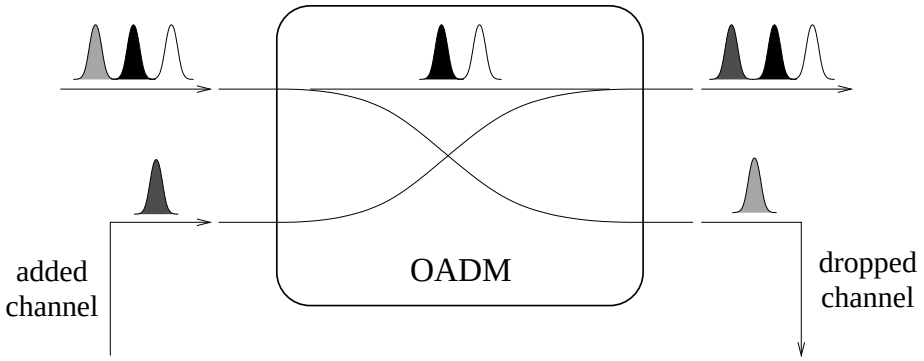

Figure 1-7: An add drop multiplexer can selectively add or drop a specific wavelength without opto-electronic conversion.

section, we present a brief introduction to wavelength switches and optical add-drop multiplexers (OADMs).

1.3.1 Wavelength Switch and OADM

Recall that WDM is based on wavelength routing. High-speed data flows, which consist of many time-division multiplexed channels, are associated with specific optical wavelengths. These flows are routed through the optical network by means of their wavelengths, without necessarily being opto-electronically converted, demultiplexed, and electronically routed.

One main feature of this kind of optical network lies in the possibility of performing these operations directly in the optical domain without requiring costly high-speed electronic equipment. Another feature is transparency, i.e., performing those functions independently of signal format. Optical add-drop multiplexers (OADMs) and wavelength (selective) switches represent the key elements that make all-optical networking possible.

A general scheme of an OADM is depicted in Fig. 1-7. It can selectively drop or add a specific wavelength on a fiber to which it is connected. The other wavelengths are passed through, optically. This optical *node* is characterized by several functionalities. For example, it could be rigid or flexible in adding/dropping one or more fixed wavelengths.

Packets are transferred to the next router on the way to their destination either over a direct link, if there exists one, or through optical switches. The transfer at the interface between wavelength switch and electronic router is handled by means of optical add-drop multiplexers as illustrated in Fig. 1-8.

A wavelength switch provides the possibility of routing individual channels coming from any of

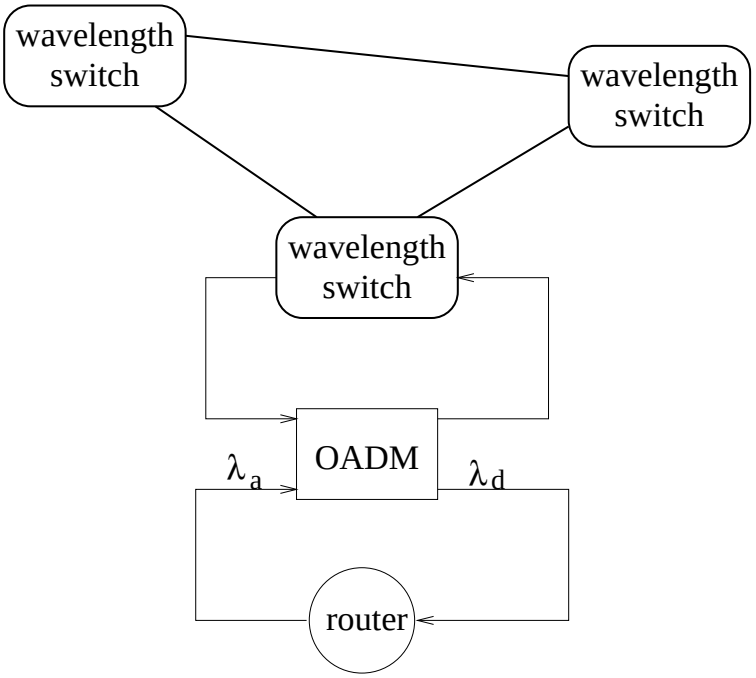

Figure 1-8: The transfer between the wavelength switches and the routers are taken care of by optical add drop multiplexers.

its input ports to any output port according to their wavelengths. There are several architectures, depending on whether the switch is rigid, rearrangable, or strictly non-blocking. The basic schemes are shown in Figs. 1-9-1-12.

The simplest configuration (Fig. 1-9) does not give any possibility of rearrangement, i.e., once a configuration is set, it cannot be changed unless the connections are modified physically. A rearrangable wavelength switch is depicted in Fig. 1-10, where a space division switching function has been introduced using optical crossconnects (OXCs). An OXC is functionally identical to a crossbar. The only difference is that an OXC is mainly based on opto-mechanical, acousto-optic, thermo-optic, or micro-electro-mechanical (MEMs) technologies, which are currently too slow for efficient packet switching. In this second architecture (Fig. 1-10), a separate OXC is used for each wavelength. Each wavelength of each input fiber can be routed to any output fiber not already using that wavelength through one of the OXCs. On the other hand, if the wavelength is already being used at the output fiber, the input cannot use it to set up a new connection. Thus, if an input and an output do not have a common unused wavelength, even if they are not fully utilized (i.e., if they are not carrying M wavelengths), a connection request cannot be met if the current wavelength assignments and the crossconnect configurations are not rearranged. If they can be rearranged, it

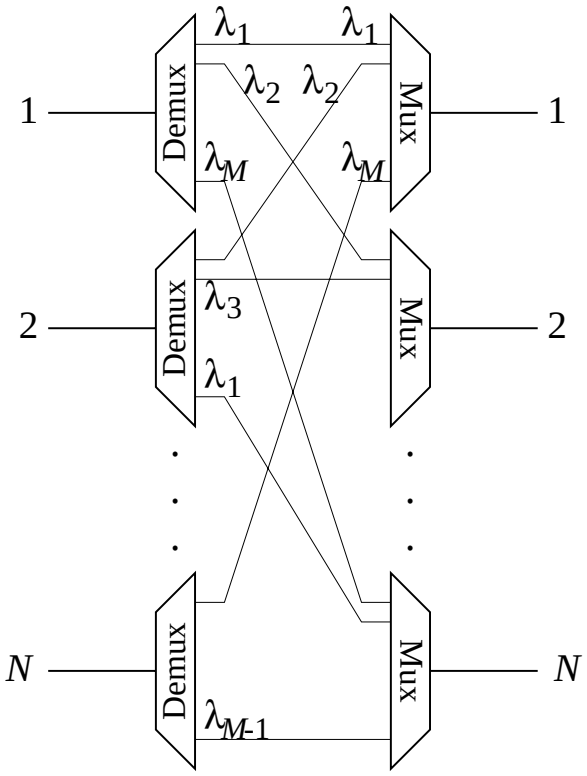

Figure 1-9: The static wavelength switch. Configuration cannot be rearranged.

can be shown that the switch given in Fig. 1-10 can meet a connection request between an input and an output that are not fully utilized. This architecture is thus called rearrangably non-blocking.

Rearrangement of the wavelength assignments is not desirable since rearrangement in one wavelength switch in a network induces a change in wavelength assignments of the others. To overcome this, we can use wavelength translators in conjunction with a large OXC inside the optical node, as shown in Fig. 1-11. This configuration eliminates the constraint that the input output pair must have a common unused wavelength to set up a connection. This architecture is strictly non-blocking if $M \geqslant N$, i.e., a connection request between an input and an output pair can be met if and only if both of them are not fully utilized. Therefore, it permits better wavelength reuse, but adds significant complexity to the routing node structure.

The wavelength switch given in Fig. 1-10 involves M OXCs each of which has a size of $N \times N$. To avoid large OXC which may be impractical, we can introduce blocking to a certain degree. An example is illustrated in Fig. 1-12 for four nodes and 3×3 OXCs. The number of OXCs in this scheme is larger than the number of wavelengths generated at each link. On the other hand, the

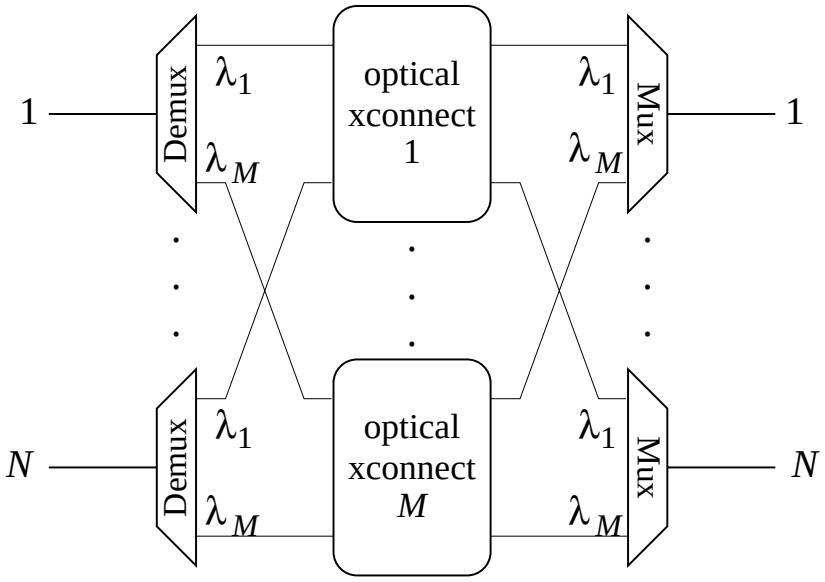

Figure 1-10: The rearrangable wavelength switch architecture.

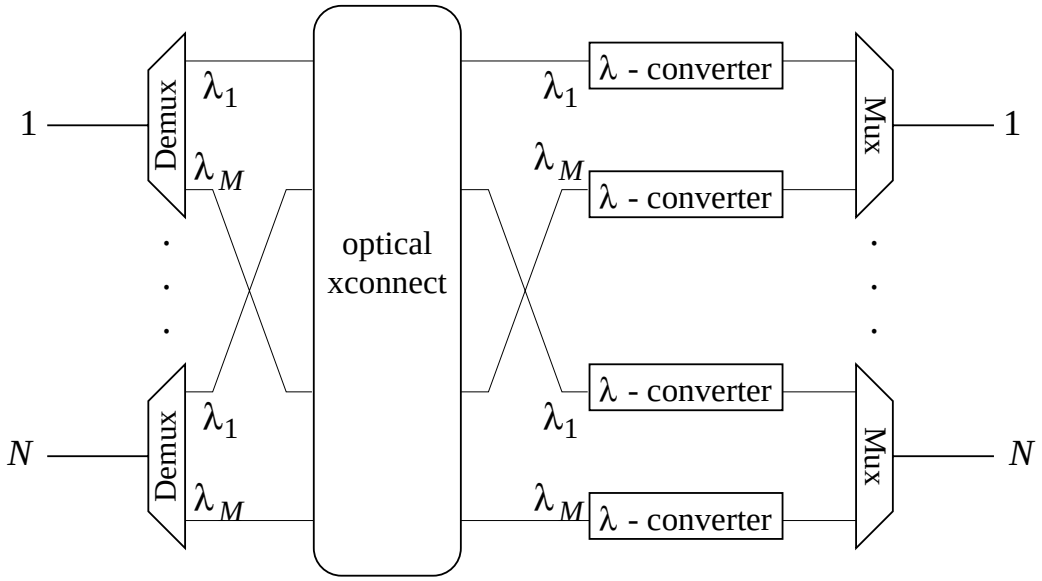

Figure 1-11: The strict sense non-blocking architecture employs an $NM \times NM$ OXC and NM wavelength converters.

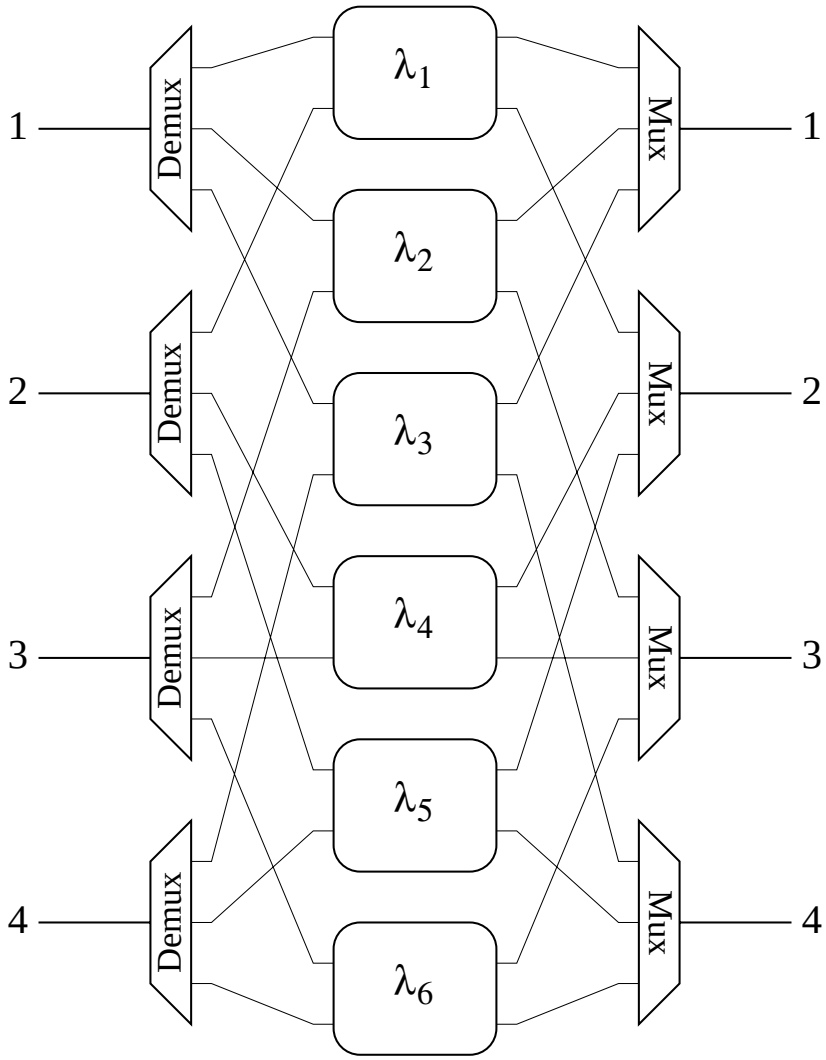

Figure 1-12: A blocking wavelength switch architecture. The size of the OXCs are smaller compared to the rearrangable version.

complexity of each OXC is cut significantly, and so is the overall crosspoint complexity. Crosspoint complexity is an important metric since the price of an OXC is roughly proportional to this metric. We will elaborate more on this architecture in this chapter.

Sone research has been carried out on the topic of photonic packet switching with the aim of fully exploiting the optical technology characteristics and the advantages of packet switching. Unfortunately, the technology for optical memory seems not yet mature enough to consider this viable, at least in the short/medium term. Solutions based on optical packet switching cannot be provided soon. One of the advantages of wavelength routing is that no optical buffer (or opto-electronic conversion of data) is needed at intermediate nodes.

1.3.2 Our Contribution

In Chapter 4, we consider two problems:

1. The first is how to support quality of service in the next generation optical Internet. Given that most of the current networks provide only best effort service, but at the same time some real-time applications require QoS support, it becomes apparent that for the optical Internet to be truly ubiquitous, one must address, among other important issues, how the WDM layer can provide basic QoS support. Even though a considerable amount of effort has been and is still devoted to developing QoS schemes for packet switches not many people have taken into account the properties of the WDM layer. Specifically, existing scheduling algorithms which are based on packet switching mandate buffering to achieve service guarantees.

   In Chapter 6, we will extend our previous results on rate reservation based scheduling algorithms for the optical wavelength switches. We will take the limitations of the OXCs and unavailability of optical memory technology into account, and modify the procedure we developed for crossbar switches to make them suitable for optical wavelength switches. First, we consider a rearrangably non-blocking wavelength switch architecture, and evaluate the number of OXCs and wavelengths necessary and show how to configure these OXCs to accommodate certain traffic requirements. Next, we will show that with an expansion in the number of wavelengths (and hence the number of OXCs), the architecture given in 1-10 can be made strictly non-blocking. That is, any change in the connection requirement between input output (I-O) pairs can be accommodated without a need for the rearrangement of the existing connections.

   A promising direction for network evolution lies in the migration of most of the switching burden into the optical domain from the electronics in order to exploit the huge fiber bandwidth. It is becoming more and more attractive since the cost of optical switching is decreasing with better technology and algorithms which enable more efficient resource utilization. The important question is how to manage the transformation from one technology to another.

   We use the insights we gained from our study of non-blocking switches and consider the following scenario. Suppose the rate of traffic generated by some set of end users approach optical wavelengths. We discuss how *efficiently* these end users can bypass routers as illustrated in Fig. 1-13, after deriving a relation between the the following parameters: the

amount of traffic injected by end users, the number of routers connected to an optical switch and *efficiency* which we define in this chapter.

2. Next, we consider the blocking architecture given in Fig. 1-12. We will study the rates supportable by this architecture and illustrate certain trade-offs involved. In particular, we will study the relation between the number of wavelengths, number of OXCs and the *region of rates* (between I-O pairs) that are supportable over this architecture: We will illustrate a system, at the center of which is the blocking wavelength switch, and find the region of rates that this system can support as a function of the number of OXCs in the switch.

1.4 Summary of the Thesis

In Chapter 2, we focus on service guarantees over a single crossbar switch. First, we give the problem model, definitions and some properties of Birkhoff's decomposition. We show that the bounds given in [15] for the delay and the delay jitter are indeed tight. Next, we discuss rate quantization. We give the algorithm, our main theorem on rate quantization and its proof. Then, we explain how to use rate quantization along with some speedup to improve the quality of rate guarantees so as to be suitable for variable bit rate sources. Then, we present a performance analysis for the single stage crossbar switch with rate quantization. We also give a probabilistic scheduling scheme and talk about the associated performance issues. Finally, we give the conclusions and possible future extensions.

In Chapter 3, we study scheduling for dynamically changing traffic in single crossbar switches. After giving some definitions, we first show that there is a one to one correspondence between the service provided by a crossbar that alternates over a number of configurations in a time division multiplexed (TDM) manner and that by a three stage Clos network composed of crossbars that have fixed configurations. We, then use some well known properties of Clos networks to develop a simple algorithm by which incremental scheduling updates can be made. Using the same model, we evaluate the necessary speedup for independent scheduling of I-O pairs.

In Chapter 4, we show that a single crossbar switch cannot support multicast rates without speedup even if it is capable of making multicast connections. Then, we derive the necessary speedup for the support of multicast rates and independent scheduling of I-O pairs.

In Chapter 5, we present an in depth study of multistage switches. In the first part, we develop some required mathematical tools. Using the intuition developed for the general structure

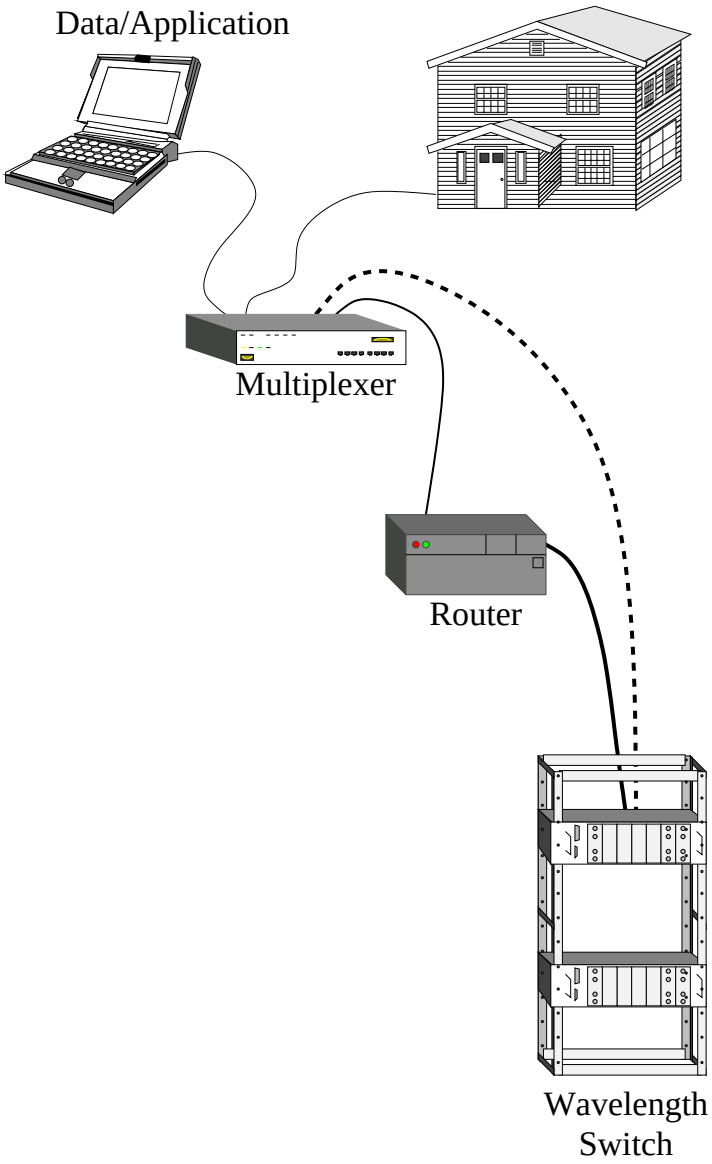

Figure 1-13: With increasing application rates, it becomes more and more feasible to move fibers closer to the end users. If the amount of traffic at some point in the edge grows to optical channel rates, it starts to make sense to bypass routers.

of multistage switches, we propose a number of architectures along with associated routing and scheduling algorithms. In the second part, we analyze the quality of service provided by these algorithms. Our algorithms and architectures illustrate how the presence of multiple paths between input output pairs can be exploited to improve the performance of a switch and simplify the scheduling algorithms. We show that satisfactory performance can be achieved with very simple multistage architectures; and even support of multicast rates is possible with one of them without a need for speedup.

In Chapter 6, we study service guarantees over certain non-blocking and blocking wavelength switch architectures. First we focus on the non-blocking switch, and show how to apply rate quantization to provide rate guarantees. After introducing *strictly non-blocking wavelength assignment*, we consider the blocking architecture given in Fig. 1-12. We specify the region of rates supportable by this architecture and illustrate certain trade-offs involved.

In Chapter 7 we summarize the results developed in this thesis and give further directions for future research.

The appendices contain some basic definitions in the theory of majorization, a number of theorems that are used in our proofs and the alternate version of rate quantization algorithm with its proof of correctness.

Chapter 2

Providing Service Guarantees over Single Crossbar Packet Switches

In this chapter, we study *rate quantization*: what it is, how it is implemented and its impacts on the performance of scheduling algorithms for single crossbar switches. Basically, rate quantization converts a set of rates into another discrete set of rates, which is then used as an input to a scheduling algorithm. First, we motivate the idea and illustrate some fundamental limitations of schedulers for rates which are picked from a continuous set. We will show that for an $N \times N$ switch, along with some (small) speedup, rate quantization improves the delay and delay jitter by a factor of order $O(N)$ even with simple schedulers. We also discuss complexities of schedulers with and without rate quantization. We show that quantization greatly simplifies scheduling algorithms.

The rest of the chapter is organized as follows. In the next section, we give the problem model, definitions and some properties of Birkhoff's decomposition. We show that the bounds given in [15] for the delay and delay jitter are indeed tight. Next, we discuss rate quantization. We give the algorithm, our main theorem on rate quantization and its proof. Then, we explain how to use rate quantization along with some speedup to improve the quality of rate guarantees so as to be suitable for variable bit rate sources. In Section 3, we present the relation between the performance and the speedup and other implications of our results. Then, we illustrate that the same performance can be achieved without speedup by putting a limit on the amount of load that each link supports. In the fourth section, we give a probabilistic scheduling scheme and talk about associated performance issues. Finally, we give the conclusions and possible future extensions in Section 5.

2.1 Problem Model

In this section, we introduce notation and present a mathematical model for crossbar switches. The general structure of a crossbar switch was given in the first chapter. Our model is built on that structure.

2.1.1 Definitions and Assumptions

We assume that each link has an identical capacity, and define a *time slot* as the time it takes for one cell to be transmitted over a link (This is also referred to as a *cell slot* in the literature). We also define a *service slot* as the time it takes one cell to be transferred from an input to an output of the crossbar. We assume that cell transfers are made synchronously over all the inputs, and a crossbar can switch from one configuration to another in negligible time. For the time being, we assume no speedup, i.e., a service slot is identical to a time slot. All of the results we present will be for a switch of size $N \times N$ but we believe that the generalization to an asymmetric ($N \times M, N \neq M$) switch is straightforward.

We assume connection oriented contracts for which durations (in number of time slots) and the number of cells to be transferred between every I-O pair within the lifetime of the contract are specified. Thus, a rate, R_{ij}, can be associated with the input-output pair (i, j) as the ratio of the number of cells to be transferred between the pair to the lifetime of the contract. Hence, R_{ij} takes on values from the set $[0, 1]$. An admission controller makes sure that no input or output link is oversubscribed, i.e., $\sum_i R_{ij} \leqslant 1$ for all j and $\sum_j R_{ij} \leqslant 1$ for all i. Succinctly, we can represent each R_{ij} as the (i, j) entry of a matrix, R. The two admission control inequalities imply that R is a doubly sub-stochastic matrix. It was shown by von Neumann ([24]) that for every doubly sub-stochastic matrix, R, there exists a doubly stochastic matrix, Q such that $Q_{ij} \geqslant R_{ij}$, $\forall i, j$. An algorithm that generates such a doubly stochastic matrix is also given in [24]. In this paper, we assume that the rate request matrix, R, is doubly stochastic. Recall that it is only with connection oriented service that rate reservation based scheduling algorithms can be developed since such algorithms require the prior knowledge of the desired rates between I-O pairs.

In this chapter, we assume that contracts between all I-O pairs are made simultaneously. Therefore, we define a contract with a rate matrix R and an associated duration T. At the maturity of a contract, a new contract (i.e., a new rate matrix and a new duration) is negotiated for all I-O pairs simultaneously. In reality, the desired rate between different I-O pairs generally change at

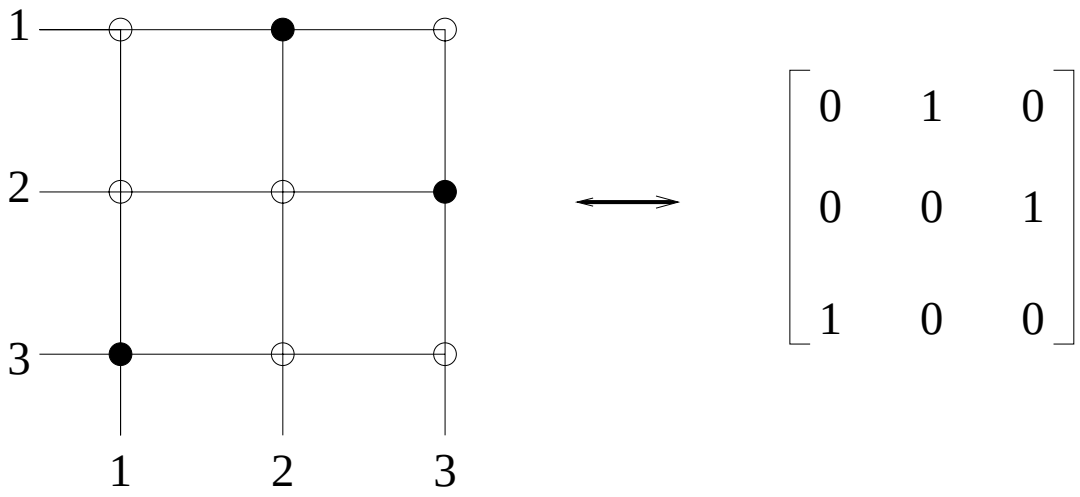

Figure 2-1: There is a one to one correspondence between permutation matrices and crossbar configurations.

different times, and thus at the end of a contract only a small number of I-O pairs need a rate update. Therefore, it makes more sense to define a separate duration for each I-O pair. However, in this chapter, we assume that the rate updates are made simultaneously and there is only one duration associated with a contract. We will deal with individual rate updates and *independent contracts* in the next chapter. In this chapter, we assume that the maturity of the rate contract between each pair is the same.

The switch is responsible for providing the desired number of service opportunities between each I-O pair. If the lifetime of the contract is T, then the switch will go through T configurations and at least TR_{ij} of them should connect input i to output j for the terms of the contract to be met. Suppose, for some switch schedule, the switch goes through configurations such that input i and output j are connected for $D_{ij}(t)$ time slots, by time $t \leqslant T$. We call the difference, $tR_{ij} - D_{ij}(t)$, the service lag for I-O pair (i, j) at time t.

Let us restate the crossbar constraint without broadcast: In a single time slot, no input can be connected to more than one output, and no output can be connected to more than one input. Thus, there corresponds a distinct permutation matrix for every feasible configuration in which no input and no output remains unmatched. Such a pairing of switch configuration and permutation matrix is illustrated for a 3×3 crossbar in Fig. 2-1. There is a 1 in every position of the matrix where the crosspoint in the corresponding location of the crossbar is connected. Suppose $P(1), \ldots, P(t)$ are the corresponding permutation matrices for the the configurations that the crossbar goes through

in $(0, t]$. Then

$$D(t) = \sum_{l=1}^{t} P(l) \tag{2.1}$$

Next, we give a couple of theorems that provide some fundamental insight about our algorithms.

2.1.2 Fundamentals of Rate Reservation Based Scheduling

First, let us focus on contracts with infinite durations and then study the algorithm developed in [15].

Long Term Contracts

For a contract with rate matrix, duration pair, (R, T), let us define the corresponding infinite duration contract as the one with the same rate matrix and $T \to \infty$

Definition 2.1 *A contract is* supportable *if there exists a schedule of permutation matrices that produce a service history, $D(t)$, for which the service lag remains upper bounded for all I-O pairs and for all $t > 0$.*

If there exists a schedule for which there exists a certain point, t, in time such that $tR_{ij} - D_{ij}(t) \leqslant 0$ for all pairs (i, j), then we say the rate matrix R is perfectly supportable *at time t.*

If R is perfectly supportable at time t, then it is also perfectly supportable at times kt $\forall k \in \mathbb{Z}^+$ since the switch can implement a periodic schedule which repeats itself every t seconds. Note that, by definition supportability is relevant only for infinite duration contracts, thus, instead of saying "the infinite duration contract with rate matrix R is supportable," we simply say, "R is supportable." Given that R is supportable with the schedule $D(t), t > 0$,

$$\lim_{t \to \infty} \frac{D_{ij}(t)}{t} = R_{ij}$$

In a crossbar switch, perfect support may not be possible for some supportable traffic.

Theorem 2.1 *A contract with the rate matrix R is supportable if and only if R can be written as a convex combination of permutation matrices.*

Proof: First, let us prove the only if part. Suppose R is supportable; then, there exists a scheduler and some $B < \infty$ such that $tR_{ij} - D_{ij}(t) \leqslant B$ for all pairs (i, j) and for all t. Hence,

$$\lim_{t\to\infty} [tR - D(t)] \leqslant B\vec{e}\vec{e}^{T} \tag{2.2}$$

where $\vec{e}$ is the N dimensional vector all of whose entries are 1, and $\vec{e}^{T}$ is its transpose. Note also that the inequality is entrywise. Since $D(t)$ is a sum of t permutation matrices as given in Eq. (2.1) and B is not a function of t, (2.2) can be written as,

$$R = \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=1}^{t} P(\tau) \tag{2.3}$$

We complete the proof by noting that the right side of the above equation (2.3) is a convex combination of finitely many permutation matrices.

Conversely, if there exists a set of non-negative coefficients, $\{\phi_1, \ldots, \phi_M\}$ such that

$$R = \sum_{l=1}^{M} \phi_l P_l \tag{2.4}$$

then it was shown in [15] that the service lag remains upper bounded at all times using a packetized processor sharing schedule of the configurations with non-zero coefficients in the decomposition. We will discuss this in the following section.

Birkhoff's Decomposition and RR Based Scheduling

The main idea in [15] is the use of a decomposition algorithm based on Birkhoff's theorem ([25]).

Theorem 2.2 (Birkhoff) *The permutation matrices constitute the extreme points of the set of doubly stochastic matrices. Moreover, the set of doubly stochastic matrices is the convex hull of the permutation matrices.*

Birkhoff's theorem is illustrated in Fig. (2-2) (for simplicity, the illustration is made in two dimensional space). Thus, every doubly stochastic matrix can be written as a convex combination of permutation matrices. The set of all $N \times N$ doubly stochastic matrices can be regarded as a convex polytope in $(N-1)^2$ dimensions[1]. Hence, the number of permutation matrices sufficient to

[1]The intuition behind this is as follows. There are N^2 entries in the matrix and $2N$ conditions (every row and every column sums to 1). But these conditions are not linearly independent; indeed, one of them is redundant. Hence,

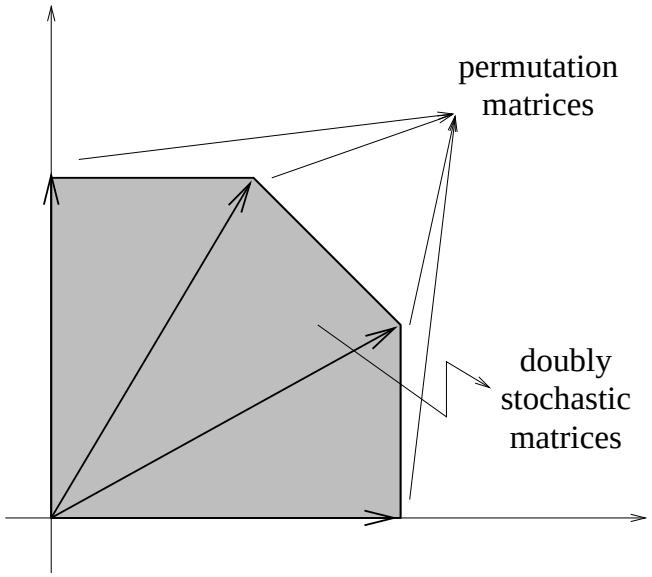

Figure 2-2: The set of doubly stochastic matrices is the convex hull of the permutation matrices.

represent an $N \times N$ doubly stochastic matrix is $1+(N-1)^2$. This follows from the Caratheodory's theorem ([26]):

Theorem 2.3 (Caratheodory) *If A is a convex subset of $\Re^d$, then each point in A is expressible as a convex combination of no more than $d+1$ extreme points.*

Thus, for any rate request matrix, the following decomposition can be made:

$$\begin{aligned} R &= \sum_{i=1}^{(N-1)^2+1} \phi_i P_i \\ 1 &= \sum \phi_i \end{aligned} \tag{2.5}$$

In view of (2.5), the fraction of time that the crossbar has to spend configured to the permutation matrix P_k is equal to ϕ_k. Since only one permutation matrix can be set each time slot, a schedule of the corresponding configurations must be constructed according to the weights in the decomposition. In [15], the use of Packetized Generalized Processor Sharing (PGPS) is proposed. Each permutation matrix is treated as a user in PGPS, and the weight of a matrix represents the desired rate of that user. Users are assumed to be backlogged all the time and the finishing times of the next unserved token for each user in the corresponding Generalized Processor Sharing (GPS)

$N^2 - 2N + 1$ points in the polytope can be chosen independently.

system is calculated starting at time 0. Indeed, the following iterative relation can be written for the finishing times of the tokens for user k:

$$FT_k^l = FT_k^{l-1} + \frac{1}{\phi_k}$$

If user k has the unserved token with the smallest finishing time, it is given service in the next time slot, i.e., the crossbar configuration corresponding to the permutation matrix, P_k is set. Note that the described algorithm is actually a special case of the PGPS algorithm where each user is backlogged infinitely. This approach does not necessarily lead to a work conserving service, and in fact, at the time a crossbar configuration is set, all the VOQs which are supposed to take advantage of the connection to send a cell through the crossbar may be empty. Despite this, the configuration is still set. This fact reflects the fundamental difference between RR based scheduling and cell scheduling. A cell scheduling algorithm would look for a better configuration where a certain objective function is maximized such as the number of transferred packets. RR based scheduling algorithms are not aggressive all the time but perform well on the average, i.e., as we shall see in the next section, the algorithm presented in [15] manages to support the set of all admissible rates in the long run, but may perform unsatisfactorily in the short run.

Computationally, running Birkhoff's decomposition along with von Neumann's algorithm (to make R doubly stochastic) is fairly complex. Indeed, for an $N \times N$ switch, von Neumann's algorithm and Birkhoff's decomposition have a computational complexity[2] of $O(N^2)$ and $O(N^{4.5})$ respectively. Also, it is not possible to run the decomposition and set configurations simultaneously; the first crossbar configuration can be set only after the decomposition algorithm terminates. If the input traffic changes frequently, or contracts have short durations, PGPS with a plain Birkhoff decomposition approach may be infeasible.

2.2 Rate Quantization

In this section, we will study the performance of the plain Birkhoff-von Neumann algorithm along with PGPS scheduling, and discuss the cases where it performs poorly. Then, we present our approach to improving the performance.

[2]The complexity analysis of von Neumann's algorithm can be found in [24]. For Birkhoff's decomposition, it takes $(N-1)^2+1$ iterations each of which involve a complexity of $O(N^{2.5})$

2.2.1 Service Guarantees with the Plain Birkhoff-von Neumann Approach

In this section, we will discuss why the Birkhoff-von Neumann algorithm along with PGPS may fail in the short term. We mentioned that there are two possible mechanisms for the delay that a packet experiences at a switch between its arrival and departure. The first is the randomness of the arrival process and the second is the jitter in the service provided by the switch. Even if the provided service perfectly matches that desired between an I-O pair, the former is not eliminated. We are mainly interested in the delay caused by the latter, since we are interested in whether the switch is successful in meeting the desired rate. For instance, for perfect service between any two points in time, $s < t$, the number of service opportunities[3], $D_{ij}(t)$, provided for the I-O pair (i, j) by time t needs to satisfy,

$$D_{ij}(t) - D_{ij}(s) \geqslant (t - s)R_{ij}$$

However, in [15], it was shown that, for all (i, j),

$$D_{ij}(t) - D_{ij}(s) \geqslant (t - s)R_{ij} - \left[(N - 1)^2 + 1\right] \tag{2.6}$$

with PGPS scheduling. Eq. (2.6) implies that the number of service opportunities provided for an I-O pair cannot be behind its desired amount by more than $(N - 1)^2 + 1$ units, which is the *service lag* for the I-O pair. The service lag is a metric to measure the jitter in the service, and it gives us the increase in the queue size due to the imperfect service. Namely, even if packets arrive deterministically, the number of packets that accumulate in the queue can be as high as the maximum service lag[4].

Since the maximum service lag given in (2.6) is constant (not a function of time), it is clear that all admissible rates are supportable using PGPS with Birkhoff's decomposition. However, we shall argue in this section that the bound in (2.6) is indeed tight, i.e., for some set of R_{ij}'s, there exist certain points in time for which the service lag is indeed close to $(N - 1)^2 + 1$ for some I-O pairs. Hence, in the short term (within time periods comparable to $(N-1)^2+1$), the rate guarantees may

[3]We define a service opportunity for an I-O pair to be the number of time slots that the pair is connected.

[4]In some sense, the service lag is the natural metric for CBR service. If, service lag is identical for two flows with different rates, then the delay they experience will be inversely proportional to their rates. In fact, any G/G/1 queueing system has the same property that if the service and arrival rates are scaled by a factor, the (steady state) expected unfinished work is cut by the same factor.

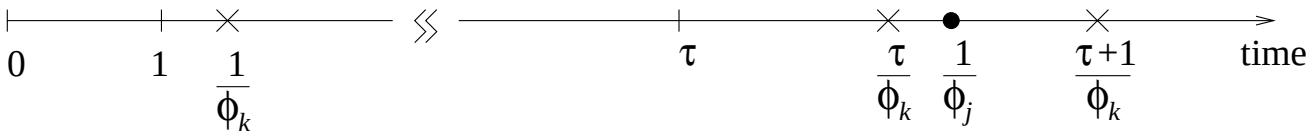

Figure 2-3: Finishing times of first t cells of user k and the first cell of user j in the corresponding GPS system.

be unsatisfactory. For instance, at the end of a short contract (e.g., O(N) time slots), an I-O pair may end up with 0 service opportunities, even if its VOQ is not empty. In fact, as shall be shown, what happens in such short periods of time is quite random with the plain Birkhoff approach.

First, we will talk about PGPS service since the algorithm under consideration is based on it. Suppose, K users share a single link and the rate desired by the k^{th} user is ϕ_k. It was shown in [27] that the number, D_k, of cells served with the PGPS algorithm for user k by time t is close to the desired amount, $\phi_k t$:

$$t\phi_k - 1 \leqslant D_k(t) \leqslant (t + K - 1)\phi_k \tag{2.7}$$

Claim 2.1 *The upper and the lower bound given in (2.7) are tight.*

Proof: We prove the claim by showing an example where the bounds will be tight at some point in time. Suppose there exists a user, k, with $\phi_k \gg \phi_l,\ \forall l \neq k$. Initially, user k will dominate all the others and get many services before the first service opportunity is given to any other user. As illustrated in Fig. 2-3, the finishing time of the first τ cells of user k is earlier than that of the first cell of any other user in the corresponding GPS system. Thus, in the first τ time slots, user k will get service. At the end of τth time slot, the number of service opportunities given to user k is,

$$\begin{aligned} D_k(\tau) &= \tau \\ &= \phi_k\tau + (1 - \phi_k)\tau \end{aligned} \tag{2.8}$$

Since,

$$\sum_{j \neq k} \phi_j = 1 - \phi_k$$

there exists a user, $j \neq k$ with rate,

$$\phi_j \geqslant \frac{1-\phi_k}{K-1} \tag{2.9}$$

Since user k is served τ times before any other user is served (shown in Fig. (2-3)),

$$\frac{\tau}{\phi_k} \leqslant \frac{1}{\phi_j} \tag{2.10}$$

Substituting (2.9) in (2.10), we get

$$\tau \leqslant \phi_k \frac{K-1}{1-\phi_k} \tag{2.11}$$

Finally, plugging (2.11) in (2.8), we get

$$\begin{aligned} D_k(\tau) &\leqslant \phi_k \tau + (1-\phi_k)\frac{K-1}{1-\phi_k}\phi_k & (2.12)\\ &= \phi_k(\tau + K - 1) & (2.13) \end{aligned}$$

Note that (2.12) is satisfied with equality at time $\tau = \frac{\phi_k}{\phi_j}$ if $\phi_j = \frac{1-\phi_k}{K-1}$. Thus, the upper bound given in (2.7) is indeed tight. Also, since user j is served for the first time before user k's $(\tau+1)$st service,

$$\phi_j(\tau+1) \geqslant \phi_k \tag{2.14}$$

Since, until the end of time slot $\tau+1$, the number of cells served for user j is 0,

$$\begin{aligned} D_j[(\tau+1)^-] &= 0 \\ &\leqslant (\tau+1)\phi_j - \phi_k & (2.15) \end{aligned}$$

We have equality in (2.15) if (2.14) is satisfied with equality. Since ϕ_k can be arbitrarily close to 1, the lower bound given in (2.7) is also tight (replace k with j and τ with $\tau+1$).

Claim 2.2 *The lower bound given in (2.6) is tight. For some I-O pairs, the cell delay can be* $O(N^3)$.

Proof: Similarly, we prove the claim by showing an example where the service lag is $\mathrm{O}(N^2)$ in

the worst case for an I-O pair with a rate N^{-1} (as shown in the first chapter, for a given I-O pair, delay = service lag/rate). Consider the following $N \times N$ rate request matrix:

$$R = \begin{bmatrix} \frac{N-1}{N} & \frac{1}{N} & 0 & \cdots & 0 \\ \epsilon_{21} & \frac{N-1}{N} & \epsilon_{23} & \cdots & \epsilon_{2N} \\ \epsilon_{31} & 0 & \frac{N-1}{N} + \epsilon_{33} & \cdots & \epsilon_{3N} \\ \vdots & \vdots & & \ddots & \\ \epsilon_{N1} & 0 & \epsilon_{N3} & \cdots & \frac{N-1}{N} + \epsilon_{NN} \end{bmatrix} \tag{2.16}$$

where we choose $\{\epsilon_{ij}\}$ such that R is a doubly stochastic matrix. Now, let us look at the $(N-1) \times (N-1)$ matrix whose entries are the elements of the set $\{\epsilon_{ij}\}$ which are placed with the identical relative ordering as in R. Each row and column of this matrix sums up to $\frac{1}{N}$. Hence it is $\frac{1}{N}$ times a doubly stochastic matrix. As we mentioned before, $(N-2)^2 + 1$ permutation matrices is sufficient to represent any $(N-1) \times (N-1)$ permutation matrix ([25]). It is also shown in [25] that this number cannot be replaced with a smaller number. Namely, for some set of $\{\epsilon_{ij}\}$ Birkhoff's algorithm will not terminate in less than $(N-2)^2 + 1$ steps for our imaginary $(N-1) \times (N-1)$ matrix. It is not hard to realize that R has a diagonal with large entries all of which are not less than $\frac{N-1}{N}$. Thus, one possible Birkhoff decomposition includes the $N \times N$ identity matrix with a weight of $\frac{N-1}{N}$. The sum of the coefficients of the other permutation matrices is $\frac{1}{N}$, and hence they all must have a 1 in their first row, second column. The rest of the decomposition is actually almost identical to the decomposition of the $(N-1) \times (N-1)$ matrix of $\{\epsilon_{ij}\}$ we just described, except that the permutation matrices are all $N \times N$ since they are all augmented with a 1 in location $(1,2)$ and zeros elsewhere in the first row and second column. Thus, for some selection of $\{\epsilon_{ij}\}$ we can make the algorithm last for $1 + \left[(N-1)^2 + 1\right]$ steps. Note that, instead of choosing the set of $\{\epsilon_{ij}\}$ directly, we can choose $(N-1)^2 + 1$ coefficients and permutation matrices. Let us choose the jth coefficient to be $\frac{1}{N^3} + \xi_j$ where $\xi_j = o\left(N^{-3}\right)$. Thus Birkhoff's algorithm can end up with the

following decomposition:

$$R = \frac{N-1}{N} P_1 + \left(\frac{1}{N^3} + \xi_2\right) P_2 + \cdots + \left(\frac{1}{N^3} + \xi_{N^2-2N+3}\right) P_{N^2-2N+3} \tag{2.17}$$

where P_1 is the $N \times N$ identity matrix, $P_2(1,2) = \cdots = P_{N^2-2N+3}(1,2) = 1$. The schedule for each permutation matrix is determined according to the PGPS algorithm. Thus, we end up in the same situation as we described in the proof of Claim 1. The first permutation matrix, P_1 will be served[5] for a long time before any other permutation can be served, i.e., it will be served until (integer) time τ where,

$$\frac{N}{N-1}\tau \leqslant \min_{j \leqslant (N^2-2N+3)} \frac{1}{\frac{1}{N^3} + \xi_j}$$

Since $\xi_j = o\left(N^{-3}\right),\ \forall j$,

$$\tau = \alpha N^2 (N-1) \tag{2.18}$$

where $\alpha \approx 1$. Thus, until time τ, only P_1 will be served and $D_2(\tau) = \cdots = D_{N^2-2N+3}(\tau) = 0$ where $D_j(t)$ is the number of times that a permutation is served by time t. Hence,

$$\begin{aligned} D_j(t) &= 0 \\ &= \left(\frac{1}{N^3} + \xi_j\right)\tau - \left(\frac{1}{N^3} + \xi_j\right)\tau \end{aligned} \tag{2.19}$$

for all $j \neq 1$. One can realize that I-O pair (1,2) gets connected when any one of the permutation matrices other than the first one is served. Indeed,

$$D_{12}(\tau) = \sum_{l=2}^{N^2-2N+3} D_l(\tau) \tag{2.20}$$

[5] "Serving a permutation matrix" means making the crossbar connection corresponding to that permutation matrix.

Plugging Eq. (2.18) and (2.19) in Eq. (2.20),

$$D_{12}(t) = \sum_{j=2}^{N^2-2N+3} \left(\frac{1}{N^3}+\xi_j\right)\tau - \left[\alpha N^2(N-1)\right] \sum_{j=2}^{N^2-2N+3} \left(\frac{1}{N^3}+\xi_j\right) \quad (2.21)$$

$$= \frac{1}{N}\tau - \alpha\frac{1}{N}(N-1)^3 + o(N^2) \quad (2.22)$$

$$= \frac{1}{N}\tau - \alpha\left(N^2-3N+3\right) + o(N^2) \quad (2.23)$$

where (2.22) follows since the first summation in (2.21) is $\frac{1}{N}t$.

We just showed that there exists a set of admissible rates, in which Birkhoff's decomposition can end up with a decomposition where the service lag for a flow with PGPS scheduling can be $O(N^2)$. The example gives us a sense about when such a worst case can happen. The decomposition may terminate with $O(N^2)$ permutation matrices and thus, there exists certain entries which are non-zero in multiple permutation matrices (in our example, $P_j(1,2)$ is non-zero in $(N-2)^2+1$ permutation matrices.). But, the PGPS algorithm takes the finishing time of every permutation matrix into consideration rather than every single I-O pair and this leads to the higher service lag. For instance, in the example we presented, I-O pair (1,2) is asking for a rate of $\frac{1}{N}$ and it gets service when the scheduler gives a service opportunity to the last $(N-2)^2+1$ permutation matrices. However, each of these permutation matrices has a weight of $\approx \frac{1}{N^3}$ and they all have to wait for a long time ($O(N^3)$) before they get the first service. This results in high delay jitter (no service for a long time and then a burst of service opportunities after) and hence a high service lag.

At this point, we can argue that there may exist packet scheduling algorithms other than PGPS for scheduling crossbar permutation matrices and they may enhance the performance. One such algorithm is worst case fair weighted fair queueing[6], WF^{2}Q ([23]). The main difference between PGPS and WF^{2}Q can be stated as follows. In the former, every user is eligible to be scheduled at any point in time and the scheduled user has the smallest finishing time in the corresponding GPS system. In WF^{2}Q however, if a user did not begin taking service in the corresponding GPS scheduler, it is not eligible to be scheduled even if it has the smallest finishing time. For a user to get its mth service, it must be the case that, in the corresponding GPS system, the $(m-1)$st service for that user is complete and mth one is currently in progress (or already complete). It turns out that this simple modification to PGPS improves the upper bound given in (2.7) to $\phi_k t+1$

[6]PGPS is also known as (vanilla) weighted fair queueing (WFQ).

and hence the provided service is always within 1 unit of the desired.

In our example, if we used a WF^{2}Q scheduler, the first permutation matrix (the one with weight $\frac{N-1}{N}$) would not be scheduled more than N time slots in a row. Thus, the delay for the first service of the user, $\mathcal{F}_{12}$, whose weight is $\frac{1}{N}$ is improved by a factor of N^2 (from O(N^3) to O(N)).

Motivated with the above fact, we want to find out if WF^{2}Q improves the bound on the service lag for the switch scenario. The answer is negative which can be seen considering the following R, which is quite similar to the one given in (2.16):

$$R = \begin{bmatrix} \frac{N-1}{N} & \frac{1}{2N}+\omega & \frac{1}{2N}-\omega & 0 & \cdots & 0 \\ \epsilon_{21} & \frac{N-1}{N} & 0 & 0 & \cdots & \epsilon_{2N} \\ \epsilon_{31} & 0 & \frac{N-1}{N} & \epsilon_{34} & \cdots & \epsilon_{3N} \\ & \vdots & \vdots & & \ddots & \\ \epsilon_{N1} & 0 & 0 & \epsilon_{N4} & \cdots & \frac{N-1}{N}+\epsilon_{NN} \end{bmatrix} \tag{2.24}$$

In this new matrix, the user with weight $\frac{1}{N}$ is split into two users with one user slightly of higher weight (by ω =o(N^{-1})). Similarly, we select $\{\epsilon_{ij}\}$ (there are $(N-2)^2$ of them) such that the Birkhoff's decomposition ends up with $1+\left[(N-2)^2+1\right]$ permutation matrices each of which has a weight of O$\left(N^{-3}\right)$. Other than the identity matrix, almost half of these matrices will have a 1 in location $(1,2)$ and the others in $(1,3)$. If we use a scheduler that operates using only the coefficients of the permutations and treat every permutation matrix equally, the delay experienced by a user can still be O(N^3). For instance with WF^{2}Q scheduler, if the coefficients of all of the permutation matrices which serves user $(1,2)$ are greater than those of user $(1,3)$, then there exists a set, $\{\epsilon_{ij}\}$, such that user $(1,3)$ experiences a delay of $\frac{1}{2}\alpha N^2(N-1)$ where $\alpha \approx 1$ before getting the first service opportunity. Thus, the service lag is O(N^2).

We believe that this is true for a general class of scheduling algorithms, and thus state the following conjecture.

Conjecture 2.1 *For the set of scheduling algorithms in which the crossbar configurations are scheduled using only the coefficients of Birkhoff's algorithm, disregarding the individual rates between I-O*

pairs, the bound given in (2.6) is tight.

2.2.2 Rate Quantization

There are a number of problems with the approaches using processor sharing with Birkhoff's decomposition. First of all, the complexity of the Birkhoff algorithm is huge and more importantly, it is a one time complexity. Namely, it cannot be run simultaneously as the permutations are scheduled. The decomposition must be complete before the first configuration can be scheduled.

The service lag of (N^2) may correspond to a delay of tens to hundreds of milliseconds for typical commercial packet switches, which is very undesirable. For instance, for a 256×256 ATM switch with lines of rate 155.5 Mbps, a cell delay of up to ~ 179 msecs can be experienced for some of the I-O pairs due to imperfections in provided service. With such delays, service contracts cannot be made for short time periods. For example, for the 256×256 switch, rate contracts need to be held for periods of order seconds for the service contract to be met reasonably closely. This rules out the algorithm as described for current multimedia applications whose traffic dynamics vary in time scales from microseconds to a few milliseconds. In this section, we will show how to use even a small speedup to cut the service lag to $O(N)$ and make the algorithms simpler at the same time.

In the above analysis, there was no speedup, i.e., only one cell can be transferred between an I-O pair per time slot. In this section, we will loosen this constraint and allow S cells to be transferred per time slot. Thus, a service slot is S times as small as a time slot. The factor, S, represents the amount of resource speedup[7]. Since more than one cell can be forwarded to the same output port, the switches are combined input and output queued.

Since the crossbar can set up to S configurations per time slot, we are no longer limited to the convex combination of permutation matrices. In fact, the region of the set of rates over which the switch can transfer cells from input side to the output side is no longer the convex hull of permutation matrices, but a linear combination. The set of transfer rates, T, has the following form:

$$\begin{aligned} T &= \sum \phi_i P_i \\ S &= \sum \phi_i \end{aligned} \tag{2.25}$$

[7]We assume that S can also be a non-integer number.

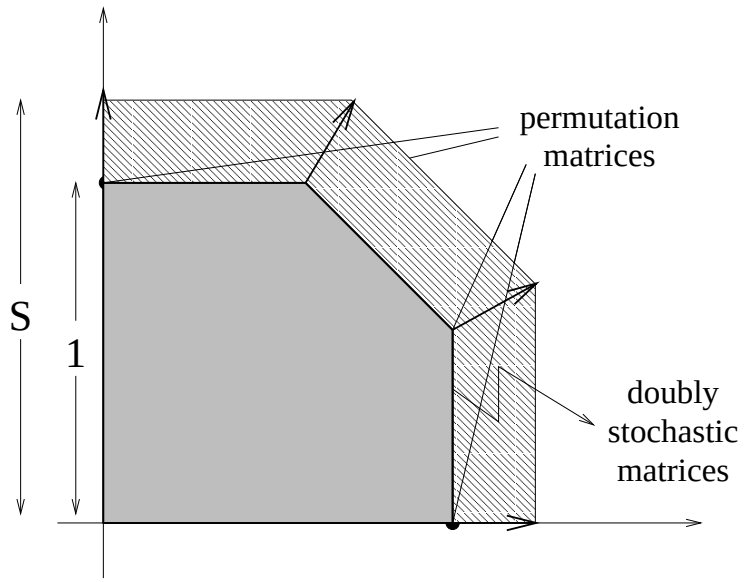

Figure 2-4: The extreme points are expanded by a factor of S. Hence, their convex combination is a superset of the set of doubly stochastic matrices which constitude the set of admissible rates and it is unchanged.

Note, however, that the region of the rates for the admissible flows is still the convex hull of permutation matrices since the capacities of the input and output links are unchanged (1 cell per time slot). The support set is, thus, a superset of the set of admissible rates (i.e., doubly stochastic matrices) as illustraetd in Fig. 2-4. Therefore, we may transfer cells through the crossbar at a higher rate than they actually arrive. In this section we show how to divide this extra rate over the I-O pairs of a switch.

If we distribute the extra resource uniformly over each pair, i.e., if we choose $T = SR$, the worst case service lag and thus the worst case delay decreases proportional to S. Hence, the service lag would still be $O(N^2)$, even though an improvement is observed for all the I-O pairs. But, the worst case service lag is not uniform over all the flows, and therefore, we do not want the improvement to be uniform. Thus, we should not assign the extra resource uniformly. The following fact, presented in [28] and [15], gives us insight on what is a good way of distributing the extra resource.

If there exists an integer f such that the matrix fR contains all integer-valued elements, then Birkhoff's decomposition terminates in at most f steps. This is illustrated for $f = 2$ in the following

example:

$$\begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \\ 1/2 & 1/2 & 0 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

We will talk about the performance implications of this fact later. Next, we present the main theorem of this chapter and its proof. We also give the rate quantization algorithm which will be used extensively in this thesis.

Theorem 2.4 (Main) *Let R be an $N \times N$ doubly stochastic matrix and s be a rational number which can be written as $\frac{1}{f}$ where f is an integer. There exists a (doubly super-stochastic) matrix, $Q = R' + U$, where R' is a doubly stochastic matrix with all the entries integer multiples of s, $U_{ij} = s$ and $Q_{ij} \geqslant R_{ij}$, $\forall 1 \leqslant i, j \leqslant N$. Thus, all rows and columns of Q sum up to $S = 1 + sN$.*

To prove this theorem, we will introduce an algorithm which constructs the doubly stochastic matrix, R', (and thus the matrix Q) for a given R and prove its correctness. Before we give the algorithm, we give an example which illustrates the theorem, and at the same time gives us an intuition on how to build our algorithm. Consider the following 3×3 doubly stochastic matrix.

Example 2.1

$$\begin{aligned} R \quad &= \quad \begin{bmatrix} 0.48 & 0.35 & 0.17 \\ 0.29 & 0.49 & 0.22 \\ 0.23 & 0.16 & 0.61 \end{bmatrix} \\ &\xrightarrow{s=0.1} \underbrace{\begin{bmatrix} 0.5 & 0.4 & 0.2 \\ 0.3 & 0.5 & 0.3 \\ 0.3 & 0.2 & 0.7 \end{bmatrix}}_{quantized} \begin{matrix} \rightarrow & 1.1 \\ \rightarrow & 1.1 \\ \rightarrow & 1.2 \end{matrix} \qquad (2.26) \\ &\longrightarrow \underbrace{\begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.3 & 0.1 & 0.6 \end{bmatrix}}_{doubly\ stochastic} + \underbrace{\begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}}_{uniform} \qquad (2.27) \end{aligned}$$

Our algorithm generates R' in two steps; in the first step, a matrix $\tilde{R}$ whose entries are integer

multiples of s is constructed and in the second step $\tilde{R}$ is modified to get R'. In the first step (2.26), every entry of the original matrix is increased by some non-zero amount, so that they all become integer multiples of s. Matrix $\tilde{R}$ is not necessarily some multiple of a doubly stochastic matrix. In the second step, some of the entries of $\tilde{R}$ are chosen and increased by another s in order to end up with a scaled doubly stochastic matrix. Equivalently, what we do is, increase every entry by s ($U_{ij} = s$ for all i and j) and choose the ones to be reduced by s. Indeed R' is a modification of $\tilde{R}$, where enough entries are reduced by s to make R' doubly stochastic. The challenging part of the algorithm is choosing which entries to reduce. To illustrate that this indeed is not a straightforward task, consider the above example and suppose we construct R' from $\tilde{R}$ starting with the first entry of the first row. Proceed with that row going through all the columns from left to right, reducing each entry by s if the sum of the entries of that column is greater than 1, until the first row sum becomes 1. Once the first row entries sum to 1, proceed with the second row and repeat the process. After completing the second row, we end up with the following matrix, whose third row is yet to be processed:

$$\begin{array}{ccccc} \left[\begin{array}{ccc} 0.4 & 0.4 & 0.2 \\ 0.3 & 0.4 & 0.3 \\ 0.3 & 0.2 & 0.7 \end{array}\right] & & \begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \end{array} & & \begin{array}{c} 1 \\ 1 \\ 1.2 \end{array} \\ \begin{array}{ccc} \downarrow & \downarrow & \downarrow \\ 1 & 1 & 1.2 \end{array} & & & & \end{array}$$

As we proceed with the third row, the only entry that can be reduced is the final one, 0.7, since all the other column sums are already 1. However, it has to be reduced by 0.2 for the resulting matrix to be doubly stochastic. If we do so, we end up with $Q_{33} = 0.6 < R_{33}$. Hence, we cannot choose the entries to be processed arbitrarily, and must be more careful in constructing Q.

Before we get to the algorithm, we will state a lemma that is fundamental to our algorithm.

Lemma 2.1 *Let x and y be two real numbers. Then $\exists \sigma \in \Re$, $0 < \sigma \leqslant y$ such that $x + \sigma$ is an integer multiple of y.*

Proof: Noting that

$$\sigma = y \left\lfloor \frac{x}{y} \right\rfloor + y - x$$

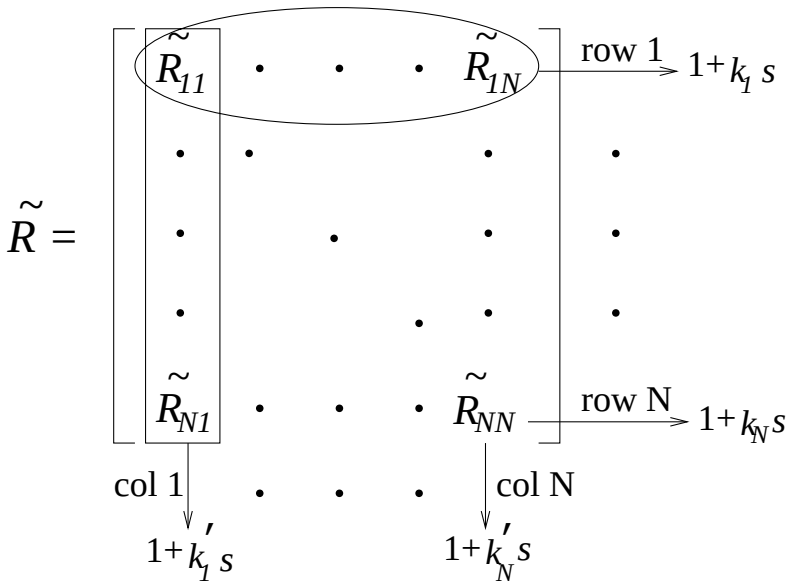

Figure 2-5: The $N \times N$ matrix $\tilde{R}$ is illustrated. Each row i and and column j sum to $1 + k_i s$ and $1 + k_j' s$ respectively where k_i snd k_j' are non-negative integers.

is in $(0, y]$ and $x + \sigma$ is an integer multiple of y, completes the proof.

Algorithm:

Step 1: Given any s, we know from Lemma 2.1 that there exists a σ_{ij}, $0 < \sigma_{ij} \leqslant s$ such that $R_{ij} + \sigma_{ij}$ is an integer multiple of s for all $1 \leqslant i, j \leqslant N$. Let σ be the matrix whose (i, j) entry is σ_{ij}. Define $\tilde{R} = R + \sigma$. All rows and columns of $\tilde{R}$ sum to integer multiples of s. By definition, 1 is also an integer multiple of s, and thus, as illustrated in Fig. 2-5, we can represent the sum of the entries of the ith row and the jth columnas $1 + k_i s$ and $1 + k_j' s$ respectively where k_i and k_j' are positive integers.

Step 2: In the second step, the algorithm scans $\tilde{R}$ row by row, starting with the row with maximum row sum $k_{\max}$, and determines whether the entry will remain unchanged or reduced by s before it is copied as the corresponding entry of the output matrix, R'. Each row is scanned starting from the entry with the largest column sum and continuing with entries of decreasing column sums. If both k_i and k_j' are positive for the current (i, j), that entry is reduced by s and otherwise it is copied directly as the corresponding entry of R'. A step by step description of the second part of the algorithm is as follows:

Initial Values: Let $i = \arg\max_{1 \leqslant l \leqslant N} k_l$ and $R' = \tilde{R}$; let k_m and k_n' be such that, $1 + s k_m$ and $1 + s k_n'$ are the mth row and nth column sum respectively, as illustrated in Fig. (2-5).

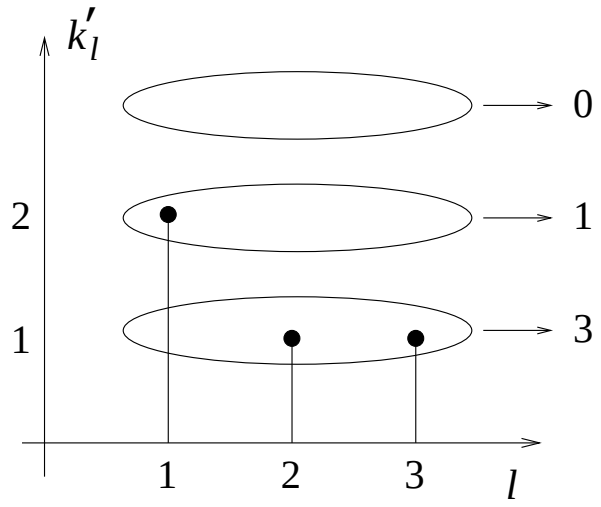

Figure 2-6: Vector $\vec{k'} = [2\ 1\ 1]^T$, thus $\vec{n'} = [3\ 1\ 0]^T$.

Repeat (1)-(2) until $k_i = 0$ for all $i \leqslant N$.

1. Set $E = \{1, \ldots, N\}$. Repeat (a)-(c) until $k_i = 0$.

   (a) $j = \arg\max_{j \in E} k'_j$

   (b) $R'_{ij} = \tilde{R}_{ij} - s,\ k_i \to k_i - 1,\ k'_j \to k'_j - 1,\ E \to E - \{j\}$.

2. $i = \arg\max_{1 \leqslant l \leqslant N} k_l$

The described algorithm reduces the elements of $\tilde{R}$ in the order of decreasing row sums. One might also randomize the procedure and work on a row randomly picked at every iteration as the processing order is unimportant. We give this modified algorithm and the proof of correctness for the modified algorithm in Appendix C.

Lemma 2.2 *The algorithm successfully terminates with a matrix R' which is doubly stochastic.*

Before we give the proof of the lemma, let us introduce some notation. Let

$$\begin{aligned} k_i &= \frac{1}{s}\left(\sum_{j=1}^{N}(R'_{ij}) - 1\right) \\ k'_j &= \frac{1}{s}\left(\sum_{i=1}^{N}(R'_{ij}) - 1\right) \end{aligned}$$

We can represent k_i and k'_j as an entry of the vectors $\vec{k}$ and $\vec{k'}$ respectively. Let $n'_i,\ i \geqslant 1$ be the number of columns j, for which $k'_j \geqslant i$. For example, if $\vec{k'} = [2\ 1\ 1]^T$, then $\vec{n'} = [3\ 1\ 0]^T$ as illustrated in Fig. 2-6.

Proof: By induction. We shall first show that initially

$$n'_1 \geqslant k_i \tag{2.28}$$

for all i, $1 \leqslant i \leqslant N$, Thus, for any $\vec{k}$ and for $i = \arg\max_{1 \leqslant l \leqslant N} k_l$ which is the first row to be processed, the algorithm will always be able to find sufficient entries to reduce (by s) to make the row sum equal to 1. We will prove a more general version of (2.28):

$$\vec{k} \prec \vec{n'} \tag{2.29}$$

namely, the vector $\vec{k}$ is majorized by the vector $\vec{n'}$. For the definition and some examples about majorization, see Appendix A.1.

First we prove that (2.29) holds at the beginning of the algorithm. Recall that $\sigma = \tilde{R} - R$. Hence,

$$\frac{1}{s}\sigma_{ij} \leqslant 1$$

$\forall i, j$. Let the lth column vector of σ be $\vec{v}_l$ and thus $v_{l,j} = \sigma_{jl}$ and $\langle \vec{v}_l, \vec{e} \rangle = k'_l s$, where $\vec{e} = [1 \cdots 1]^T$. From Kemperman's theorem (Appendix B.1), $\vec{v}_l$ is majorized by any vector for which k'_l entries are s, and the other $N - k'_l$ entries are 0. Hence,

$$\vec{v}_l \prec [\underbrace{s \cdots s}_{k'_l} \; \underbrace{0 \cdots 0}_{N-k'_l}]^T \equiv \vec{v}_l^{\max} \tag{2.30}$$

Thus, the vector on the right side of (2.30) is the *maximal vector* (in the sense of majorization) of the set of vectors whose entries are between 0 and s and $\langle \vec{v}, \vec{e} \rangle = k'_l s$. Let us denote the maximal vector of the lth column vector by $\vec{v}_l^{\max}$.

Now, let us define a new matrix, $\frac{1}{s}[\vec{v}_1^{\max} \cdots \vec{v}_N^{\max}]$, where each column is the maximal vector of the corresponding column of $\frac{1}{s}\sigma$. Note that the vector of column sums for this new matrix is $\vec{k'}$, and thus the corresponding distribution will be $\vec{n'}$; however, the row sums are not $\vec{k}$. Let the vector of row sums for our matrix be $\vec{k}_{\text{new}}$. Thus, $k_{\text{new},1}$ is the number of columns with $k'_j \geqslant 1$, i.e., n'_1; $k_{\text{new},2}$ is the number of columns with $k'_j \geqslant 2$, i.e., n'_2, and so on. More precisely, $k_{new,i}$ is

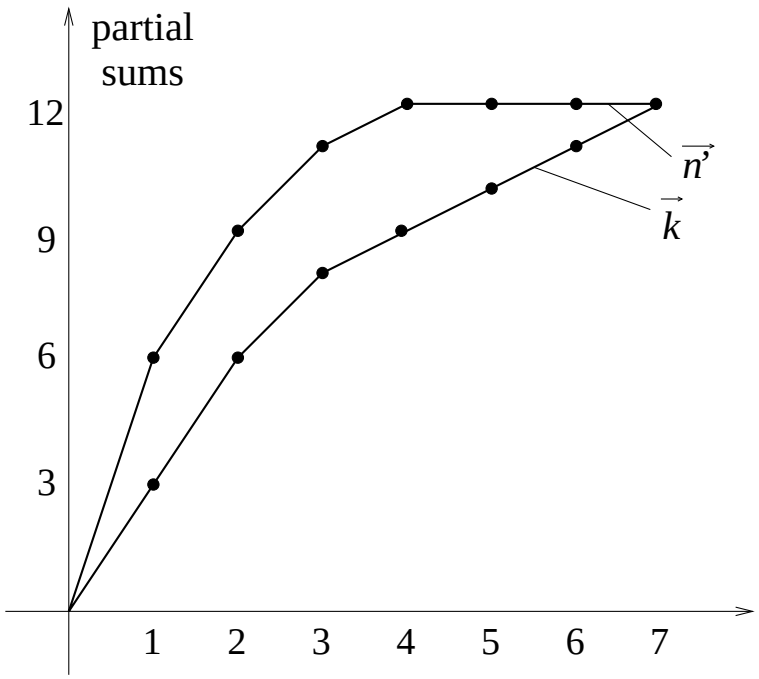

Figure 2-7: Sample Lorentz curves for $\vec{n'}$ and $\vec{k}$ are illustrated. Since $\vec{n'} \succ \vec{k}$ initially, the partial sum curve of $\vec{n'}$, is above that of $\vec{k}$.

the number of columns j, for which $k'_j \geqslant i$. Thus,

$$k_{new,i} = n'_i \tag{2.31}$$

But the vectors, $\vec{v}_l^{\max}$, $l \in \{1, \ldots, N\}$ are order symmetric (see Appendix A.2 for the definition). Hence we get the desired result using Day's theorem (Appendix B.2):

$$\begin{aligned} \vec{k}_{new} &= \vec{n}' = \frac{1}{s}\sum_{l=1}^{N} \vec{v}_l^{\max} && (2.32)\\ &\succ \frac{1}{s}\sum_{l=1}^{N} \vec{v}_l && (2.33)\\ &= \vec{k} && (2.34) \end{aligned}$$

We just showed that at the beginning of the algorithm, $\vec{n'} \succ \vec{k}$, and thus, $n_1 \geqslant k_i$, for all $i \leqslant N$. That is, the first step of the algorithm can be executed successfully to make the first row sum to 1. The partial sums[8] of the two sequences are illustrated in Fig. 2-7. Such curves are called Lorentz

[8]The mth partial sum of a vector, $\vec{v}$, is defined to be $\sum_{j=1}^{m} v_j$. Recall that $\vec{v}^I \prec \vec{v}^{II}$ if every partial sum of $\vec{v}^{II}$ is

curves and if, for two vectors, $\vec{v}^I \prec \vec{v}^{II}$, then the partial sum curve for $\vec{v}^{II}$ will always be above that of $\vec{v}^I$.

Next, we will prove that a similar majorization relation holds at the beginning of every step of the algorithm. We will use induction as follows. We have shown that $\vec{k} \prec \vec{n}'$ at the beginning of the first step. We now assume that it holds at the beginning of the ith step, $1 \leqslant i \leqslant N-1$ and show that it still holds at the end of the ith step. As a byproduct, we also show that the algorithm can successfully complete each step.

Suppose, the algorithm successfully constructed the first i rows of R'. We will show that (2.29) still holds at the beginning of the $(i+1)$st step, and the corresponding row of R' can be formed successfully.

First, let us focus on the two vectors, $\vec{n'}$ and $\vec{k}$ at the beginning of step i. At this point, $k_{M(1)}, \ldots, k_{M(i-1)} = 0$ where $M(q)$ is the qth entry in decreasing order from the largest in $\vec{k}$ at the beginning of tha algorithm (before any row is processed). The sum of the entries of the row that is currently being processed is $k_{M(i)}$. By the induction hypothesis, we assume $\vec{k} \prec \vec{n'}$; therefore, there should be as many 0s in vector $\vec{n'}$ as there are in $\vec{k}$ (verified in Appendix A.3). Since there are at least $i-1$ 0s in $\vec{k}$, we have $n'_{N-i+2}, \ldots, n'_N = 0$. At the beginning of the ith step, the entries of $\vec{n'}$ and $\vec{k}_{\downarrow}$ (defined in Appendix A.1 as the decreasing rearrangement of the entries of $\vec{k}$) can be listed as follows:

$$\begin{array}{cccccccc} n'_1 & \cdots & n'_{r-1} & n'_r & n'_{r+1} & \cdots & \overbrace{0 \cdots 0}^{i-1} \\ k_{M(i)} & \cdots & k_{M(i+r-2)} & k_{M(i+r-1)} & k_{M(i+r)} & \cdots & \underbrace{0 \cdots 0}_{i-1} \end{array}$$

Since $\vec{k} \prec \vec{n'}$, there exists at least one entry in $\vec{n'}$ which is greater than or equal to $k_{M(i)}$. Let the smallest such entry be n'_r.

Lemma 2.3 *At the end of ith step, the only change in $\vec{n'}$ is that the entries n'_r and n'_{r+1} will be replaced with $\left[n'_{r+1} + \left(n'_r - k_{M(i)}\right)\right]$ and a 0.*

Proof: These two changes can be explained as follows. The algorithm will look into the current R' for the column with an entry which has not yet been reduced in step i and which has the maximum column sum, and reduce it by s. Suppose this maximum column sum is ms for some $m \in \mathbb{Z}^+$. This

at least as great as that of $\vec{v}^I$

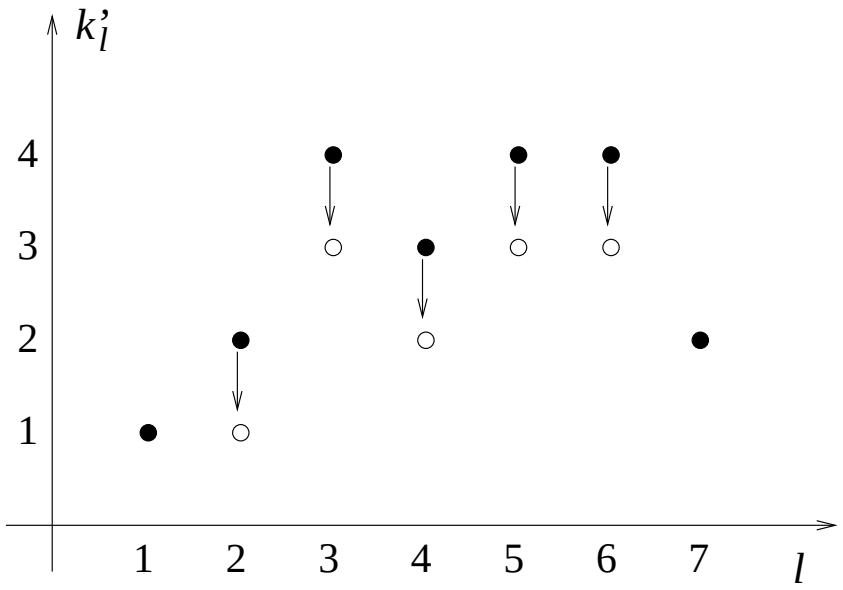

Figure 2-8: If $k_i = 5$, the entries of the ith row that are decreased are illustrated above.

operation will reduce the number of columns, j, such that $k'_j = m$ by 1. Thus, the only change in $\vec{n'}$ will be in the smallest non-zero entry, n'_m, which will decrease by 1. If that entry is greater than 1, then there were multiple entries with the maximum column sum. The algorithm continues with these other entries. Hence, if the original value of $k_{M(i)}$ is greater than n'_m, then after processing n'_m entries, n'_m will become 0 and $k_{M(i)} - n'_m$ entries will be left to be decreased at the row currently being processed. The algorithm will go on with the entries that have not been reduced before and with highest possible column sums. At this stage, the new value, $\hat{n}'_m$, of n'_m is 0 and the new value, $\hat{k}_{M(i)}$ of $k_{M(i)}$ is $k_{M(i)} - n'_m$. Note that n'_m potential entries have already been processed, and if $k_{M(i)}$ is greater than the second largest entry, n'_{m-1}, of $\vec{n'}$ then n'_{m-1} will be reduced to $\hat{n}'_{m-1} = n'_m$ but no further beyond that, since n'_m potential entries have already been processed. Similarly, each entry of $\vec{n}'$, which is smaller than $k_{M(i)}$ will be replaced with the next entry in order. Finally, the first entry, n'_r, in $\vec{n'}$ that is greater than $k_{M(i)}$ will be reduced by only $n'_r - k_{M(i)}$. Hence, after the ith row is processed, $\vec{n'}$ will have a 0 replacing n'_r, and a $\left[n'_{r+1} + \left(n'_r - k_{M(i)}\right)\right]$ replacing n'_{r+1}. Note that at the end of the ith step, $\vec{k}_{\downarrow}$ will be the same except $k_{M(i)}$ will be replaced with a 0.

This process is illustrated in Fig. 2-8 assuming $\vec{k'} = [1\ 2\ 4\ 3\ 4\ 4\ 2]$, i.e., $\vec{n'} = [7\ 6\ 4\ 3\ 0\ 0\ 0]$ at the beginning of step i. If $k_{M(i)} = 5$, then at the end of step i, $\vec{n'} = [7\ 5\ 3\ 0\ 0\ 0\ 0]$. Notice that 6 is the smallest entry in $\vec{n'}$ greater than or equal to $k_{M(i)} = 5$. Hence, 6 and 4 are changed to $6 + (4 - 5) = 5$ and 0 respectively.

Now, we show that, $\vec{k} \prec \vec{n}'$ at the end of the ith step of the algorithm. But before that we

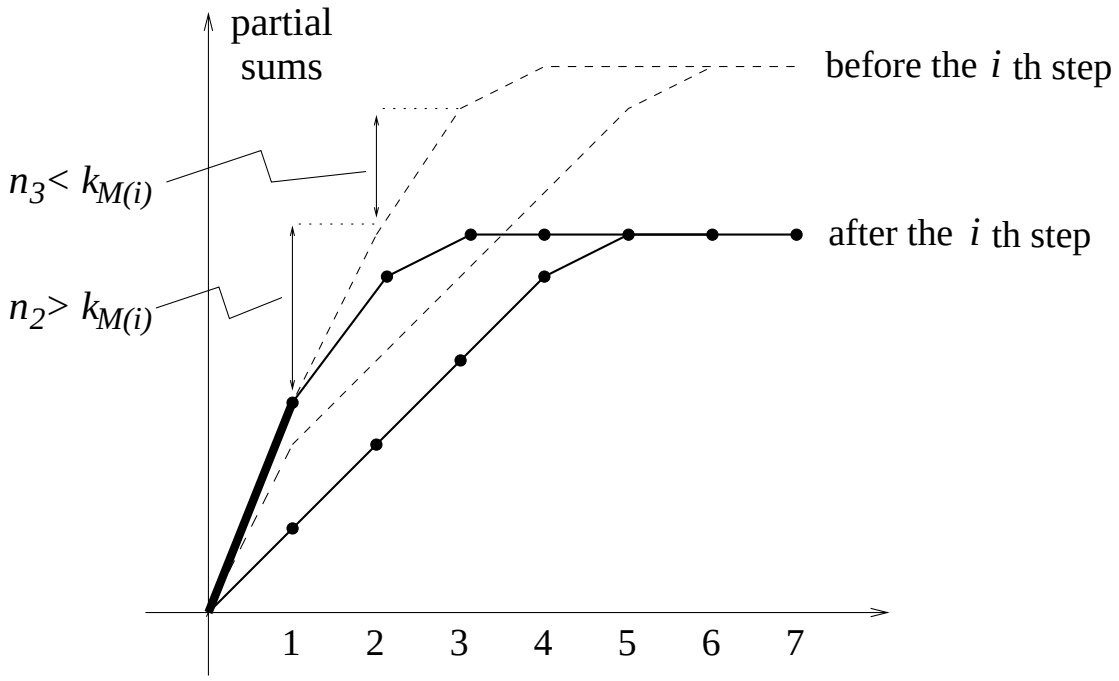

Figure 2-9: In the ith step, $k_{M(i)}$ is removed (replaced with a 0) and the smallest entry of $\vec{n}'$ greater than or equal to $k_{M(i)}$ is reduced by $k_{M(i)} - n'_{r+1}$, and the following entry, n'_{r+1} is removed (replaced with a 0). The dashed curve is the initial curve, and the solid one is the one at the end of the ith step. The bold segment is the one which do not change. The distance between the two curves does not decrease at all.

present a graphical illustration of what happens in the ith step. The Lorentz curves of $\vec{k}$ and $\vec{n}'$ are illustrated in Figures 2-9. The entry, $k_{M(i)}$ is removed from $\vec{k}$. The new Lorentz curve for $\vec{k}$ can be sketched from the old one by just removing the first segment segment of the curve and attaching the rest of the curve to the origin as illustrated in the figure. The new Lorentz curve for $\vec{n}$ can similarly be sketched with some modification to the old one. The algorithm will find the segment with the smallest increment greater than $k_{M(i)}$. Then, it will reduce this increment by $k_{M(i)} - n'_r$, remove n'_r, and attach the two separate parts. The two Lorentz curves intersect at 0 and at $\sum_l n'_l = \sum_l k_l$. Initially, these are the only two points they intersect, and the curve for $\vec{n}'$ is always above the curve for $\vec{k}$, otherwise. We need to show that this is the case after the ith step. This can be easily observed from Fig. 2-9. Since the removed segment in $\vec{k}$ is to the left of the reduced segment of $\vec{n}'$, the distance between the two curves will only increase in between these modified segments, and remain the same outside this region at the end of the ith step. We can prove this statement as follows. There are two regions we need to consider as shown in the following table:

n'_1	$\cdots$	n'_r	n'_{r+1}	n'_{r+2}	n'_{r+3}	$\cdots$	$\overbrace{0\cdots0}^{i-1}$
$k_{M(i)}$	$\cdots$	$k_{M(i+r-1)}$	$k_{M(i+r)}$	$k_{M(i+r+1)}$	$k_{M(i+r+2)}$	$\cdots$	$\underbrace{0\cdots0}_{i-1}$
		I					**II**

At the end of the ith step, the partial sums of the two sequences are as follows. In region I, at the end of the ith step, $k_{M(i)}$ will be replaced with a 0 and it will no longer be in the second region. All the entries of $\vec{n}'$ will be unchanged up to n'_r. Thus, the partial sums will change in favor of $\vec{n'}$ by an extra $k_{M(i)}$ from the beginning all the way down to n'_r. This entry is replaced with $\left[n'_{r+1} + \left(n'_r - k_{M(i)}\right)\right]$, and the next entry, n'_{r+1} will be replaced with a 0 and removed from the second region. The total decrease in the partial sums of $\vec{n'}$ in the first region is $k_{M(i)}$. The extra $k_{M(i)}$ gained in favor of $\vec{n'}$ earlier by the removal of $k_{M(i)}$ from vector $\vec{k}$ is good enough to make up for this loss of $\vec{n}'$. The second region for both $\vec{n'}$ and $\vec{k}$ are expanded similarly, with the addition of a 0. This will not affect the partial sums, and hence the majorization is preserved.

Thus, we proved that at the beginning of each step, (2.29) holds and $n'_1 \geqslant k_{M(i)}$, for all $i \leqslant N$. Therefore, the algorithm will always be able to find the desired number of entries to reduce, and

at the end of the algorithm, $k_i = 0$, for all $i \leqslant N$. But, since

$$0 = \sum_{i=1}^{N} k_i = \sum_{j=1}^{N} k'_j \tag{2.35}$$

and $k'_j \geqslant 0$, for all $j \leqslant N$, it is also true that $k'_j = 0$, for all $j \leqslant N$ completing the proof.

Lemma 2.4 *Every entry of R' is an integer multiple of s.*

Proof: The input matrix, $\tilde{R}$, of the algorithm already has all the entries integer multiples of s. We complete the proof noting that the change in each entry from $\tilde{R}$ ro R' is an integer multiple of s (either reduced by s or left unchanged).

Lemma 2.5 *Every entry of R' is at least as great as its counter part in R decreased by s:*

$$R'_{ij} \geqslant R_{ij} - s, \quad 1 \leqslant i, j \leqslant N \tag{2.36}$$

Proof: Note that

$$\tilde{R}_{ij} \geqslant R_{ij}, \quad 1 \leqslant i, j \leqslant N \tag{2.37}$$

Since the algorithm reduces every entry by at most s,

$$R'_{ij} \geqslant \tilde{R}_{ij} - s, \quad 1 \leqslant i, j \leqslant N \tag{2.38}$$

Inequality (2.36) is immediate by (2.37) and (2.38).

Putting all three Lemmas 2.2, 2.4 and 2.5, together, we completed the proof of correctness for the rate quantization algorithm, and thus our main theorem is also proved. In Appendix C we prove Lemma 2.2 holds even if the algorithm processes the rows of matrix $\tilde{R}$ in an arbitrary order, rather than processing the one with the maximum row sum in each step.

2.3 Performance with Rate Quantization

In the previous section, we presented the rate quantization algorithm. Like quantization in the context of data compression, regions are specified in some intial set, and a representation point is assigned to each region. The initial set in our problem is the set of doubly stochastic matrices.

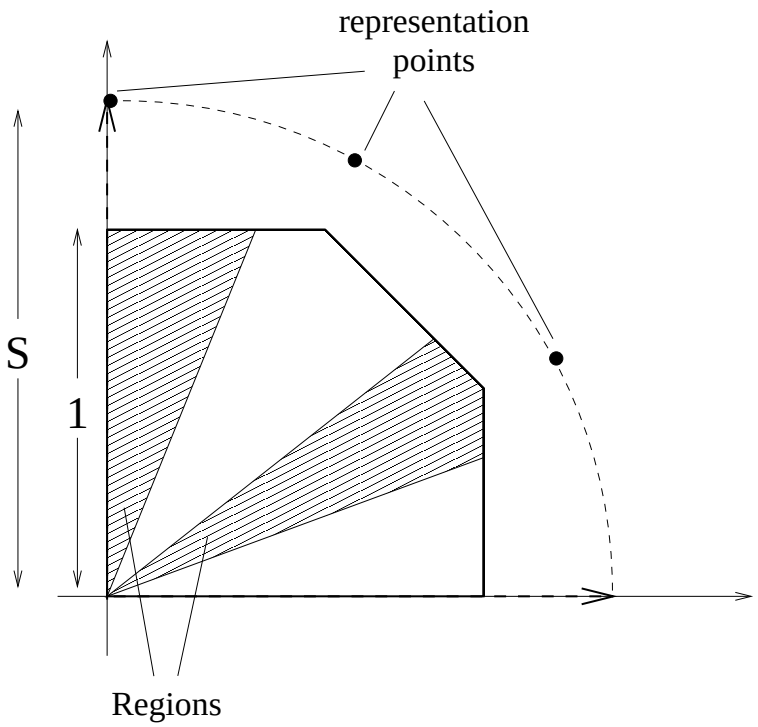

Figure 2-10: Regions and representation points are illustrated. Representation points are outside the set of doubly stochastic matrices.

Unlike the typical practice in data compression, the representation point of a region is outside the region, and even outside of the initial set, since we increase every entry of a matrix to find its representation point. Rate quantization is thus a mapping of the contiuum of rates into a discrete set of points. A set of regions and representation points is illustrated in Fig. 2-10. Note that this picture is by no means an exact description of what happens; it is an illustration of some of the ideas.

In this section, we study the impacts of rate quantization on the performance of an input queued switch. We show that rate quantization cuts the service lag by a factor of $\mathrm{O}(N)$ and we illustrate that satisfactory performance can be attained even with small speedup. For a given speedup, there exists a certain time period, T, for which a contract, (R, T) is perfectly supportable for any admissible rate matrix, R. We also explore the complexity of the entire process, from quantization to decomposition, and show that it can be reduced significantly with deterministic scheduling.

2.3.1 Performance Analysis with Speedup

Recall that, given a doubly stochastic matrix, R, and the parameter s, our algorithm generates a doubly super-stochastic matrix Q such that $Q_{ij} \geqslant R_{ij}\ \forall i, j$. Suppose R is the rate matrix for a contract for which the switch is to provide service. Supporting the matrix Q would be sufficient for supporting R. However, if we write Q as a positive linear combination of permutation matrices, the coefficients sum to $1+sN$ where $N \times N$ is the size of Q. Since each permutation matrix corresponds to a crossbar configuration, the crossbar must serve $1+sN$ permutations per time slot, i.e., it must transfer $1+sN$ cells per time slot[9]. Hence, a service slot is $(1+sN)^{-1}$ of a time slot. We show that a speedup (for the crossbar and the buffers) of $1+sN$ is sufficient for supporting any admissible rate.

Let us examine the decomposition for Q. We showed in the last section that Q can be written as a sum of a doubly stochastic matrix, R', with entries that are integer multiples of s, and a constant matrix, U. As shown in [15], if a doubly stochastic matrix is composed of entries all of which are integer multiples of s, then the decomposition terminates with at most

$\frac{1}{s}$ permutation matrices. Also, if we define E to be the constant matrix with all 1's, then,

$$U = sN\frac{1}{N}E$$

Note that $\frac{1}{N}E$ is a doubly stochastic matrix and it can be written as a convex combination of N permutation matrices. As a result, Q can be written as a linear combination of at most $\frac{1}{s}+N$ permutation matrices:

$$Q = \sum_{i=1}^{\frac{1}{s}+N} \phi_i P_i, \quad \sum \phi_i = 1 + sN \tag{2.39}$$

Corollary 2.1 *All the coefficients, $\{\phi_i\}$ of the decomposition, (2.39) are integer multiples of s.*

Proof: When all the entries of Q are integer multiples of s, the first permutation matrix found by the algorithm has a coefficient which is also an exact multiple of s. To begin the second step, some the entries of the matrix either remain the same or are decreased by the coefficient of the first permutation matrix. In particular, those entries, (i, j) for which $P_{1,ij} = 1$ are decreased. Hence, at

[9]Note, however that $1 + sN$ does not need to be an integer, it is just a factor by which the switch operates faster than line rates. Namely, a service slot is $(1 + sN)^{-1}$ of a time slot. For instance, if $1 + sN = 1.5$, then three cells every two time slots can be transferred from the input to the output of the switch.

the beginning of the second step all entries are still integer multiples of s. Similarly, if the matrix has entries all of which are integer multiples of s at the beginning of any step, the coefficient of the permutation matrix found at that step is also an exact multiple of s. Thus, by induction, the algorithm shall terminate with coefficients all of which are exact multiples of s.

Suppose at this point that the permutations are scheduled using a regular PGPS scheduler. Note that the scheduler will operate similarly, but it will schedule $1+sN$ permutation matrices per time slot. Since this number is not necessarily an integer, it is easier to visualize the scheduler as scheduling a frame of $\frac{1}{s}+N$ permutation matrices in a time period of length $\frac{1}{s}$ slots. The PGPS scheduler will similarly give a service opportunity to the permutation matrix with the smallest finishing time in the corresponding GPS system.

Theorem 2.5 *If the weight of each permutation in a PGPS scheduler is an integer multiple of s where s is the reciprocal of an integer, then the schedule is periodic with $\frac{1}{s}$ time slots and for all $t=\frac{k}{s},\ k\in\mathbb{Z}^+$,*

$$D_i(t)=\phi_i t,\quad \forall i,j \tag{2.40}$$

where D_i and ϕ_i are, respectively, the service opportunities given to, and the weight of, permutation i.

Proof: At time $t=0$, (2.40) holds. Suppose it holds at $t=\frac{k}{s}$ for some $k\in\mathbb{Z}$. If we show that it holds at $t=\frac{k+1}{s}$, then, by induction the proof will be complete. For a permutation i, the induction hypothesis is:

$$D_i\left(\frac{k}{s}\right)=\phi_i\frac{k}{s} \tag{2.41}$$

We will show that Eq. (2.41) also holds when k is replaced with $k+1$ by contradiction. Suppose

$$D_i\left(\frac{k+1}{s}\right)>\frac{k+1}{s}\phi_i$$

Since

$$\sum_m \phi_m = 1+sN$$

and

$$\sum_m D_m\left(\frac{k+1}{s}\right) = \frac{k+1}{s}(1+sN)$$

there exists a permutation, j for which

$$D_j\left(\frac{k+1}{s}\right) < \frac{k+1}{s}\phi_j$$

But, the GPS finishing time for the $\left(\frac{k+1}{s}\phi_i\right)$th service opportunity of permutation i and the finishing time for the $\left(\frac{k+1}{s}\phi_j\right)$th service opportunity of permutation j are identical:

$$FT_i^{(k+1)\frac{\phi_i}{s}} = FT_i^{(k+1)\frac{\phi_j}{s}} = \frac{k+1}{s}$$

Thus, the PGPS server cannot schedule the $\left[(k+1)\frac{\phi_i}{s}+1\right]$st service opportunity for permutation i before the $\left[(k+1)\frac{\phi_j}{s}\right]$th service opportunity of permutation j, hence the contradiction. Similarly, a contradiction can be reached in the reverse case, i.e.,

$$D_i\left(\frac{k+1}{s}\right) < \frac{k+1}{s}\phi_i$$

and the proof is complete.

Next, consider a simpler scheduler which schedules crossbar configurations simultaneously as the decomposition is run as follows. As soon as the first permutation matrix, P_1 and its coefficient , ϕ_1 are found by the algorithm, the corresponding crossbar configuration is set and kept for $\frac{\phi_1}{s}$ service slots. Then, the second permutation matrix, P_2 is found and the corresponding crossbar configuration is kept for $\frac{\phi_2}{s}$ service slots, and so on. Since the permutations are scheduled according to the order they are generated by the decomposition, rather than being scheduled according to an order determined by their coefficients, and the pemutations can be generated in any order, we will call the scheduler a *arbitrary order scheduler* (AOS).

First of all, the schedule generated by an AOS is also periodic with period $\frac{1}{s}+N$ permutation matrices ($\frac{1}{s}$ time slots) just like PGPS. It was shown in the proof of Theorem 2.5 that within a typical frame of a PGPS schedule, P_i is scheduled $\frac{\phi_i}{s}$ times. By definition, with AOS, the number of times P_i is in effect within the same frame is $\frac{\phi_i}{s}$ as well. Thus, the frequency of each permutation in a frame is exactly the same as that with PGPS, and (2.40) holds for all $t = \frac{k}{s}$, $k \in Z^+$ with

AOS.

Next, unlike PGPS, the decomposition process need not be complete before the switch starts being configured. Indeed, the two processes can be run simultaneously and no extra delay is experienced for the decomposition.

We showed that the number of service opportunities provided to an I-O pair at times $\frac{k}{s}$, $k \in \mathbb{Z}$ is identical to the desired service by each user. Next, we analyze what happens in between these points and derive an upper bound on the service lag for AOS. Because of the periodic nature of the services, it suffices to look at the first period, $\left[0, \frac{1}{s}\right]$. Recall that $\frac{1}{s} + N$ crossbar connections are scheduled every $\frac{1}{s}$ time slots. Since,

$$\text{service slot} = \frac{\text{time slot}}{\text{speedup}} \tag{2.42}$$

the provided service is periodic with a period $\frac{1}{s} + N$ service slots. Let us define the number of service opportunities, $D_{ij}(t)$ for an I-O pair (i, j) as the number of service opportunities given to the permutation matrices whose (i, j) entry is 1 by time t. In the following theorem, we derive an upper and a lower bound on the service lag experienced by an I-O pair.

Theorem 2.6 *Let R_{ij} be the desired rate and $D_{ij}(t)$ be the number of service opportunities for the I-O pair (i, j) by time t. The following holds for the service lag with rate quantization along with a speedup of $1 + sN$ over the AOS:*

$$-L(s) \leqslant D_{ij}(t) - R_{ij}t \leqslant U(s) \tag{2.43}$$

where t is in time slots and

$$L(s) = \begin{cases} \frac{1}{4s} + \frac{N}{4} & , \ s \leqslant \frac{1}{N} \\ \\ \frac{N}{1+sN} & , \ s > \frac{1}{N} \end{cases} \tag{2.44}$$

and

$$U(s) = \begin{cases} \left(\frac{1}{4s}\right)\left[\frac{(1+s(N+2))^2}{1+sN}\right] & , \ s \leqslant \frac{1}{N-2} \\ \\ \frac{N}{1+sN}(1+2s) & , \ s > \frac{1}{N-2} \end{cases} \tag{2.45}$$

Figure 2-11: A schedule for the I-O pair (i, j). The pair gets a service opportunity at the end of each frame.

In fact, the upper and the lower bounds we derive are essentially the same phenomenon. Consider the schedule given in Fig. 2-11 for some I-O pair, (i, j). Suppose this pair gets service opportunities in the regions as shown in the figure. If we define $D_{ij}[0, t)$ to be the number of service opportunities given to I-O pair (i, j) in the period $[0, t)$, then

$$\begin{aligned} D_{ij}(0, \tau] &= 0 \\ &= R_{ij}\tau - R_{ij}\tau \end{aligned}$$

On the other hand,

$$\begin{aligned} D_{ij}(\tau, \frac{1}{s}] &= \left(\frac{1}{s} - \tau\right)(1 + sN) \\ &= R_{ij}\left(\frac{1}{s} - \tau\right) + (1 + sN - R_{ij})\left(\frac{1}{s} - \tau\right) \end{aligned}$$

Hence, service lag and service *lead* are dual, and a service lag may turn into a service lead depending on what point is taken as the origin.

Proof of Theorem 2.6: Recall that, given the quantization parameter, s, the rate quantization algorithm generates the Q matrix for every doubly stochastic matrix R such that the corresponding entries of R and Q are within $2s$ of each other:

$$R_{ij} < Q_{ij} \leqslant R_{ij} + 2s, \quad \forall i, j \tag{2.46}$$

since each entry is not increased more than $2s$ in the quantization process.

1. *Lower bound*: Cells with the I-O pair (i, j) can be served through multiple permutation matrices and in the worst case it can get no service in the first $(1 - Q_{ij})\frac{1}{s} + N$ service slots and given service in all of the final $Q_{ij}\frac{1}{s}$ service slots of a frame as illustrated in Fig. 2-12. This happens with AOS if the final $Q_{ij}\frac{1}{s}$ permutation matrices have $P_{\cdot,ij} = 1$ and all the others have a 0 in this location. The desired and provided service curves are illustrated in

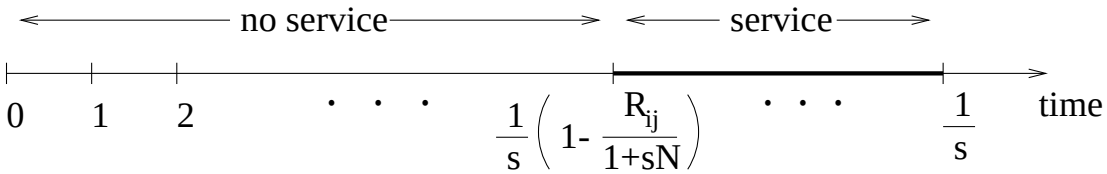

Figure 2-12: In the worst case scenario, the I-O pair (i,j) can be served through $R_{ij}\frac{1}{s}$ permutation matrices, each of which has a weight of s and can be delayed by as many as $\left(1-\frac{R_{ij}}{1+sN}\right)\frac{1}{s}$ time slots before getting the first service opportunity. This corresponds to $(1+sN-R_{ij})\frac{1}{s}$ service slots.

Fig. 2-13 for this scenario. Thus, user (i,j) can possibly delayed by no more than

$$\begin{aligned}\left[(1-Q_{ij})\frac{1}{s}+N\right]\frac{\frac{1}{s}}{\frac{1}{s}+N} &= \frac{1}{s}\left(1-\frac{Q_{ij}}{1+sN}\right)\\ &< \frac{1}{s}\left(1-\frac{R_{ij}}{1+sN}\right)\end{aligned}$$

time slots, since $Q_{ij} > R_{ij}$. Therefore,

$$D_{ij}(t) \geqslant R_{ij}t - \frac{1}{s}\left(1-\frac{R_{ij}}{1+sN}\right)R_{ij} \tag{2.47}$$

Hence, the bound on the service lag varies with the desired rate. If we solve the constrained optimization problem to minimize the lower bound over all possible rates:

$$\max_{R_{ij}\in[0,1]} \frac{1}{s}\left(1-\frac{R_{ij}}{1+sN}\right)R_{ij}$$

we get,

$$R^*_{ij} = \min\left\{1, \frac{1+sN}{2}\right\}$$

and when we substitute this into our objective function, we get the lower bound, $L(s)$, of (2.44).

2. *Upper bound*: Similarly, at certain points in time, the number of provided service opportunities can exceed the desired amount. For instance, I-O pair (i,j) can get all its service opportunities in the first $R_{ij}\frac{1}{s}+2$ service slots of a frame, as illustrated in Fig. 2-14. The extra 2 time slots of service are due to (2.46). The desired and provided service curves are illustrated in Fig.

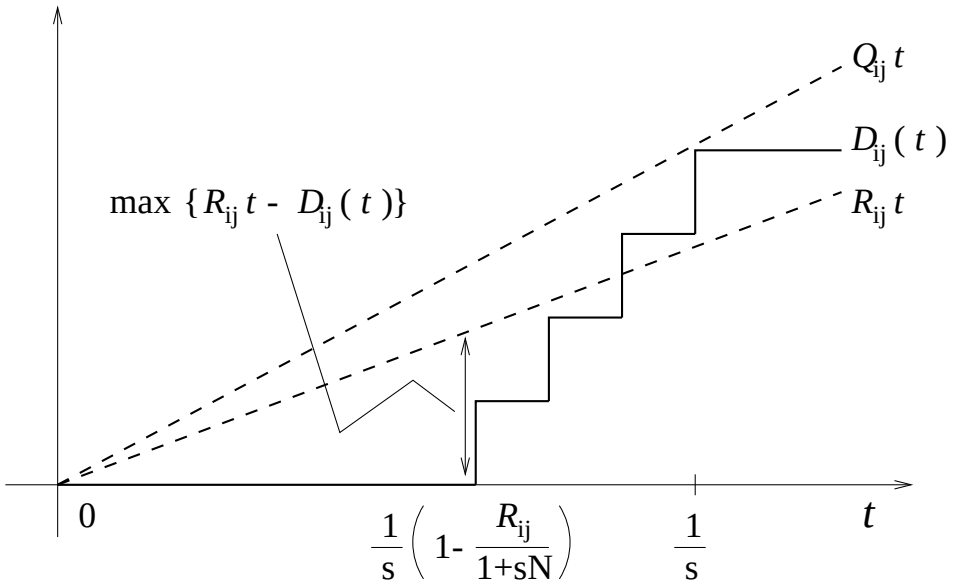

Figure 2-13: The service curves for the case illustrated in Fig. 2-12.

2-15 for this scenario. This delay can be converted into time slots using Eq. (2.42) as

$$\left(R_{ij}\frac{1}{s}+2\right)\frac{\frac{1}{s}}{\frac{1}{s}+N}=\left(\frac{1}{s}\right)\left(\frac{R+2s}{1+sN}\right)$$

and since I-O pair (i,j) gets $R_{ij}\frac{1}{s}+2$ service opportunities in this time, the "service lead" can be bounded as,

$$D_{ij}(t)\leqslant R_{ij}t+\left(\frac{1}{s}\right)\left(\frac{R+2s}{1+sN}\right)(1+sN-R_{ij}) \tag{2.48}$$

Similarly, solving the constrained optimization problem to maximize the upper bound over all possible rates:

$$\max_{R_{ij}\in[0,1]}\left(\frac{1}{s}\right)\left(\frac{R+2s}{1+sN}\right)(1+sN-R_{ij})$$

we get,

$$R_{ij}^{*}=\min\left\{\frac{1+s(N-2)}{2},1\right\}$$

Substituting this in our objective function, we get the upper bound of (2.43) completing the proof.

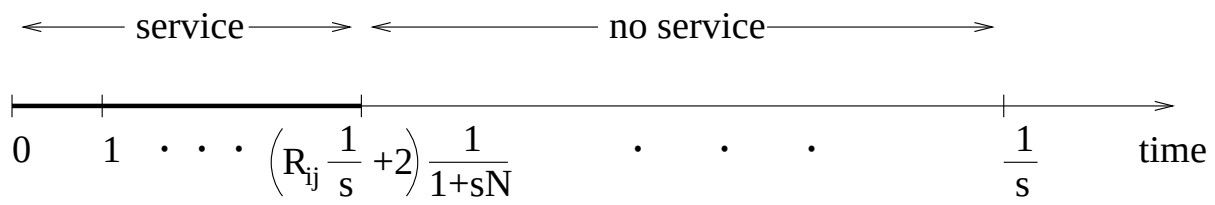

Figure 2-14: I-O pair (i, j) can be given service opportunities at the beginning of every period for $\frac{1}{s}\left(\frac{R_{ij}+2s}{1+sN}\right)$ time slots. This corresponds to $\frac{1}{s}(R_{ij}+2s)$ service slots.

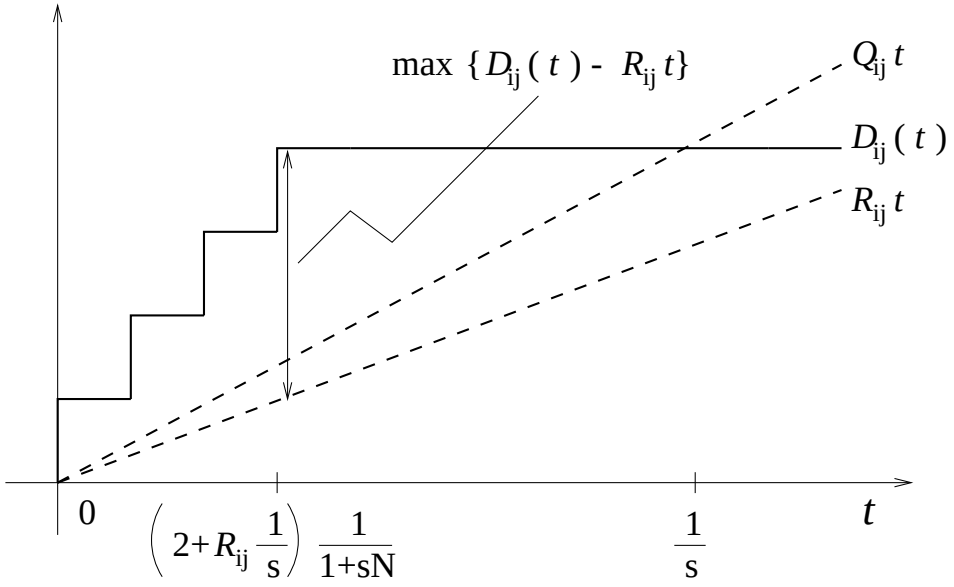

Figure 2-15: The service curves for the case illustrated in Fig. 2-14.

Note that the lower and upper bounds derived for AOS are also tight for the PGPS scheduler. Consider some input output pair, (i, j). Suppose, after rate quantization, $Q_{ij} \approx R_{ij}$ and all the permutation matrices that have $P_{\cdot,ij} = 1$ have a coefficient s in the decomposition. Since s is the minimum possible coefficient for a permutation matrix in a decomposition, it is indeed possible for all the matrices with $P_{\cdot,ij} = 1$ to get scheduled at the end of a frame of a PGPS schedule, and thus, the lower bound for the service lag is tight. Similarly, the I-O pair (i, j) may get all its service opportunities in the first $R_{ij}\frac{1}{s} + 2$ service slots of a frame, and hence, the upper bound is also tight.

2.3.2 Implications

Improved Service

For sufficiently large values of N, the upper bound given in (2.48) and the lower bound given in (2.47) are approximately the same and they increase linearly with N. The normalized maximum upper and the minimum lower bounds are sketched as a function of speedup in Fig. 2-16 for large values of N (there is only one curve since they converge eachother for large N). We will call these bounds the quality of service bounds. The actual upper and lower bounds are also given in Fig. 2-17 as a function of the pair, (speedup,rate). All the curves illustrate the bounds normalized with respect to the switch size, N. The ordinate should be multiplied by N for the actual service bound.

Note that speedup improves the provided service a great deal in the sense that the provided service follows the rate request much more closely when compared to the no speedup case. Indeed, for sufficiently large speedup, the service lag does not exceed N/speedup whereas it can go as high as $O(N^2)$ without speedup. For example with a speedup of 2, the deviation is no more than $\frac{N}{2}$ time slots. For a switch of size 256×256, this bound is 128 time slots; without speedup it can be as high as 65,000 time slots!

We can approximate the speedup necessary to support a given maximum delay as given in the following relation:

$$\text{speedup} \approx \frac{N \times \text{packet size}}{\text{link data rate} \times \text{delay}} \tag{2.49}$$

Note that the link data rate is in terms of bits/sec.

Example 2.2 *Suppose we have a* 128×128 *ATM switch whose links have a maximum capacity of 622 Mbps. The switch supports delay sensitive traffic and with a maximum delay of no more than*

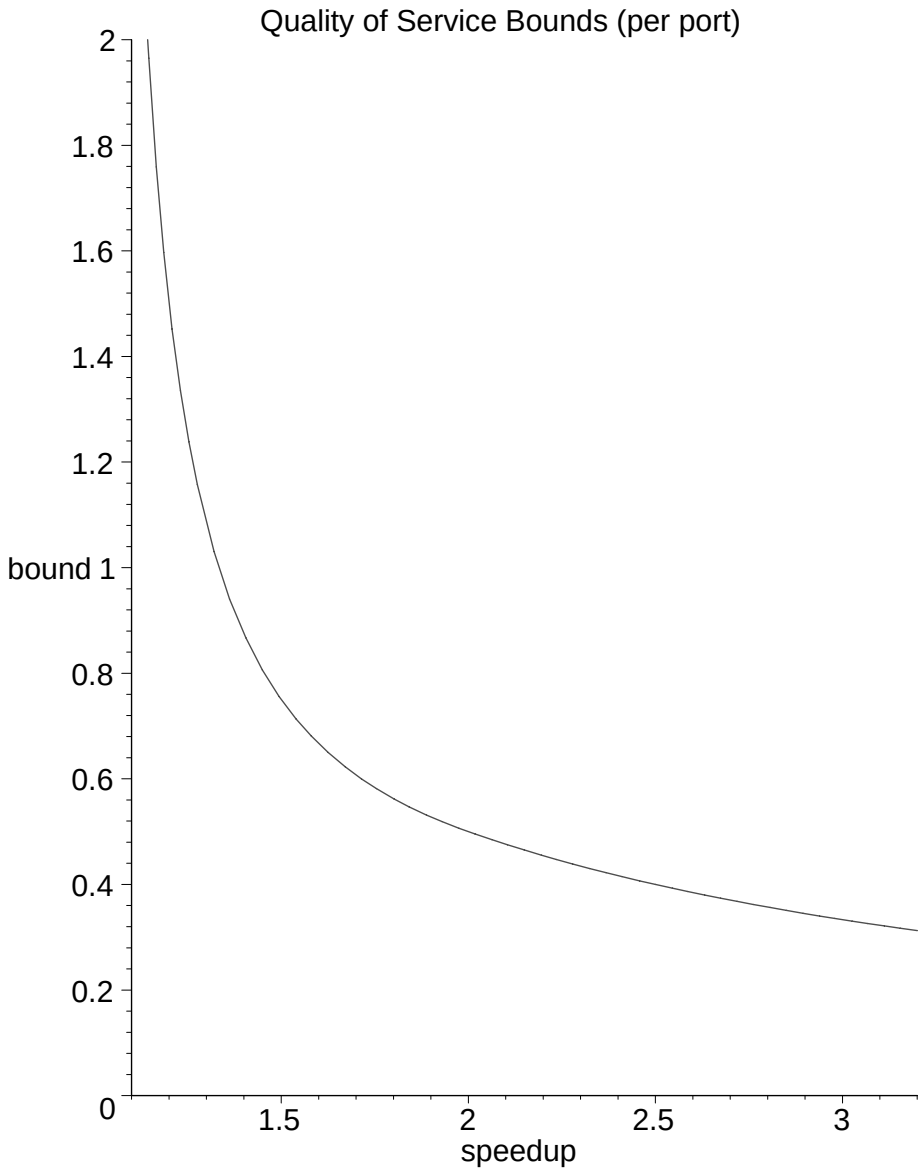

Figure 2-16: Illustration of the quality of worst case service bounds over all possible rates (given in (2.43)) as a function of the speedup for large N. Note that the origin corresponds to speedup $= 1$, hence no speedup. The ordinate should be multiplied by N for the actual service bound.

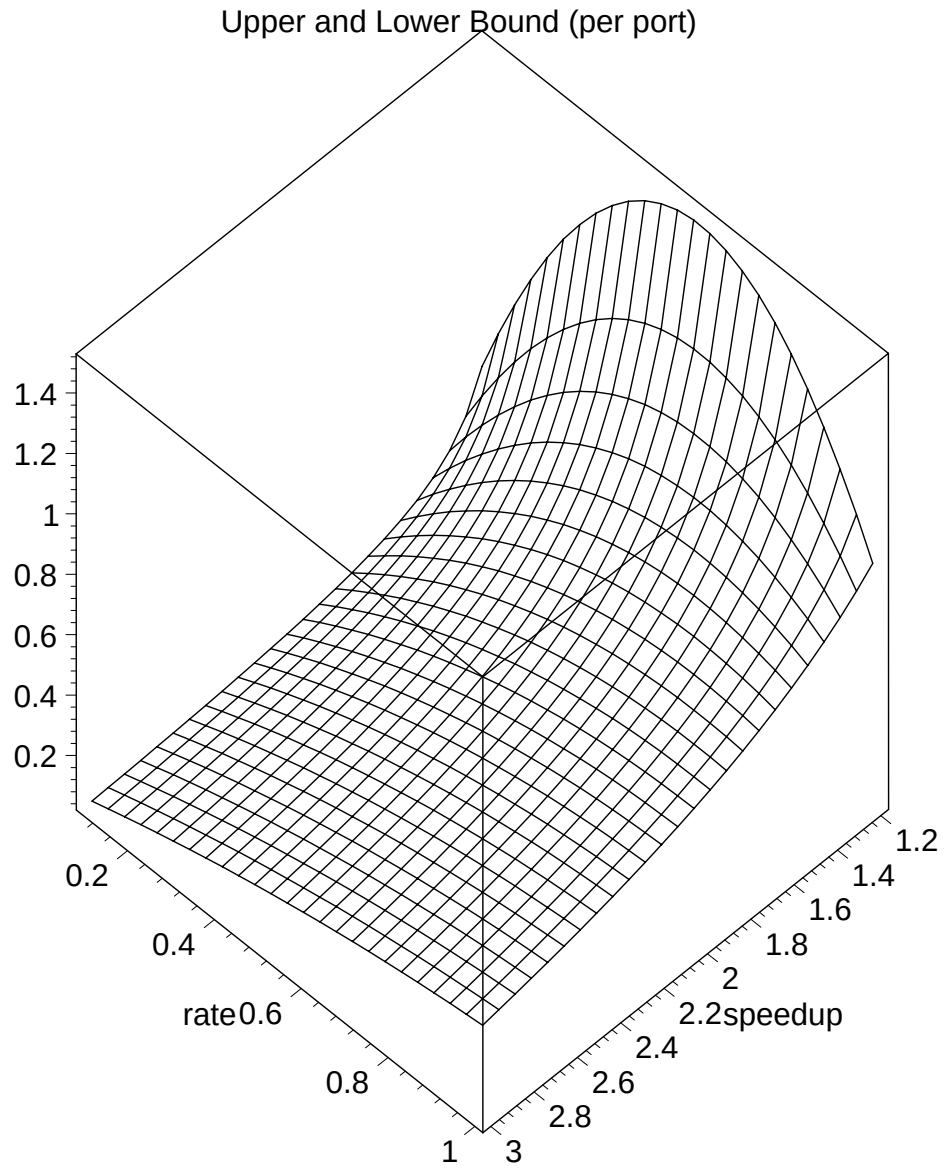

Figure 2-17: Illustration of the upper and the lower bound on the difference between provided and desired service (given in (2.47)) for large N as a function of the speedup and the desired rate.

50 μsec. Then, with rate quantization and AOS, the crossbar fabric should run with a speedup of,

$$\begin{aligned} speedup &= \frac{128 \times 53 \ bytes/packet \times 8 \ bits/byte}{622 \ Mbps \times 50 \ \mu sec} \\ &= 1.745 \end{aligned}$$

Contract Durations

Recall that contracts between all I-O pairs are made simultaneously. Therefore, we define a contract with a rate matrix, R and an associated duration T. Consider the contract, (R, T), where R is doubly stochastic. Since any doubly stochastic matrix is supportable with the algorithm based on the plain Birkhoff decomposition with no speedup,

$$\lim_{T \to \infty} \frac{D_{ij}(T)}{R_{ij}T} = 1 \tag{2.50}$$

However, unlike traditional voice traffic, many applications today ask for more than constant bit rate guarantees. Hence, the packet switches should also offer satisfactory guarantees for finite contract durations. The following can be written for the provided service with the plain Birkhoff decomposition,

$$1 - \frac{N^2}{T} \leqslant \frac{D_{ij}(T)}{R_{ij}T} \leqslant 1 + \frac{N^2}{T}$$

for all pairs (i, j). Thus, contracts with durations, T, comparable to or less than N^2 may not get satisfactory service quality. For example, for a 512×512 ATM switch with link rates of 155 Mbps, contract durations must be of order 700 msecs for desirable operation.

On the other hand with rate quantization,

$$\frac{D_{ij}(T)}{R_{ij}T} \leqslant 1 + \frac{N/\text{speedup}}{T}$$

In this scenario, to get satisfactory performance, contract durations must be of order $(N/\text{speedup})$. This is much better than the plain Birkhoff approach. For instance, contracts with durations ~ 1 msec would get satisfactory service guarantees in our previous example with a speedup of 1.5. Thus, the set of supported applications and traffic sources is much larger with rate quantization than without rate quantization.

As mentioned before, with rate quantization, certain contract durations exist for which all

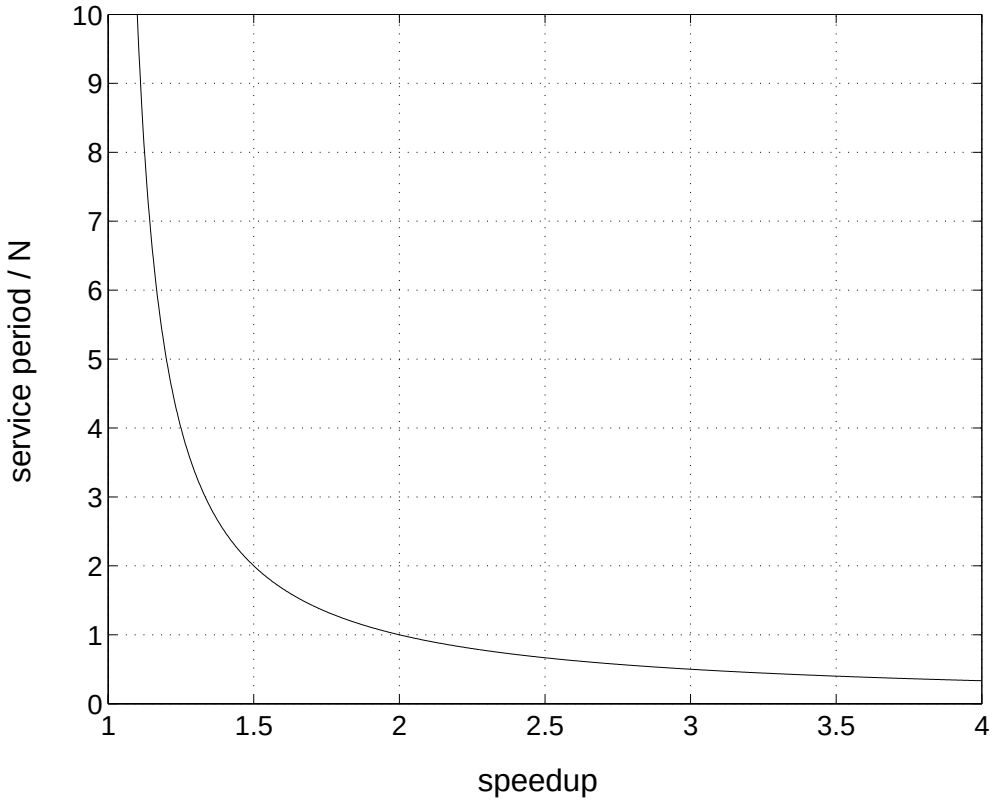

Figure 2-18: Contract durations, for which perfect support is possible with rate quantization are illustrated as a function of the speedup.

doubly stochastic matrices are perfectly supportable. Indeed, once every $N/(\text{speedup} - 1)$ time slots,

$$D_{ij}(t) \geqslant R_{ij}t$$

for all i, j as illustrated in Fig. 2-18. These instants are perfect for renegotiations and initiating connections. Note that for the set of doubly stochastic rate matrices, there are no such points using the plain Birkhoff decomposition approach with no speedup. In fact, at any point in time there exists an I-O pair (i, j) such that

$$D_{ij}(t) < R_{ij}t$$

Thus, it is not trivial to find times of renegotiation and regardless of when a new service is renegotiated it will be unfair for at least one I-O pair.

In reality, the rates between different I-O pairs can change at different times. In the next chapter, we will present an algorithm by which the schedule updates for different I-O pairs can be made without a significant change in the currently existing schedle. We also show that with an extra speedup factor of 2, the schedule updates for an I-O pair can be made without a need to rearrange the existing schedule at all. With both of these algorithms, we do not need to be concerned about the existence of points in time at which the service lag is 0 over all I-O pairs since they enable the switch to set up individual contracts (separate contract between each I-O pair). As shall be shown, rate quantization is essential for both of these to be possible.

Complexity

The complexity of Birkhoff's decomposition is $O(N^{4.5})$ since it takes $O(N^2)$ iterations of maximum matches each of which is $O(N^{2.5})$. With the plain Birkhoff approach, the decomposition should be complete before scheduling the first crossbar connection. Before finding all the permutation matrices and their coefficients, the PGPS scheduler cannot determine which permutation matrix to schedule first. Once the decomposition is complete, it takes $O(\log N)$ complexity to run the PGPS.

The most important difference arising from the use of rate quantization and AOS is that the decomposition need not be complete before the first crossbar configuration can be set up. In fact, after generating the first permutation matrix of the decomposition, it can be scheduled at once, and as the corresponding cells are being transferred, the decomposition can continue.

The rate quantization algorithm has a computational complexity of $O(N^2)$. It has to be complete before the scheduling process can start, and thus its complexity is "one time." Complexity of AOS is $O(N^{2.5})$ per service slot, which is identical to that of a maximum match based algorithm.

Comparison of the plain Birkhoff approach and rate quantization with AOS is not straightforward since the nature of complexity for the two are quite different. In the first algorithm, a huge one time cost is paid every time contracts are renewed. In the rate quantization algorithm, a smaller cost is paid and it is not a one time cost but spread in time.

As mentioned at the end of the previous section (2.3.2), in our model, we assumed that each time the rate matrix changes (even if a small fraction of entries change), the decomposition algorithm is rerun. We will show in the next chapter that with rate quantization this is not necessary. We will present an algorithm by which the schedule update for a rate update can be made using an $O(N)$ algorithm, as opposed to $O(N^{2.5})$ for rerunning the decomposition. We also show that with

an extra speedup factor of 2, the schedule updates for an I-O pair can be made without a need to rearrange the existing schedule, i.e., with O(1) complexity.

2.3.3 Performance Analysis without Speedup

Motivation and Description

In some cases it may be undesirable for the crossbar and the buffers to run faster than the link speed. For example, the link rates may be very high and it may be infeasible for the crossbar to schedule packets at rates exceeding those of the links. Besides, if the fabric runs faster than the link speed, some output queueing is necessary. In this section, we present a number of conditions under which a pure input queued (no speedup) switch performs as well as the switch with speedup and show that identical performance can be achieved without speedup.

Our main motivation is the fact that in reality, links of packet switches are not fully utilized almost all the time. In fact, recent measurements show that the average utilization of a typical link at the core router level is no more than 50%. This motivates us to redesign our algorithm to work for a utilization of less than 100%. This time, we assume that our admission controller keeps the traffic at each link below some certain level, $(1+sN)^{-1}$. Hence the rate request matrix is a doubly sub-stochastic matrix whose rows and columns sum[10] to $(1+sN)^{-1}$. Next, we state a slightly modified version of our main theorem for this case and its proof. It is clear that we achieve identical performance to the scenario with speedup, however, we will carry out the details carefully in what follows.

Theorem 2.7 *Let R be a $N \times N$ doubly sub-stochastic matrix with row and column sums identical to $(1+sN)^{-1}$ where s is a rational number which can be written as $\frac{1}{z}$ where z is an integer. There exists a (doubly stochastic) matrix, $Q = R' + U$, where R' is a doubly sub-stochastic matrix with all the entries integer multiples of $s(1+sN)^{-1}$ and row and column sums identical to $(1+sN)^{-1}$, $U_{ij} = s(1+sN)^{-1}$ and $Q_{ij} \geqslant R_{ij},\ \forall 1 \leqslant i,j \leqslant N$.*

Notice that the difference of this theorem from the main theorem is that, here we construct a doubly stochastic matrix rather than a doubly super-stochastic matrix and everything is an integer multiple of $s(1+sN)^{-1}$ instead of s.

[10]Note that each row and column of the matrix actually sums to something less than $(1+sN)^{-1}$. However, von Neumann's theorem guarantees the existence of a matrix whose entries are at least as large as the corresponding entries of the original matrix and whose rows and columns sum up to $(1+sN)^{-1}$.

Proof: If we put the matrix, $R(1+sN)$ (which is doubly stochastic) as the input to our algorithm, we get a doubly super-stochastic matrix, Q, whose rows and columns sum to $1+sN$ and whose entries are integer multiples of s as proved in the main theorem. We complete the proof noting that $Q(1+sN)^{-1}$ has the desired form.

Performance without Speedup

As can be seen from the above proof, we can write the matrix we constructed as a sum of $\frac{1}{s}+N$ permutation matrices each of which has a coefficient that is an integer multiple of $s(1+sN)^{-1}$. In the algorithm without speedup, a service slot is identical to an time slot. The periodic nature of services is still the case and the period is again $\frac{1}{s}+N$ service slots, but this time this corresponds to $\frac{1}{s}+N$ rather than $\frac{1}{s}$ time slots. Next, we analyze the performance bounds with the modified algorithm.

Theorem 2.8 *Let R_{ij} be the desired rate and $D_{ij}(t)$ be the number of service opportunities given for the cells with I-O pair (i,j) by time t (in time slots). The following holds for the modified rate quantization algorithm over the AOS:*

$$-L'(s) \leqslant D_{ij}(t) - R_{ij}t \leqslant U'(s) \tag{2.51}$$

where

$$L'(s) = \begin{cases} \frac{1}{4s}(1+sN) & , \ s \leqslant \frac{1}{N} \\ \\ \frac{N}{1+sN} & , \ s > \frac{1}{N} \end{cases} \tag{2.52}$$

and

$$U'(s) = \begin{cases} \left(\frac{1}{s}+N\right)\left(\frac{1}{2}+\frac{s}{1+sN}\right)^2 & , \ s \leqslant \frac{1}{N-2} \\ \\ \left(\frac{N}{1+sN}\right)(1+2s) & , \ s > \frac{1}{N-2} \end{cases} \tag{2.53}$$

Proof: Recall that, given the quantization parameter, s, the rate quantization algorithm generates the Q matrix for every doubly stochastic matrix, R such that the corresponding entries of R and

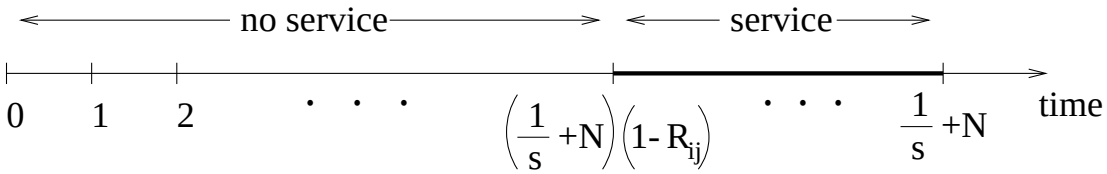

Figure 2-19: In the worst case scenario, I-O pair (i,j) can be served through $R_{ij}\left(\frac{1}{s}+N\right)$ permutation matrices each of which has a weight of $s(1+sN)^{-1}$ and can be delayed by as many as $(1-R_{ij})\left(\frac{1}{s}+N\right)$ time slots before getting the first service opportunity.

Q are within $2s(1+sN)^{-1}$ of each other:

$$R_{ij} < Q_{ij} \leqslant R_{ij} + \frac{2s}{1+sN}, \quad \forall i,j \tag{2.54}$$

1. *Lower bound*: The I-O pair, (i,j), can be served through multiple permutation matrices and in the worst case it can get no service in the first $(1-Q_{ij})\frac{1}{s}+N$ service slots and given service all of the final $Q_{ij}\frac{1}{s}$ service slots of a frame as illustrated in Fig. 2-19. This happens with AOS if the final $Q_{ij}\frac{1}{s}$ permutation matrices have $P_{\cdot,ij}=1$ and all the others have a 0 in this location. Thus, user (i,j) can possibly delayed by no more than

$$(1-Q_{ij})\left(\frac{1}{s}+N\right) \leqslant (1-R_{ij})\left(\frac{1}{s}+N\right)$$

time slots. Therefore,

$$D_{ij}(t) \geqslant R_{ij}t - (1-R_{ij})\left(\frac{1}{s}+N\right)R_{ij} \tag{2.55}$$

Hence, the bound on the service lag varies with the rate asked. If we solve the constrained optimization problem to minimize the lower bound over all possible rates:

$$\max_{R_{ij}\in[0,1]} (1-R_{ij})\left(\frac{1}{s}+N\right)R_{ij}$$

we get,

$$R_{ij}^{*} = \min\left\{\frac{1}{2}, \frac{1}{1+sN}\right\}$$

and when we plug this in our objective function, we get the lower bound, $L(s)$, given in (2.51).

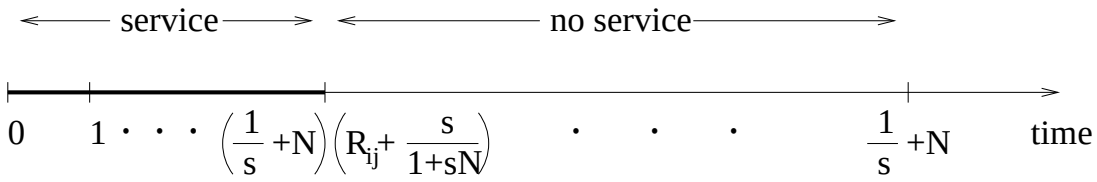

Figure 2-20: I-O pair (i,j) can be given service opportunities at the beginning of every period for $\left(\frac{1}{s}+N\right)\left(R_{ij}+\frac{2s}{1+sN}\right)$ time slots.

2. *Upper bound*: Similarly, the number of provided service opportunities can exceed the requested amount. For instance, i-O pair (i,j) can get all his service in the first $R_{ij}\left(\frac{1}{s}+N\right)+2$ time slots as illustrated in Fig. 2-20. The extra 2 time slots of service are due to (2.54). Thus, the "service lead" can be bounded as,

$$\begin{aligned} D_{ij}(t) &\leqslant R_{ij}t + \left[R_{ij}\left(\frac{1}{s}+N\right)+2\right]R_{ij} \\ &= R_{ij}t + \left(\frac{1}{s}+N\right)\left(R_{ij}+\frac{2s}{1+sN}\right)(1-R_{ij}) \end{aligned} \tag{2.56}$$

Similarly, solving the constrained optimization problem to maximize the upper bound over all possible rates:

$$\max_{R_{ij}\in[0,1]} \left(\frac{1}{s}+N\right)\left(R_{ij}+\frac{2s}{1+sN}\right)(1-R_{ij})$$

we get,

$$R_{ij}^{*} = \min\left\{\frac{1}{2}-\frac{s}{1+sN}, \frac{1}{1+sN}\right\}$$

Plugging this in our objective function, we get the upper bound given in (2.51) completing the proof.

Note that the lower and upper bounds derived for AOS are also tight for the PGPS scheduler. Consider some input output pair, (i,j). Suppose, after rate quantization, $Q_{ij} \approx R_{ij}$ and all the permutation matrices that have $P_{\cdot,ij} = 1$ have a coefficient, $s(1+sN)^{-1}$ in the decomposition. Since $s(1+sN)^{-1}$ is the minimum possible coefficient for a permutation matrix in a decomposition, it is indeed possible for all the matrices with $P_{\cdot,ij} = 1$ get scheduled at the end of a frame of a

PGPS schedule, and thus, the lower bound for the service lag is tight. Similarly, I-O pair (i, j) may get all its service opportunities in the first $R_{ij}\frac{1}{s} + 2$ service slots of a frame, and hence, the upper bound is also tight.

Even though, the lower bound derived in (2.55) and the upper bound derived in (2.55) are different from their counterparts with speedup, the minimum lower bound and maximum upper bound given in (2.51) are identical to those with speedup. The reason for this is even if the bounds (with and without speedup) are different, the solutions to the constrained optimization problems are identical in both cases; however, the rates that maximizes the objective functions are different. Indeed these rates without speedup are the scaled versions of their counterparts with speedup with a factor of $1 + sN$, which is not surprising. For large values of N, the upper bound and the lower bound are approximately the same and they scale linearly with N. The quality of service bounds with speedup (and hence the no speedup) for large N are illustrated in Fig. 2-21. The actual upper and lower bounds are illustrated in Fig. 2-22 as a function of the pair, (S,rate) where $S = 1 + sN$ is the speedup. The bounds in the curves are given per port and should be scaled with the number of ports, N. Notice that feasible rate region is $R_{ij} \leqslant \frac{1}{1+sN}$ and outside of this region is clipped out in the graph.

2.4 Probabilistic Scheduling

So far, we have talked about deterministic schedulers and performance guarantees. After the decomposition step, a deterministic schedule was generated and it remained fixed. In this section, we will introduce a very simple probabilistic scheduler and talk about its performance *on average*.

Suppose we have the Birkhoff decomposition for the rate matrix. It does not matter whether this matrix is quantized or not in this part. Let us consider the decomposition for a matrix which is not quantized. There is a weight, ϕ_i associated with every permutation matrix, P_i. The sum of weights for all the permutation matrices with $P_{\cdot,ij} = 1$ gives the total rate, R_{ij}, between I-O pair (i, j).

Consider a scheduler that sets up the crossbar configuration corresponding to P_i with probability ϕ_i every time slot, independent of everything else. With this scheduler, in each time slot, the I-O pair, (i, j), gets a service opportunity with probability R_{ij}. Thus, the service opportunities provided

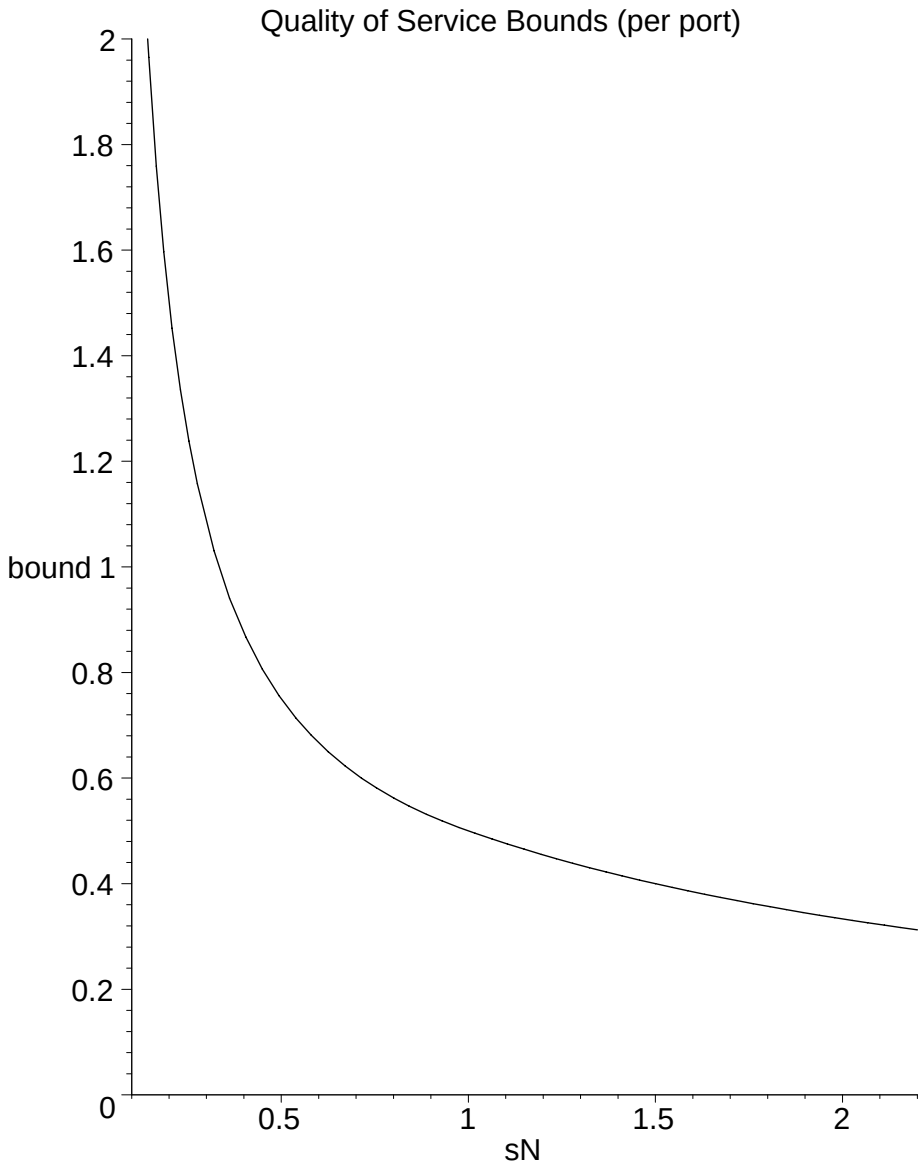

Figure 2-21: Illustration of the quality of service curves without speedup (given in (2.51)) as a function of $S-1$ for large N, where $S = 1 + sN$ is the speedup.

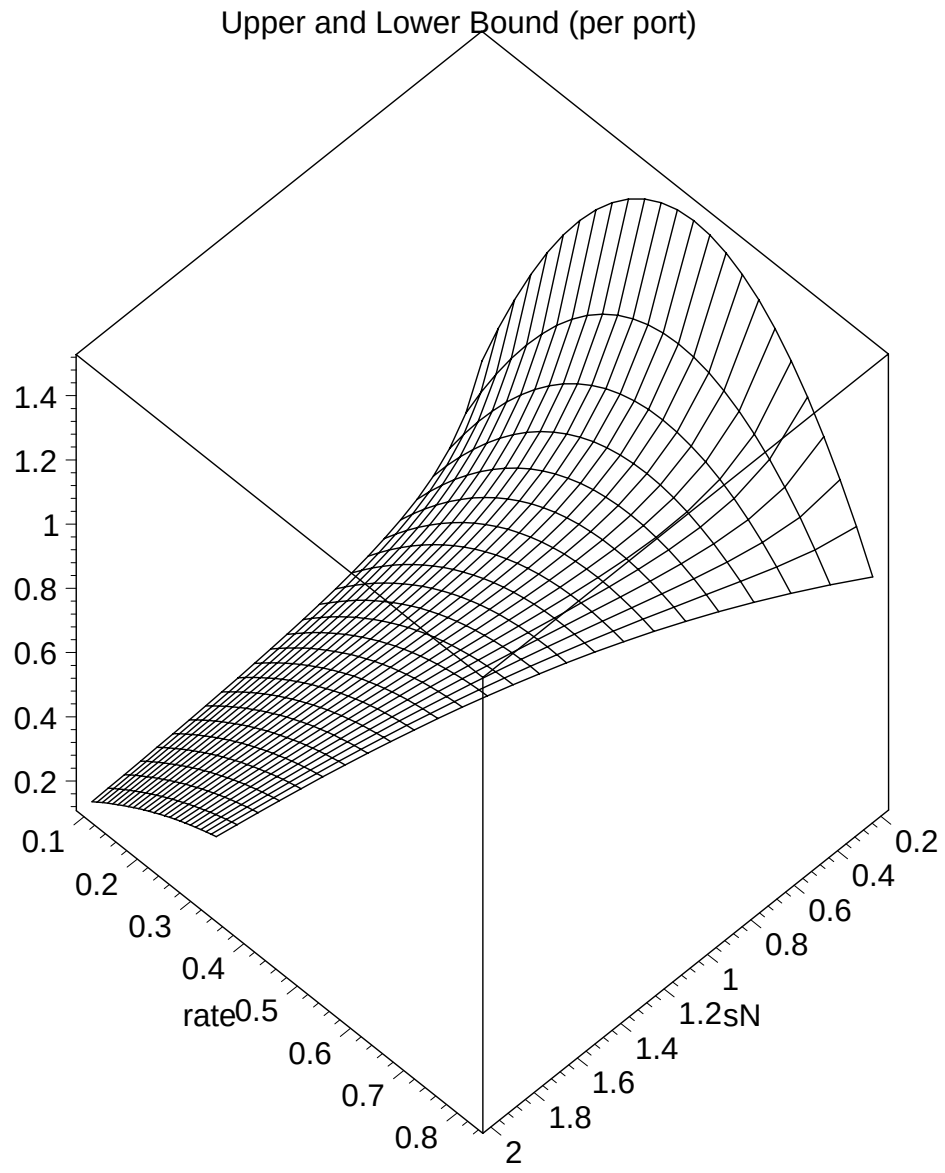

Figure 2-22: Illustration of the upper bound (given in (given in (2.56)) as a function of $S-1$ and the desired rate where $S = 1 + sN$ is the speedup. The clipped region is $R_{ij} > \frac{1}{S},\ \forall i, j$ and it is infeasible.

to this pair is a geometric random process with parameter, R_{ij}. By the strong law of large numbers,

$$\lim_{t\to\infty} \frac{D_{ij}(t)}{t} = R_{ij} \tag{2.57}$$

with probability 1. Next, let us focus on the service lag. Let t and t' be two points in time, such that $t < t'$. Define $D_{ij}(t, t']$ as the number of service opportunities given to the pair in period $(t, t']$. Then,

$$\begin{aligned} \Pr\left[\left(D_{ij}(t,t'] - R_{ij}(t'-t)\right) > L\right] &= \sum_{l=L+\lfloor R_{ij}(t'-t)\rfloor+1}^{t'-t} \left(\begin{array}{c} t'-t \\ l \end{array}\right) R_{ij}^{l} \left(1-R_{ij}\right)^{t'-t'-l} \\ &\leqslant \left(1 - R_{ij} + R_{ij}e^{r}\right)^{(t'-t)} e^{-r(R_{ij}(t'-t)+1)} e^{-rL} \end{aligned} \tag{2.58}$$

for all $r \geqslant 0$, where (2.58) follows from the Chernoff bound (see e.g., [51]). The bound is a convex function of the parameter r. It can be minimized over $r \geqslant 0$ to evaluate the value for which the bound is the tightest. We will not do that, but, note that (2.58) is exponentially tight, i.e., the probability that the service lag exceeds some $L > 0$ decays to 0 exponentially fast in L.

Even though deterministic guarantees cannot be given by this randomized scheduling algorithm, the probabilistic guarantees are very impressive given the simplicity of the algorithm. For instance, recall that the upper bound on the service lag with plain Birkhoff approach is (N^2). The probability that the service lag exceeds N^2 has a dominant term of $\exp(-rN^2)$, which approaches 0 more than exponentially fast as N. We conclude this section noting that probabilistic schedulers can be very compelling in both their simplicity and performance.

2.5 Conclusions and Future Work

We presented a rate quantization algorithm which, along with some speedup, significantly improves the performance and practicality of reservation based scheduling algorithms. We showed that the service bounds for Birkhoff-von Neumann switches derived in [15] are tight. Besides, there is a high complexity associated with the decomposition algorithm and it is not possible to run it simultaneously as the crossbar connections are made. Thus, the quality of service provided over such switches may not be satisfactory for a large set of applications and traffic sources.

Rate quantization improves the bounds by a factor $O(N)$, and there are certain points in time where the service lag is 0 simultaneously for all I-O pairs. Also, the decomposition can run

simultaneously as the crossbar configurations are set up; the complexity of decomposition is thus spread over a larger time period. We illustrated that even with a small speedup, rate quantization with AOS expands the set of contracts for which satisfactory service guarantees can be provided a great deal.

In some cases, speedup may be undesirable. If there are not long periods of time where links are fully utilized, we showed that rate quantization performs well without speedup. It does so by utilizing the unused resources to smooth the service and improve the quality of service bounds.

In what follows, we give directions for future research.

- The schedulers we considered take only the decomposition coefficients into consideration, and disregard individual service parameters completely. Instead, we can define a matrix of costs (one for every I-O pair) that increases proportional to the desired rate and decreases each time the corresponding pair is given a service opportunity. Then, an algorithm based on stable marriage match or maximum weight match can be used to schedule crossbar connections. We believe that such an algorithm will improve the quality of service bounds a great deal.

- Throughout this chapter, we assumed that contracts between all I-O pairs are made simultaneously and rate updates are made synchronously over all I-O pairs. In reality, the rates between different I-O pairs can change at different times. In our model, we assumed that each time the rate matrix changes (even if a small fraction of entries change), the decomposition algorithm is rerun. We will show in the next chapter that, this is not necessary. We will present an algorithm by which the schedule update for a rate update can be made using an $O(N)$ algorithm, as opposed to $O(N^{2.5})$ for rerunning the decomposition. We also show that with an extra speedup factor of 2, the schedule updates for an I-O pair can be made without a need to rearrange the existing schedule. As shall be shown, rate quantization is essential for both of these results to be possible.

- The crossbar fabric is attractive since it is non-blocking and easy to manufacture. However as the size of a switch gets larger, coordination among all the ports becomes increasingly difficult and thus, many algorithms get very complex as the switch size grows. This compels us to look into distributed architectures with less coordination among different units of the architecture. An in depth study of multistage architecures can be found later in this thesis.

Chapter 3

Isomorphism Between Crossbar Switch Schedulers and Clos Networks

3.1 Motivation

In the previous chapter, we studied service guarantees for connection based contracts. We defined a contract as a doubly stochastic rate matrix and a duration that represents the "lifetime" of the contract. According to that model, at the end of the contract lifetime, another contract with a new duration and set of rates is negotiated. The switch has to calculate a new schedule, and crossbar configurations are set according to the new schedule. In this chapter, we study contracts where the duration of contracts is not necessarily the same for all input output pairs. The main motivation for such an effort can be given as follows.

For a switch of size $N \times N$, there are N^2 input output (I-O) pairs. In practice, the desired rates between different I-O pairs may change independently of each other. Suppose each I-O pair updates its rate every T units of time on the average. This corresponds to N^2/T changes per unit time. This has a significant impact on the implementation complexity of the scheduling algorithms. Indeed, if rates are updated one at a time, even with rate quantization, a given set of rates must be kept for about N time slots for satisfactory service quality with the type of scedulers we introduce. This corresponds to $O(N^3)$ time slots for maturity of an individual contract.

First we show that there is a one to one correspondence between the service provided by a crossbar that alternates over a number of configurations in a time division multiplexed (TDM) manner and the service provided by a three stage Clos network composed of crossbars that have

fixed configurations.

Then, we study the Slepian-Duguid algorithm (originally developed for Clos networks; see [37] for an in depth treatment) with which rate updates can be made with minimal modification to the existing schedule in a simple and efficient way. We show that rate quantization is necessary for this approach to be successfully implemented, and discuss certain trade-offs. Then we will evaluate the necessary speedup that enables a single crossbar switch to schedule contracts independently of each other subject to feasibility constraints. Our purpose is to accommodate rate updates of an I-O pair without changing the existing schedule of configurations.

3.2 Space Switching vs. Time Switching, Clos Network Analogy

After giving some definitions, we show that there is a one to one correspondence between the service provided by a crossbar that alternates over a number of configurations in a time division multiplexed (TDM) manner and the service provided by a three stage Clos network composed of crossbars that have fixed configurations. For an extensive treatment of Clos networks, see [37], [56].

We consider the single crossbar switch model developed in the first two chapters. Suppose we have a rate matrix R, whose entries are integer multiples of τ^{-1}, for some $\tau \in \mathbb{Z}^+$. We call such matrices τ-periodic since they can be supported with a schedule of τ crossbar configurations. Matrix τR has integer entries and the (i, j) entry represents the number of times input i should be connected to output j within τ time slots. In this chapter, we assume R to be doubly stochastic. Suppose we want to construct a schedule for the support of R.

We convert this problem into a dual problem as follows:

1. We defined a time slot as the period of time it takes for a typical link to transmit one cell. Let a *frame* be the time it takes for a link to transmit τ cells. Thus, one frame contains τ time slots, and a link can transmit up to τ cells in a frame. We use t and t' to identify time slots and frames respectively, e.g., $(t'-1, t')$ is the t'th frame. A frame is illustrated in Fig. 3-1. Suppose, within a frame, $(t'-1, t')$, τ' service opportunities are given. Hence, the switch operates at a speedup of τ'/τ, and a service slot is a fraction τ/τ' of a time slot.

2. Instead of having the crossbar schedule τ' configurations in a frame and repeating it every frame, let us use τ' parallel crossbars each of which have a fixed configuration. Namely, each crossbar is configured to only one permutation matrix. Let the kth crossbar be set to the

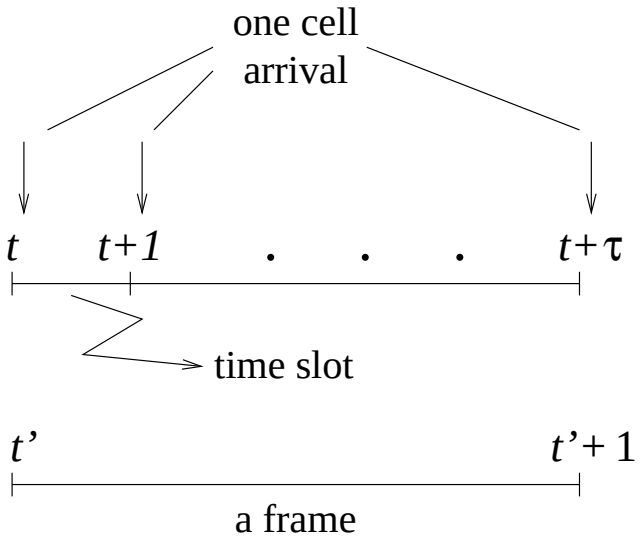

Figure 3-1: A frame is defined as the time it takes for a link to transmit τ cells.

configuration corresponding to the permutation matrix implemented by the original crossbar in the kth service slot of a frame. The, τ' crossbars perform the function of the single fast crossbar; but instead of configuring τ' permutation matrices one each service slot, in this scenario they are configured simultaneously one each crossbar use. These $N \times N$ crossbars are illustrated in the middle of the system on the right side of Fig. 3-2.

At this point, we completed the first part of the analogy. Before we go on, let us describe the time division multiplexers and demultiplexers (time division switches) shown in Fig. 3-2. Up to τ cells arrive at each time division demultiplexer within a frame. These cells are reordered in the frame for each one to be sent to the appropriate middle crossbar. Similarly, in a frame, each time division multiplexer gets cells from τ of τ' middle crossbars and reorders them before transmission. The reordering at a time division switch can be performed by a device called a time slot interchanger (TSI), as shown in Fig. 3-3. Conceptually, a $\tau \times \tau$ TSI can be viewed as a buffer which reads from a single input and writes to a single output. The three stage system is illustrated in Fig. 3-4. This three-stage switching system is also known as a time-space-time (TST) switching network (see e.g., [37]).

To summarize, there is a one to one correspondence between the configurations that the single (fast) crossbar switch (left side of Fig. 3-2) goes through within a frame time and the configurations of the middle crossbars of the three stage system (right side of Fig. 3-2).

Next, we get to the second part of the analogy.

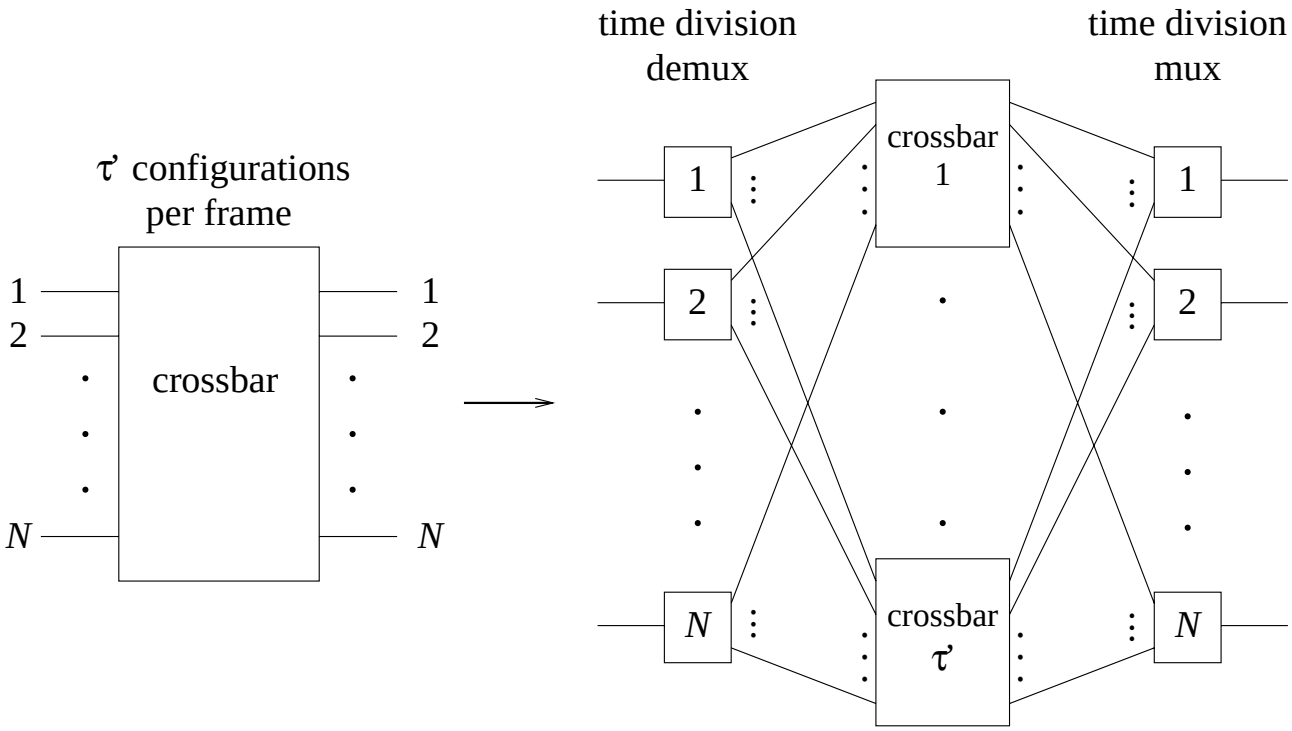

Figure 3-2: Instead of a crossbar which provides τ' service opportunities per frame, consider using τ' crossbars in parallel with static configuration.

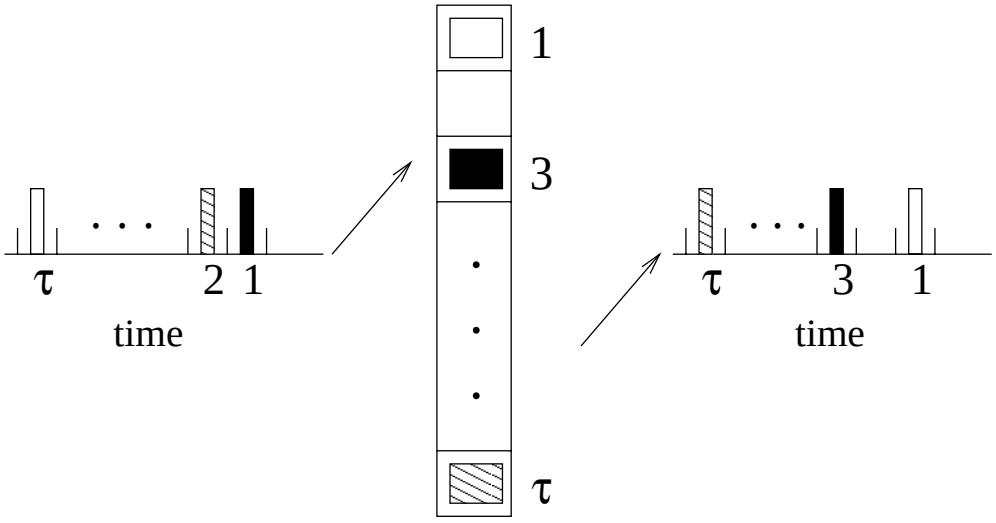

Figure 3-3: A $\tau \times \tau$ time slot interchanger

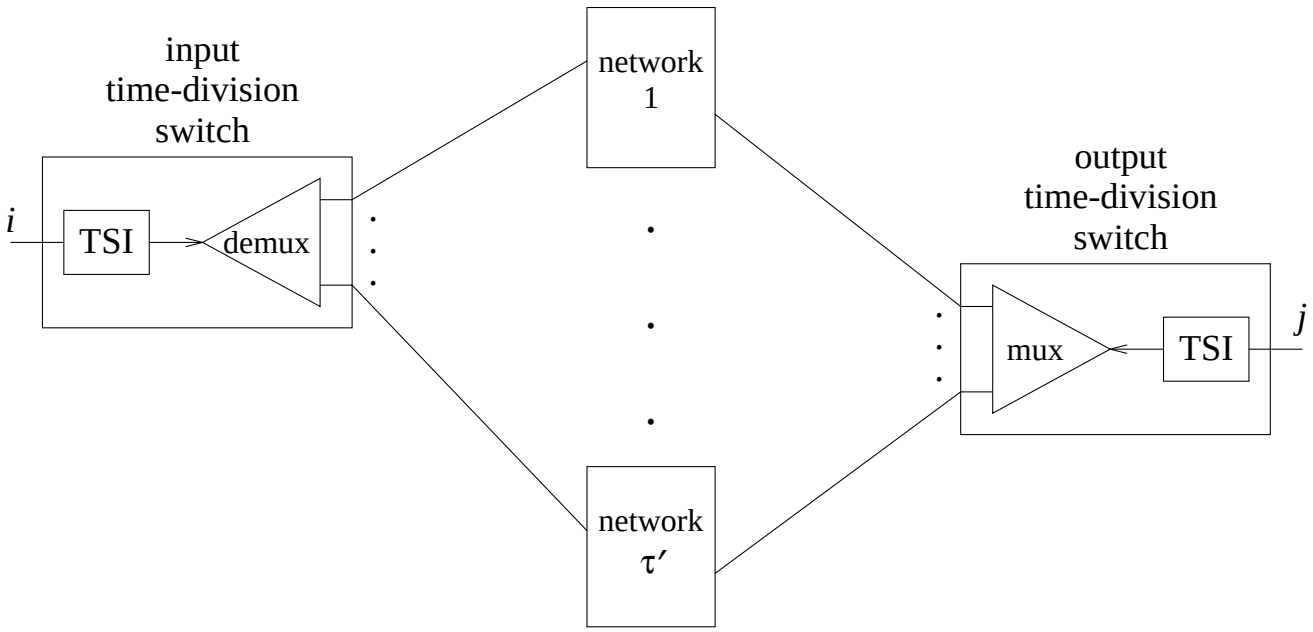

Figure 3-4: A detailed illustration of the input and output stage crossbars shown in Fig. 3-2. They have TDM switches which make use of TSIs for switching (swapping time slots). This network is also called a TST switching network.

3. If we view the inputs of the system given in Fig. 3-4 as τ cells arriving in parallel rather than in series, then the time division switches become crossbars. More precisely, suppose we divide each input and output link into τ links that are slower by a factor τ compared to the original links. Hence, instead of τ serial cell arrivals on a single input, there is one arrival on each of τ parallel inputs. The time division switches are thus replaced with crossbars of size $\tau \times \tau'$. Similarly, the output switches are replaced by $\tau' \times \tau$ crossbars. These crossbars are functionally equivalent to the time division switches as illustrated in Fig. 3-5, and the overall system is thus analogous to the network illustrated in Fig. 3-6. It is called a three stage Clos network and is the space-space-space (SSS) equivalent of the TST switching network given in Fig. 3-2. We will represent the Clos network given in this figure with three parameters, the number of input links per input crossbar, the number of input (output) crossbars and the number of middle crossbars: $\mathrm{C}(\tau, N, \tau')$. Note that, since the traffic is τ-periodic, the destination of the cells arriving at one of the τ input links on an input crossbar remains unchanged. Namely, each input link in the Clos network has to be connected to the same output link at all times. Hence all the crossbars in the new system will have fixed configurations.

We have just established an analogy between the end to end configurations of an $N\tau \times N\tau$ three

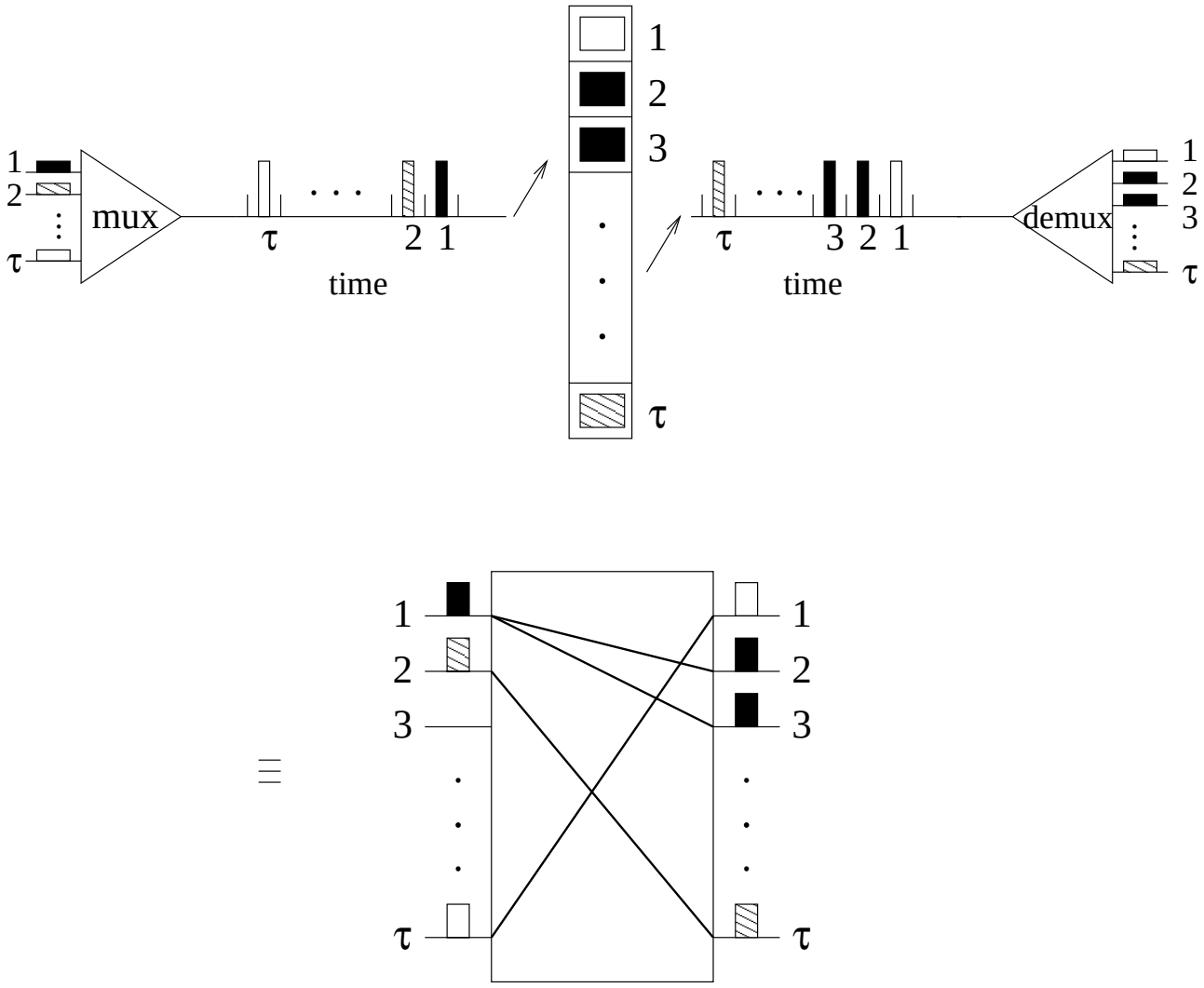

Figure 3-5: The time division switch consisting of a multiplexer, a TSI and a demultiplexer in cascade is identical to a non-blocking space division switch.

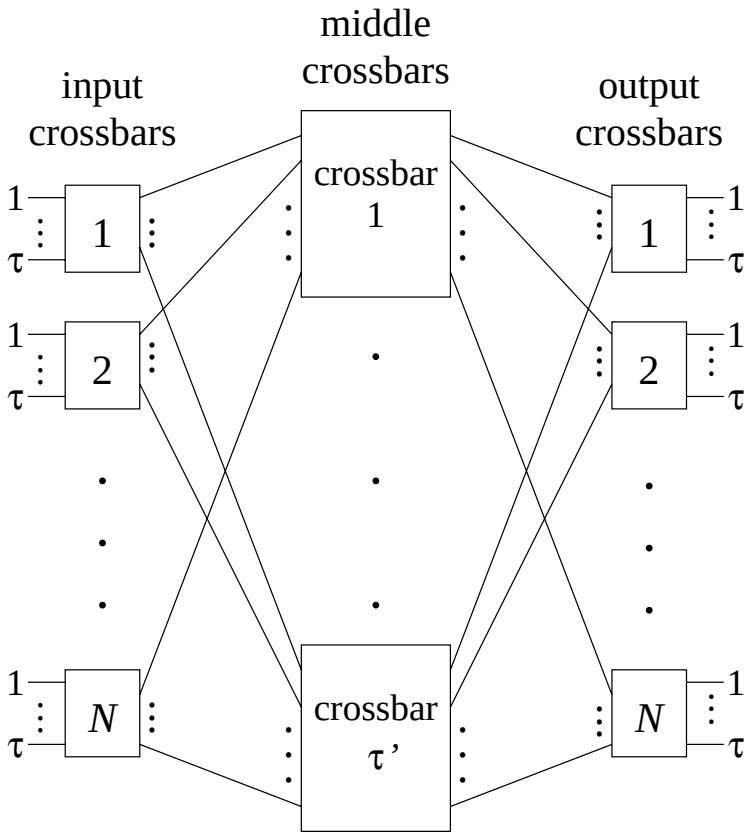

Figure 3-6: A typical Clos network. It is also the SSS equivalent of the network given in Fig. 3-2.

stage Clos network, $C(\tau, N, \tau')$, composed of crossbars with fixed configurations and the schedule of configurations of an $N \times N$ crossbar under τ-periodic traffic. Indeed, any τ-periodic schedule for the original crossbar corresponds to a *fixed circuit assignment* (see e.g., [37]) in the Clos network, $C(\tau, N, \tau')$. The configuration held by the original crossbar in the kth service slot of each frame corresponds to that of the kth middle crossbar in the Clos architecture.

Next, we will talk about a number of properties for the non-blocking three stage Clos networks, and comment about their significance in the context of the single crossbar.

3.3 Non-blocking Scheduling for Crossbar Switches

Blocking is the failure to satisfy certain connection requirements because of the absence of non-conflicting paths between inputs and outputs for those requirements. In this case a connection may not be established even if the I-O pair asking for the connection is not busy handling other connections. An interconnection network is non-blocking if a connection can always be set up between any idle input and any idle output. This version of non-blocking behavior is also known as non-blocking in the classical sense. There are multiple degrees of non-blocking in the classical sense:

1. A network is strictly non-blocking if a connection between an idle input and output can always be established without rearranging the existing connections between other I-O pairs.

2. If a connection between an idle I-O pair cannot necessarily be established without rearranging the existing connections, the network is called rearrangably non-blocking.

3. If the necessity of rearranging the existing connections to accommodate new ones can be avoided using some algorithm to set up new connections, the network is called wide sense non blocking.

For example a crossbar is strictly non blocking in the classical sense. Note that every strictly non-blocking network is also wide sense non-blocking and every wide sense non-blocking network is also rearrangable, but the reverse of these are not necessarily true. The necessary and sufficient number of middle crossbars for rearrangably non-blocking Clos networks is given by the following theorem.

Theorem 3.1 *$C(\tau, N, \tau')$ is rearrangably non-blocking if and only if $\tau' \geqslant \tau$.*

This result is attributed to Slepian ([53]). He presents an algorithm for rearrangements to accomodate a new connection in $C(\tau, N, \tau')$. We use a simplified version of Slepian's rearrangement algorithm in our schedulers.

In general, to upgrade a network from one degree of non-blocking to another, resource speedup is introduced. Resource speedup can be provided in various ways. For instance, a crossbar may transfer cells faster than the links transmit, or the number of middle crossbars in the three stage Clos network may be increased. The speedup necessary and sufficient for $C(\tau, N, \tau')$ to be strictly non-blocking is $2 - \frac{1}{\tau}$:

Theorem 3.2 *$C(\tau, N, \tau')$ is strictly non-blocking if and only if $\tau' \geqslant 2\tau - 1$.*

This result was shown by Clos himself in his 1953 paper ([52]) and can be found in standard texts on switching (e.g., [37]).

3.3.1 Slepian Duguid Theorem and Non-blocking Scheduling for Crossbar Switches

In this section, we illustrate how a new connection request can be handled in a Clos network with minimal modification to the existing connections using the Slepian Duguid algorithm. Then, we discuss its implications on single crossbar switch scheduling.

Consider fixed connections between the $N\tau$ input links and the $N\tau$ output links of the Clos network $C(\tau, N, \tau')$. Let R_M be an $N \times N$ matrix of integers where the entry (i, j) represents the total number of connections between the ith input crossbar and the jth output crossbar. Thus, R_M is τ times a τ-periodic rate matrix. The problem of finding the configurations of τ middle crossbars is known as the fixed circuit assignment problem.

Birkhoff's decomposition can be used to find the middle crossbar configuration. Due to the special form of R_M, the coefficient of each permutation matrix in the decomposition is an integer and these integers sum to τ. A permutation matrix with coefficient k is assigned to k middle crossbars. Thus, the necessary number of crossbars is $\tau' = \tau$ since the decomposition algorithm terminates with that many permutation matrices (which need not be distinct).

A Clos network is said to be fully connected if each input link is matched with an output link. A fully connected Clos network and the corresponding rate matrix are illustrated in Fig. 3-7.

Now suppose we have a Clos network which is not fully connected, and a new connection between an idle input link at input crossbar i and an idle output link at output crossbar j is to be set up. Since input crossbar i has an idle input link, it must also have an idle output link. Since an input

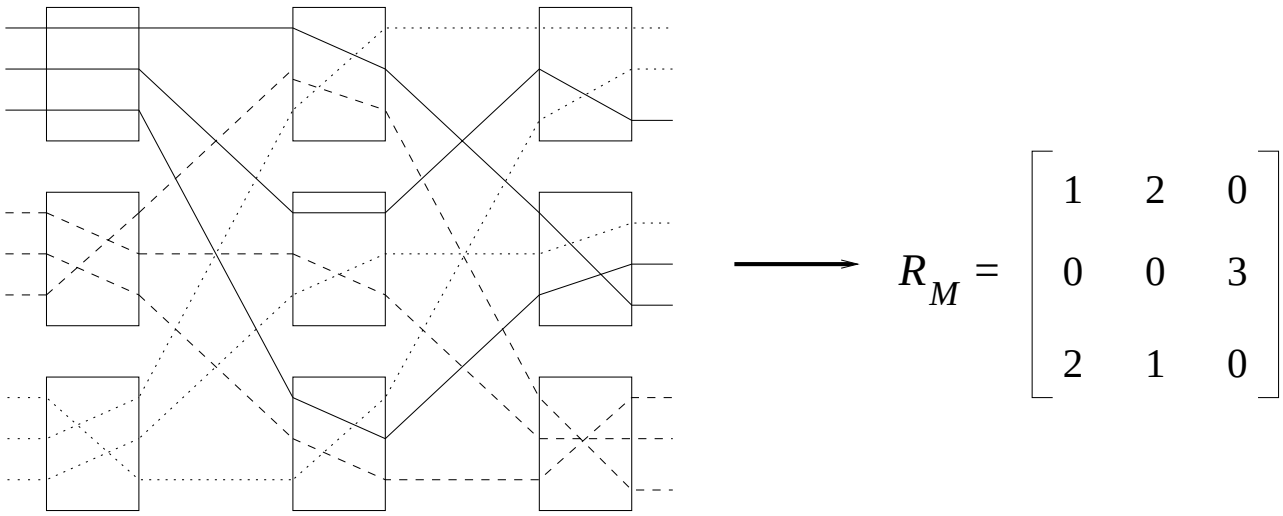

Figure 3-7: A fully connected Clos network, and the corresponding R_M.

crossbar has a link to all the middle crossbars, there exists a middle crossbar, x, which has an idle input and thus an idle output link. Similarly, since output crossbar j has an idle output link, there exists a middle crossbar, y, that has an idle input link and an idle output link. If the middle crossbars x is the same crossbar as y, then input crossbar i can be connected to output crossbar j through that middle crossbar without a need for any rearrangement of the existing connections. Otherwise, a rearrangement is necessary. In fact, the rearrangement process involves only the two crossbars, x and y. We present the following approach for the rearrangement process.

First, we construct a reduced rate matrix, R_M^{xy}, as follows. Let the configuration matrices of x and y be P_x and P_y respectively. Let

$$R_M^{xy} = P_x + P_y + I_{ij}$$

where (i, j) entry of I_{ij} is 1 and all others are 0. One can see that R_M^{xy} has all integer entries and $\frac{1}{2}R_M^{xy}$ is a doubly substochastic matrix. Thus, one can find two permutation matrices, P_x' and P_y' such that

$$P_x' + P_y' \geqslant R_M^{xy}$$

where the inequality is entrywise. There are simple ways to find the two connection matrices, P_x' and P_y' due to the special form of R_M^{xy}. We present one of them, which is slightly modified version of the Slepian Duguid algorithm.

Let us define the matrix $\rho_M^{xy} = xP_x + yP_y$. Note that the middle crossbar IDs, x and y are used as symbols only. Namely, each entry of ρ_M^{xy} is an element of the set $\{0, x, y, x+y\}$. We emphasize that $x+y$ is not a sum of two numbers, but rather represents a connection between an input crossbar output crossbar pair through both middle crossbars x and y. Each row and column of matrix ρ_M^{xy} contains x and y at most once (a row contains symbol x if there is an x or an $x+y$ entry at that row). If the (l,m) entry of matrix ρ_M^{xy} is x, then the lth input and the mth output of crossbar x are connected. If that entry is $x+y$, then I-O pair (l,m) is connected in both crossbar x and crossbar y. Matrix ρ_M^{xy} is a reduced version of *Paull's connection matrix* (see e.g., [37] for a definition) for our three stage Clos network.

Recall that a connection between input crossbar i and the output crossbar j is requested. Since there is a need for a rearrangement, the ith row of ρ_M^{xy} contains either symbol x or symbol y, but not both, and the jth column contains the other symbol. Without loss of generality, let us assume row i contains symbol x and column j contains symbol y. The algorithm has two steps, the search and the rearrangement.

The search process is illustrated in Fig. 3-8. First, we search matrix ρ_M^{xy} for an x or an $x+y$ in the row of the entry which contains the y in column j. If such an x can be found, we then look for a y in that column. If such a y can be found, we search for an x in its row. The alternating search process terminates if the symbol in consideration (either x or y) cannot be found. The search process takes at most $2N-2$ steps since there are a total of $2N-2$ rows and columns combined, other than row i and column j.

Once the search process terminates, the rearrangements can be made as shown in Fig. 3-9. All the entries on the path from the first y to the last entry that the search process finds are flipped: Each x is replaced with a y and each y is replaced with an x. The y in column j is thus changed into an x, and a y can be inserted in position (i,j) since neither the ith row nor the jth column contains a y any more. Thus, we can accomodate the new connection request between input crossbar i and output crossbar j through middle crossbar y after the rearrangement. By inspection, one can see that after this process, each row and each column of the modified Paull's matrix still has no more than one x and one y.

We started the search process with the row that contains the y. We could as well start it with the column that contains the x, and the procedure would still work. Indeed, Paull ([54]) proposed a slight modification to the Slepian Duguid algorithm: Instead of constructing one chain starting

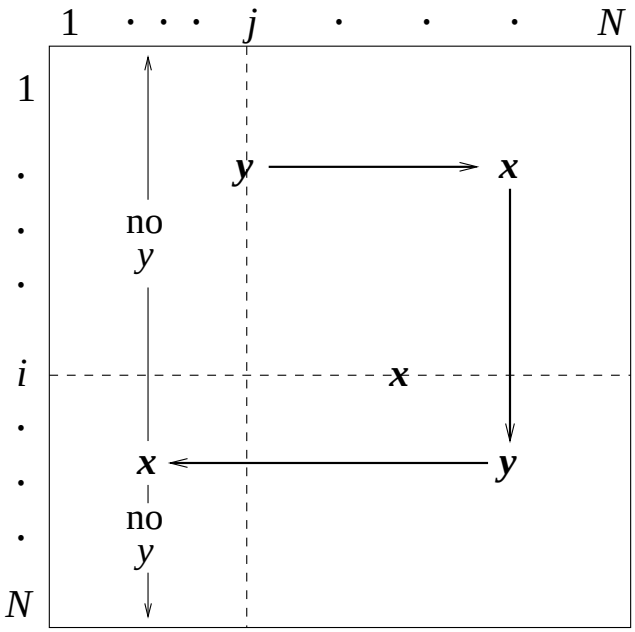

Figure 3-8: The alternating search process terminates when a y in the column of an x or an x in the column of a y cannot be found in matrix ρ_M^{xy}. The process takes at most $2N-2$ steps.

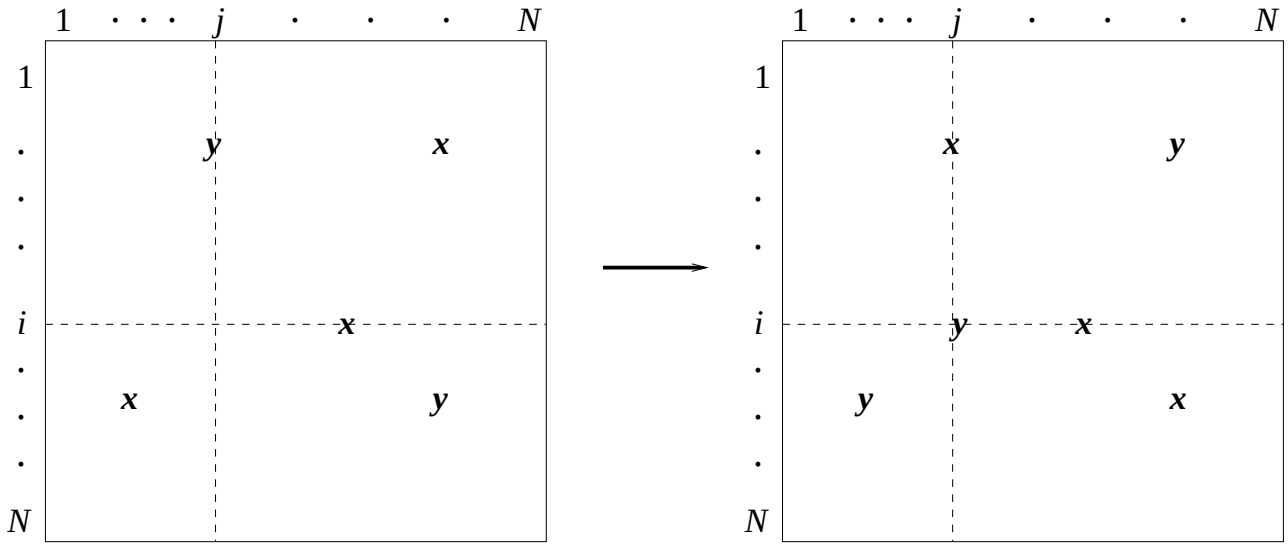

Figure 3-9: Each symbol found by the search process, starting with the first y is flipped, i.e., each y is replaced with an x and each x is replaced with a y. At the end, an extra y can be inserted as the (i, j) entry of the matrix.

with one of the symbols, x or y, construct two chains, one for each symbol, and run the two search processes in parallel. Since the sum of the length of the two chains can at most be $2N$ (since there are that many entries that contains an x or a y), one of these chains has a length no more than N. Once the shorter chain is constructed, the rearrangement procedure can be applied on this chain. This cuts the maximum number or rearrangements down to a half of that of the original procedure, hence we need at most $N-1$ rearrangements.

The described rearrangement algorithm enabled the connection between input crossbar i and output crossbar j to be made through middle crossbar y. Note that, even though up to $N-1$ rearrangements is necessary for a connection request to be fulfilled, these rearrangements involve only two middle crossbars. Hence, after the rearrangement process $\tau-2$ middle crossbars will keep their configurations.

In the light of the above, let us focus on the scheduling problem for the single crossbar switch. Suppose we have a τ-periodic rate matrix, and a number of entries are updated. For the I-O pairs which decrease their rates, we can simply vacate that pair from sufficiently many configurations. Since the coefficient of each permutation matrix is τ^{-1}, one such pair should be vacated per τ^{-1} decrement. For the I-O pairs which increase their rates we can use the Slepian Duguid rearrangement procedure exactly the same way it is used to establish a new connection in a Clos network. To accommodate each τ^{-1} increment, we need to modify only two permutation matrices among a total of τ. The complexity of the rearrangement procedure is $\mathrm{O}(N)$, which is an $\mathrm{O}N^{1.5}$ improvement over rerunning the decomposition, each time an entry is changed. Only two permutation matrices are modified and the rearrangement process implies a schedule change for no more than $N-1$ I-O pairs other than the one which asks for the rate increase. Similarly, this is a factor $\mathrm{O}(N)$ improvement since, if we ran the decomposition after each schedule change, it might be the case that the entire schedule needs to be changed.

For the rearrangement algorithm to be applied to the single crossbar switch schedule, our main assumption is that the traffic can be made τ-periodic, i.e., all the entries of the rate matrix are integer multiples of τ^{-1}, for some $\tau \in \mathbb{Z}^+$. If we do not have a τ-periodic rate matrix, rates must be quantized as described in Chapter 2. There are some trade-offs we need to take into consideration to choose the quantization parameter, τ. The necessary speedup for quantization is $1+\frac{N}{\tau}$, which increases as τ decreases. On the other hand, in a system where desired rate changes are small[1],

[1]For instance, suppose the rates are updated according to the state of the VOQs. As the number of cells start to

the frequency of rate updates will increase as τ increases. Also, if the change in the desired rate is high compared to τ^{-1}, the number of users that may be affected by the update will increase. Indeed, with each τ^{-1} increase in the rate, the number of users that possibly need a schedule update increases by N.

3.3.2 Strictly Non-blocking Scheduling for Single Crossbar Switches

Recall that Theorem 3.2 gives the number of middle crossbars necessary for strictly non-blocking Clos networks. Using that result, we derive the necessary speedup to enable independent scheduling in the crossbar switch.

Theorem 3.2 implies that if the number of middle crossbars is doubled, a new connection (a request for a fixed circuit) can be accommodated without rearranging the currently existing middle crossbar configurations.

An equivalent statement for the $N \times N$ crossbar switch is the following. Any admissible unicast τ-periodic traffic can be supported over a crossbar with a speedup $S = \frac{\tau'}{\tau} = 2 - \frac{1}{\tau}$. Moreover, if multiple entries are changed in the original rate matrix, the new rates can be accommodated without a need for rearranging the schedule of the I-O pairs with unchanged rates. The I-O pair for which a decrease in rate occurred can vacate sufficiently many configuration entries. The I-O pairs with increased rates can search the $2\tau - 1$ configuration matrices for the one that has a 0 in the corresponding location. The existence of such a matrix is guaranteed by Theorem 3.2. We will call such schedules for the crossbar *strictly non-blocking*.

Strictly non-blocking scheduling requires τ-periodic traffic to generate the schedule of $2\tau - 1$ configuration matrices. Since τ can be picked arbitrarily large, for any rate matrix with real entries, a τ can be found for which every entry is arbitrarily close to an integer multiple of $\frac{1}{\tau}$. However, the length of the schedule is also proportional to τ, and hence, it is not desirable to pick τ very large. Instead, suppose we pick a predetermined value for τ, so that the schedule length is no longer than $2\tau - 1$ permutation matrices. If the rate matrix is not τ-periodic, we can use the rate quantization procedure described in chapter 2. We can generate a Q matrix whose rows and columns sum to

increase, the rates are increased accordingly, and vice versa. In such a system, the rate updates may be desired quite frequently and consequently, the amount of change may be small.

$1 + \frac{1}{\tau}N$, and for strictly non-blocking scheduling, an overall speedup of

$$\begin{aligned} S &= \left(1 + \frac{N}{\tau}\right)\left(2 - \frac{1}{\tau}\right) \\ &= 2 + \frac{2N-1}{\tau} + o\left(\tau^{-1}\right) \end{aligned}$$

would be necessary. Thus, rate quantization requires an extra speedup of $\frac{2N-1}{\tau}$. As shown earlier, the service lag and the corresponding cell delay is proportional to τ, whereas the necessary speedup is inversely related to this parameter, which is, in fact, the fundamental trade-off for all rate quantized systems.

3.4 Conclusions

Multimedia sources and data sources require service guarantees which can be dynamically updated in the short term as well as the long term. We introduced two different methods for making schedule updates for changing rate requirements between I-O pairs with only minimal changes in the existing schedule. The first one is based on Slepian Duguid theorem for circuit rearrangements in Clos networks. It does not require any speedup beyond that necessary for rate quantization. The second one completely decouples the schedule updates of different I-O pairs, i.e., any change in the rate of an I-O pair can be accommodated, without any need for rearrangement of the existing schedule of other I-O pairs. Using an analogy between single crossbar switch schedules and three stage Clos networks, we showed that a speedup of 2 is sufficient for strictly non-blocking switch schedules for unicast traffic over single crossbar switches.

In this chapter, we did not talk about *wide sense non-blocking* scheduling for single crossbar switch. If we follow a certain wide sense non-blocking algorithm (see e.g., [56]), scheduling of rate updates could be achieved without a need for rearrangements with a speedup lower than 2. For a detailed treatment of such algorithms and corresponding speedups, see e.g., [56].

In the next chapter, we study the support of multicast rates over crossbar switches. We show that, unlike unicast, such rates are not supportable over single crossbar switches without speedup. Then, we will use the Clos network analogy again to show that a speedup of $O(\log N)$ is necessary for multicast support with strict sense non-blocking scheduling. We will also use the insights we gained in this chapter to study rate guarantees over wavelength switches.

Chapter 4

Multicast Support over a Single Crossbar Switch

4.1 Introduction

In this chapter, we will study *multicast support* over crossbar switches. Many applications and traffic sources may request that the same information be sent to multiple points in the network, a situation called multicast. An increasing proportion of traffic on the Internet is multicast. Instead of sending a separate packet to each of the destinations, the source might send a single packet to a *multicast address*. The network then delivers a copy of that packet to each of the destinations in the network. While multicast is not supported by many of the routers in the Internet, wide area multicast is made available via the Multicast backbone (Mbone), [32], [33]. The Mbone is a logical internet layered over the top of the current Internet. That is, multicast-enabled routers tunnel to each other through the existing Internet. Any regular routers between two multicast enabled routers process only their own (unicast) headers and never have to worry about multicast addresses.

The trivial solution to multicast support is to duplicate multicast packets upon arrival to a switch and to treat each one as a separate unicast packet. However, higher throughput can be attained if we take advantage of the natural multicast properties of switching fabrics. For instance, the crossbar can copy one input cell to any number of outputs for which there is no conflict in a single cell time[1]. In Fig. 4-1, crosspoints (1,1) and (1,2) are shorted which assumes that the

[1] In the second chapter, we assumed that a crossbar is incapable of making broadcast connections. In the second

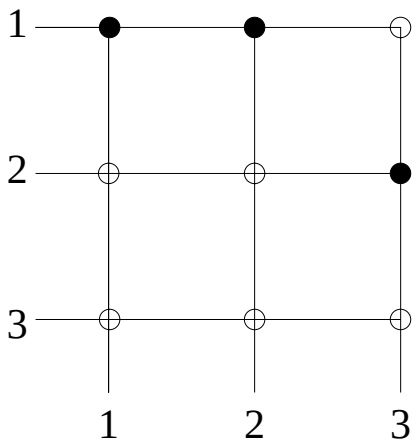

Figure 4-1: Crosspoints (1,1) and (1,2) are connected which enables the crossbar to copy the same packet at the first input link onto outputs 1 and 2 simultaneously.

crossbar can copy the same cell at the first input link onto outputs 1 and 2 simultaneously rather than having to send them at different times.

A number of different architectures and implementations have been proposed for multicast switches (see e.g., [4], [34]). The algorithms used in all of these implementations are based on cell scheduling rather than bandwidth reservation. Due to the complicated nature of multicast traffic, it is very hard to quantify the quality of service provided. This past work focuses on specific examples and presents simulations that illustrate the performance for these examples. To our knowledge, there are no bandwidth reservation based scheduling algorithms developed in the context of multicast.

The rest of this chapter is organized as follows. First, we give the problem model. Next, we will show that not all feasible multicast rates can be supported over such a switch without speedup, even if the crossbar constraint is relaxed to allow broadcast connections. Finally, we will evaluate the necessary speedup for strictly non-blocking scheduling for multicast support.

4.2 Model and Fundamentals

First, let us introduce some notation. In the unicast scenario, each cell at an input link has a single destination and cells that share the same I-O pair are placed in the same VOQ. In multicast, a cell does not necessarily have a single destination. We define a *class* of cells as those that share the same input and the same set of outputs. Without multicast, there would be N classes per input

part of this chapter, we will remove that condition and adapt a *broadcast enabled* version of the crossbar constraint.

link, and each class of cells would be stored in the same VOQ. In multicast, we assume that each class is kept in a separate per-class queue.

Let k_{ij} be the number of classes of cells which arrive at input i and have output j as one of the destinations. Let the first one of these be the unicast class[2] and the other $k_{ij} - 1$ be multicast classes. Also let the number of destinations and the rate for the lth such class be $n_{ij}(l)$ and $R_{ij}(l)$ respectively. The rate matrix, R, can be broken into a sum of matrices, one for the unicast and a number of others for the multicast classes. For instance, consider the following 3×3 system:

$$R = \underbrace{\begin{bmatrix} 0 & 0.2 & 0.5 \\ 0.6 & 0.4 & 0 \\ 0.1 & 0.1 & 0.4 \end{bmatrix}}_{1} + \underbrace{\begin{bmatrix} 0.2 & 0.2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{2} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}}_{3}$$

where the first matrix represents the rates of the unicast class and matrices 2 and 3 represent the rates for the two multicast classes. The multicast class cells whose rate is represented in the second matrix arrive at the first input link and are duplicated inside the crossbar. One copy is then sent to each one of the destinations, 1 and 2. The last multicast class has a rate of 0.1 and its cells are destined to all three output links. Hence, $k_{12} = 2$ and $n_{12}(2) = 2$. Note that for each multicast class, the rate matrix, R, has multiple entries increased by an amount identical to the rate of the class. For instance if a class has I-O pairs (1,2) and (1,3), its rate is added to both R_{12} and R_{13}. Hence, it may be the case that, for some i,

$$\sum_{j=1}^{N} \sum_{l=1}^{k_{ij}} R_{ij}(l) > 1 \tag{4.1}$$

That is, R is not necessarily doubly stochastic. If (4.1) holds for some i, then the cells of a multicast class arriving at input link i cannot be duplicated in the per-class queues at the input; otherwise, these queues will overflow. But the actual rate of cells at the ith input link is not the sum of the entries over the ith row. In the presence of admission control, the following holds:

$$\sum_{j=1}^{N} \sum_{l=1}^{k_{ij}} \frac{R_{ij}(l)}{n_{ij}(l)} \leqslant 1 \tag{4.2}$$

[2]By definition, there can be only one unicast class with the same input and output pair.

for all i. The summation on the left side of (4.2) is, indeed the actual rate of all the classes arriving at input link i. The rate of each class is divided by the number of destinations (fanout) of that class since they are represented that many times in the same row of the rate matrix. The admission control inequality for the jth output link is as follows,

$$\sum_{i=1}^{N} \sum_{l=1}^{k_{ij}} R_{ij}(l) \leqslant 1 \tag{4.3}$$

Thus, the columns of R must sum to no more than 1. Note that this time we did not divide the rate of each flow by the fanout of the class since each flow is duplicated inside the crossbar and each copy of a cell must be counted separately. In fact, there are as many as

$$\sum_{i=1}^{N} k_{ij}$$

classes where the set of destinations includes j, whereas there are

$$\sum_{j=1}^{N} \sum_{l=1}^{k_{ij}} \frac{1}{n_{ij}(l)}$$

classes at the ith input link. If we compare the total number of classes at all of the input links to that at all the output links, we observe that the former is never greater than the latter:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{l=1}^{k_{ij}} \frac{1}{n_{ij}(l)} \leqslant \sum_{j=1}^{N} \sum_{i=1}^{N} k_{ij}$$

This is plausible since cells are duplicated inside the crossbar.

4.3 Multicast Support is not Possible without Speedup

Suppose we have an $N \times N$ crossbar with broadcast capability, i.e., one input can be connected to multiple outputs at the same time, and a cell can be duplicated inside the crossbar and sent to multiple outputs simultaneously. With this assumption, the set of configurations is a superset of permutation matrices. A configuration can have multiple 1s in a row, but only a single 1 in each column. Even with this expanded set of configuration matrices, the set of all admissible rates

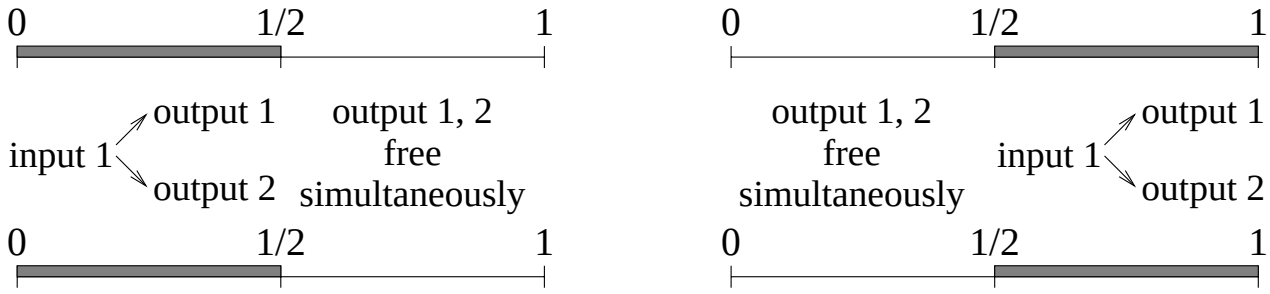

Figure 4-2: No matter where the multicast flow is served, both outputs 1 and 2 will be idle simultaneously.

cannot be supported by a crossbar in the presence of multicast. Consider the following rate matrix:

$$R = \begin{bmatrix} 0 & 0 & 0.5 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.4}$$

It can be easily seen that no input or output link is oversubscribed under this traffic. There is only one multicast class and it is at the first input link. The second input link is fully utilized since the sum of the second row entries, all of which are unicast rates, is 1. Therefore, at any point in time, input 2 must be connected to either output 1 or output 2, but not both since all the cells are unicast at the second input. On the other hand, input 1 needs to be connected to these two outputs simultaneously half the time to transfer multicast cells. This implies that these two outputs can be freed by the first input only half of the time as shown in Fig. 4-2, where the time period illustrated can be arbitrarily long. Thus, whenever the first input serves a multicast cell, input 2 must remain idle. However, since input 2 is fully utilized, it cannot remain idle.

The other alternative is that the multicast cells get duplicated at the input and transferred to outputs 1 and 2 separately. But, if the multicast cells are duplicated at the input, queues at this input will overflow since the total rate of cell arrivals exceeds 1. We conclude that R is not supportable, even though it is admissible.

This example illustrates that supporting multicast rates is in fact much more complicated than supporting unicast rates. If R were a unicast rate matrix, it would supportable if and only if it could be written as a convex combination of configuration matrices (i.e., permutation matrices for the crossbar without broadcast). If R is a multicast rate matrix, then this no longer holds. In the

example we just considered,

$$R = 0.5 \underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{1} + 0.5 \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{2}$$

both 1 and 2 are valid configurations in the crossbar with broadcast capability. However, even if we decomposed the rate matrix as a convex combination of configurations of the crossbar, it is not supportable by the crossbar as shown in the above example. Next, we will evaluate the necessary speedup for strictly non-blocking scheduling for all admissible multicast rates over the crossbar with broadcast capability.

4.4 Strictly Non-blocking Scheduling for Multicast Rates

In Chapter 3, we illustrated an analogy between the end to end configurations of an $N\tau \times N\tau$ three stage Clos network and the schedule of configurations to support τ-periodic unicast traffic over a single crossbar. Now, we will construct a similar analogy for τ-periodic multicast traffic over a broadcast enabled crossbar. Thus, we assume that the crossbar configurations for the single crossbar switch can have inputs connected to multiple outputs. Since there is a one to one correspondence between the configurations of the single crossbar switch and the configurations of the middle crossbars in the Clos network, they must also have the same property that an input to a middle crossbar can be connected to multiple outputs for the analogy to be valid. We represent such a Clos network as $\mathrm{C}_M(\tau, N, \tau')$ where the subscript, M, represents the multicast capability. The following theorem is the version of Theorem 2 for Clos networks with multicast.

Theorem 4.1 *Network $C_M(\tau, N, \tau')$ cannot be strictly non-blocking for all multicast connections between $N\tau$ output links of the input crossbars and $N\tau$ input links of the output crossbars unless $\tau' \geqslant \left(\tau \frac{\log N}{\log \log N}\right)$.*

Note that in network $\mathrm{C}_M(\tau, N, \tau')$, only the middle crossbars are broadcast enabled. The following result is shown in a recent paper by Yang, et.al. ([55]). In a three stage Clos network with broadcast enabled input and output crossbars as well as middle crossbars, the number of middle crossbars must be no less than $\tau \frac{\log N}{\log \log N}$ for a new multicast connection request between an idle input link and

a set of idle output links to be accommodated without a need to rearrange the existing (multicast) connections between other inputs and outputs. Theorem 4.1 does not make any assumptions for the input or output crossbars since it is only concerned with possible connections between the input crossbars and output crossbars through the middle crossbars, rather than between all the input links and the output links. However, it can be observed from the proof presented in [55] that this restriction makes no difference, and for any multicast connection between $N\tau$ output links of the input crossbars and $N\tau$ input links of the output crossbars to be possible, the necessary number of middle crossbars is no less than $c\frac{\log N}{\log\log N}$. Indeed, if all multicast connections between the input and the output crossbars can be made, it can be shown that any multicast connection between $N\tau$ input and $N\tau$ output links of the Clos network can be made as well. Hence, $c\frac{\log N}{\log\log N}$ is a necessary speedup in our scenario.

For a strictly non-blocking schedule to be possible for all admissible multicast traffic over a single crossbar, a speedup of $\mathrm{O}\left(\frac{\log N}{\log\log N}\right)$ is necessary. Unfortunately, to our knowledge, the version of Theorem 4.1 for rearrangable non-blocking networks is yet to be derived.

4.5 Conclusions

In this chapter, we studied the support of multicast rates over crossbar switches. We showed that, unlike unicast, multicast rates are not supportable over single crossbar switches without speedup. Then, we used the analogy between connections in a Clos network and crossbar switch schedules to show that a speedup of approximately $\mathrm{O}(\log N)$ is necessary for multicast support. This speedup is a limiting factor on the scalability of single crossbar switches with multicast support. In the next chapter, we will discuss a number of different architectures and algorithms by which multicast support is possible without a need for the speedup.

Chapter 5

Multistage Switch Architectures with Quality of Service and Multicast Support

5.1 Introduction and Motivation

The core of a packet switch is composed of a switch fabric and memory elements. The function of the fabric is to set up connections between the input and the output links. A very important class of fabrics is the non-blocking class. A fabric is non-blocking if a connection between an input and an output link, both of which are not already a part of other connections, can be set up. The most popular non-blocking fabric is the crossbar. It can be thought of as a set of lines, and crosspoints that connect these lines as illustrated in Fig. 5-1. The most important limitation of a crossbar is the so called "crossbar constraint:" At any point in time, only one input can be matched with an output, and only one output can be matched with an input[1]. For example, in Fig. 5-1, the first input is connected to the second output and hence no other connection can be made by the second output.

Despite the difference in their approaches to the problem of providing quality of service, the switch model considered in a majority of papers published in this literature is almost identical: Input buffers that are running at least at line speed (possibly faster depending on the speedup), a

[1]Note that, in this chapter, we assume a crossbar does not have broadcast capability.

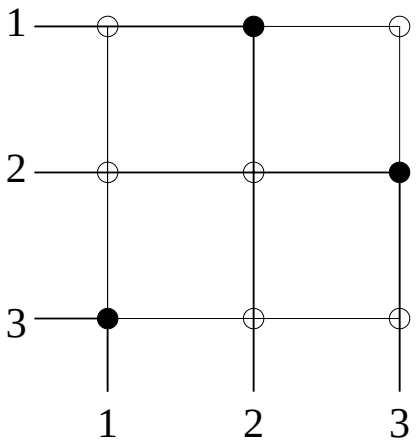

Figure 5-1: A 3×3 crossbar fabric. Crosspoints are set to connect the lines to enable end to end connections. For instance, input 1 and output 2 are connected through the corresponding crosspoint.

single crossbar fabric as the interconnect, output buffers which should run as fast as the fabric[2]. This model is common to almost all the papers published in this literature.

The main limitation of the schedulers in the classical architecture is the crossbar constraint. Due to this constraint, at the input of a crossbar switch, a packet competes not only with other packets that are destined to the same output, but also with those sharing the same input. Given that the connection fabric is a single crossbar, there is no way to avoid this constraint, which is the main difficulty in designing high performance scheduling algorithms. In the first two parts of this thesis, we studied such algorithms and associated complexities.

There are a number of studies on different architectural choices as well. In a recent work by Iyer et.al. [35], it is shown how to simply modify the classical architecture to make use of the extra capacity of the links when the memories run slower than the line rate. The basic idea is to divide each pipe into multiple, say k, pipes by means of demultiplexers and use k switches in the middle stage before multiplexing packets into a single link again.

Another interesting modification to the classical architecture is proposed by Stephens et.al. [36] to overcome the contention between the cells sharing the same input. Instead of keeping the virtual output queues at the input, they are pushed inside the crossbar next to their corresponding crosspoints as illustrated in Fig. 5-2. For instance, the queue at input i that holds the packets destined to output j is moved to the crosspoint, (i, j). When a packet arrives with an input-

[2]Even with no speedup, buffers at the output is desired in general since one may want to control the delay jitter of a flow at an output link rather than possibly sending its packets in bursts.

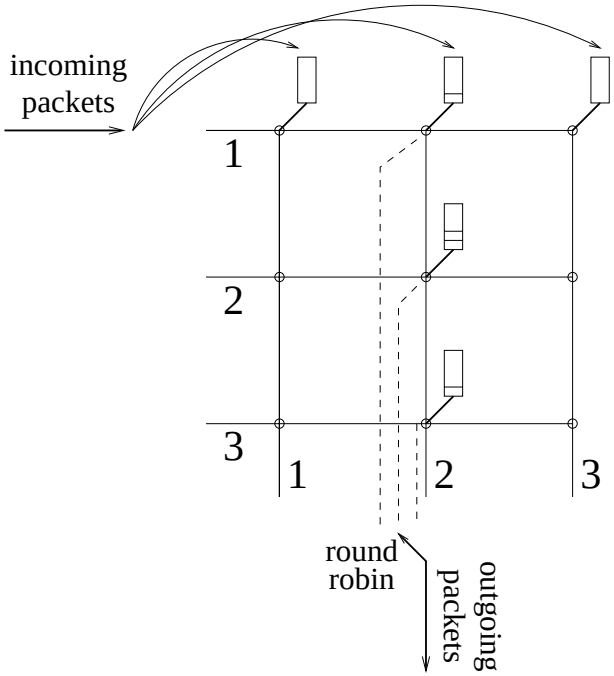

Figure 5-2: Virtual output queues are moved inside the fabric next to the corresponding crosspoints. Incoming packets are placed into these queues upon arrival. Each output link can then independently implement a round robin schedule among those queues which have packets destined to itself.

output pair, it is directly forwarded inside the crossbar fabric, placed into the buffer located at the corresponding crosspoint. This way, multiple crosspoints in the same input (e.g., $(i, j_1), (i, j_2), \dots$) can be connected simultaneously and an input can send packets simultaneously to different outputs since the queues are physically separate. Hence, packets with different destinations do not contend at the input for the crossbar, even though contention between packets destined to the same output is still not eliminated. Each output can separately apply round robin scheduling between the queues located at the crosspoints which connect the inputs to this output. The disadvantage of this architecture is that it is drastically different from the traditional crossbar and hard to manufacture. Also, since the N^2 buffers in an $N \times N$ crossbar are physically separate, the advantages associated with statistical multiplexing are lost. Note that, in the regular VOQ scheme, we do not need to keep a separate queue at each input to implement virtual output queueing. We implement VOQs by linked lists, i.e., we can store all the cells arriving at an input in a single buffer, and assign a pointer to each cell to keep track of the output it is destined to.

Turner ([38]) considered the Benes architecture for packet switching, and showed that 100% throughput can be achieved over a Benes architecture, given that the packets between an input-

output pair do not necessarily follow the same path. However, Turner did not focus much on the performance issues, such as packet delay and queue sizing.

In this chapter, we will present an in depth study of multistage switches. In the first part, we will develop the mathematical tools that we will use, build some intuition for the general structure of multistage switches and then propose a number of architectures along with associated routing and scheduling algorithms. In the second part, we analyze the quality of service provided by these algorithms. Our algorithms and architectures illustrate how the presence of multiple paths between input-output pairs can be exploited to improve the performance of a switch and simplify the scheduling algorithms. Also, we show that some of our architectures are capable of providing rate guarantees for all admissible multicast rates without speedup, which would not be possible with a single stage broadcast enabled crossbar switch as shown in Chapter 4.

5.2 Definitions and Model

We define an interconnection to be a structure which can provide a set of connection configurations or matchings between its input links and output links. In this section, we give some fundamental properties of interconnections connected in cascade. Each interconnection is also known as a *stage* and the system is called a *multistage switch* or a *network*. First we develop a model for interconnections and multistage switches, and give a brief introduction to *fluid techniques*. Then, we show how to use these tools to identify how the load is divided over intermediate stages as flows are routed through the set of interconnections.

5.2.1 An Algebra for Multistage Interconnections without Internal Queueing

We view a multistage switching system as a set of interconnections which are cascaded as follows. Let the mth interconnection, $\mathcal{N}_m$ have a size $I_m \times O_m$, and the total number of stages be M. The number of output links from a stage is equal to the number of input links to the following stage, i.e., $I_{m+1} = O_m$ for $m \leqslant M-1$. The input and the output links of each network are numbered such that ith input of $\mathcal{N}_{m+1}$ is connected to the ith output of $\mathcal{N}_m$. In all but one of the architectures we deal with in this chapter, $I_m = O_m$ for $m \leqslant M$. However, most of the results can be generalized to non-symmetric interconnections.

Every interconnection, $\mathcal{N}_m$, is modeled by a set of configurations, $\mathcal{C}_m$. The elements of these sets are matchings between the inputs and the outputs of the interconnection. Each such configuration

can be represented by a permutation matrix; indeed, the permutation matrix corresponding to a configuration has a one in each location for which an input-output pair is connected. We assume an interconnection is subject to the crossbar constraint, i.e., an input can be connected to only one output and an output can be connected to only one input at a time.

At any point in time, $\mathcal{N}_m$ can configured to only one element of $\mathcal{C}_m$, independent of the configurations of the other stages. In fact, the independent configurabililty is the defining property of a stage. When $\mathcal{N}_m$ is set to a $c \in \mathcal{C}_m$ and if $c_{ij} = 1$ for some i and j, input i and output j of $\mathcal{N}_m$ are connected.

Now, let us examine the system of two fabrics in cascade. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be configured to c_1 and c_2 respectively. If the system from the input of $\mathcal{N}_1$ to the output of $\mathcal{N}_2$ is visualized as a single *combined* interconnection, $\mathcal{N}_{1\to2}$, then the current configuration, $c_{1\to2}$, of this interconnection will be the permutation matrix,

$$c_{1\to2} = c_1 c_2$$

If $c_{1,il}$ and $c_{2,lj}$ are 1 simultaneously, then $c_{1\to2,ij}$ will also be 1. This can be justified physically as follows. If the first stage connects input link i to its output link l, and stage 2 connects its input link l to output link j, then input link i will be connected to output link j in the combined interconnection. This assumes no queueing between stages, so the output from one stage immediately gets in the input to the next.

The set of configurations for the combined interconnection is,

$$\mathcal{C}_{1\to2} = \{c | c = c_1 c_2,\ c_1 \in \mathcal{C}_1, c_2 \in \mathcal{C}_2\}$$

Let $|\cdot|$ represent the cardinality of a set. Note that,

$$\max\{|\mathcal{C}_1|, |\mathcal{C}_2|\} \leqslant |\mathcal{C}_{1\to2}| \leqslant |\mathcal{C}_1|\,|\mathcal{C}_2| \tag{5.1}$$

where the lower bound follows since, if the configuration of one of the interconnections is kept fixed, $|\mathcal{C}_{1\to2}|$ will be equal to the cardinality of the set of configurations for the other interconnection, and the upper bound follows since a pair of configurations, (c_1, c_2), defines a unique end to end configuration (although a single end to end configuration might be generated by many pairs).

Thus, we showed that the total number of end to end configurations may or may not increase

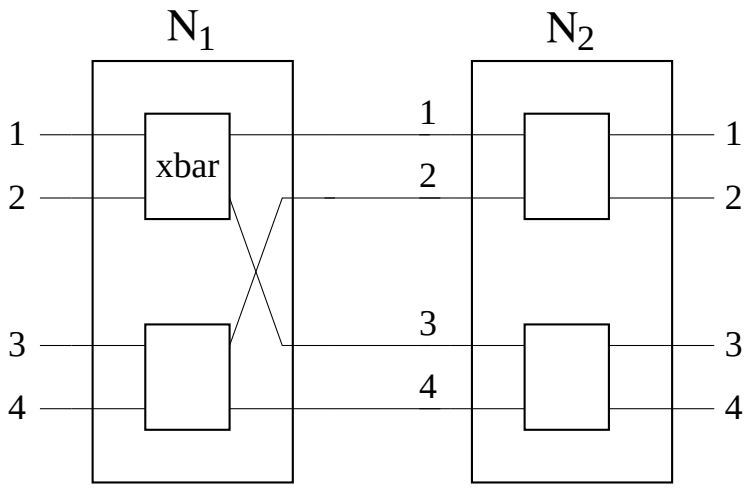

Figure 5-3: The two stage Banyan architecture composed of 2×2 crossbars is illustrated.

by cascading interconnections. If it does, it can increase at most exponentially with the number of stages. There are different implications of either case. It may even be desirable that the number of end to end configurations remain constant as interconnections are cascaded. We will discuss these issues after illustrating the definitions in the following examples.

Example 5.1 *Consider the 4×4 Banyan switch given in Fig. 5-3. Each 2×2 unit shown in the figure is a crossbar. For this network,*

$$\mathcal{C}_1 = \left\{ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \right\}$$

$$\mathcal{C}_2 = \left\{ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \right\}$$

For any pair of configurations, $c_1, c_1' \in \mathcal{C}_1$ and $c_2, c_2' \in \mathcal{C}_2$ the inequality, $c_1 c_2 \neq c_1' c_2'$ holds whenever

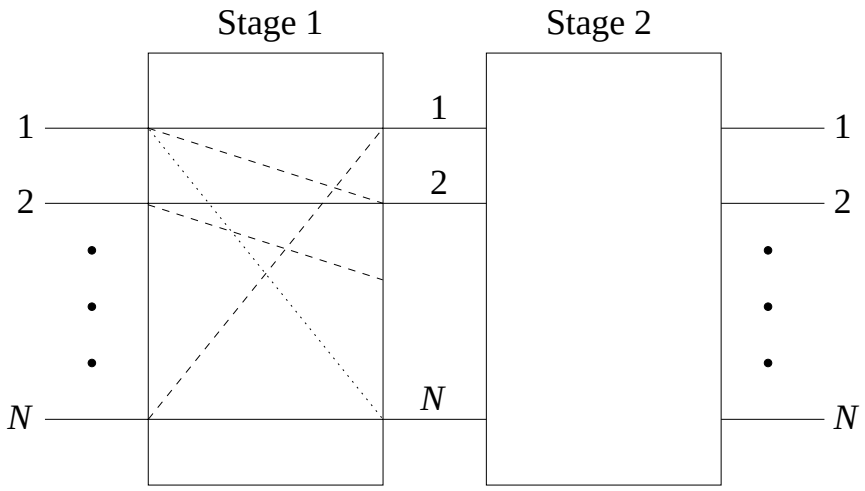

Figure 5-4: The two stage system consists of two cascaded switches each of which supports only the identity connection and $N-1$ cyclic shifts of it.

$c_1 \neq c_1'$ *or* $c_2 \neq c_2'$. *Thus, the upper bound given in (5.1) is satisfied with equality:*

$$|\mathcal{C}_{1\to 2}| = |\mathcal{C}_1|\,|\mathcal{C}_2|$$

Example 5.2 *Consider the two stage cyclic shift system given in Fig. 5-4. Each stage is identical and can support only N configurations, one of which is the identity configuration (the connection matrix is the identity) and the other $N-1$ are the cyclic shifts of it. Hence $\mathcal{C}_1 \equiv \mathcal{C}_2$.*

For any two configurations, $c_1 \in \mathcal{C}_1$ and $c_2 \in \mathcal{C}_2$,

$$c_1 c_2 \in \mathcal{C}_1$$

Thus, the lower bound given in (5.1) is satisfied with equality:

$$\begin{aligned} |\mathcal{C}_{1\to 2}| &= \max\{|\mathcal{C}_1|, |\mathcal{C}_2|\} \\ &= |\mathcal{C}_1| \\ &= N \end{aligned}$$

The presence of multiple paths between the same input-output (I-O) pair is a property that multistage switches posess, but single stage switches do not. This property gives the switch some freedom on how to route incoming packets between a given I-O pair. For instance, the second stage in the previous example (Example (5.2)) does not increase the number of end to end configurations,

and therefore it may seem redundant. However, as will be shown, it increases the number of paths between every I-O pair by a factor N and this property will be used extensively in our routing algorithms.

Now, we evaluate the number of paths between input-output (I-O) pairs in a multistage switch. Suppose we have an M stage system, $\mathcal{N}_1, \ldots, \mathcal{N}_M$. The number of paths between input i and output j is identical to the number of distinct M-tuples, $(c_1, \ldots, c_M)$, where $c_{1\to M,ij} = \left(\prod_{\mu=1}^{M} c_\mu\right)_{ij} = 1$. Instead of searching over all possible M-tuples, we can evaluate the number of paths between all input-output pairs using matrix algebra as follows.

For any two permutation matrices, c, c', let us define $\vee$ to be the entry-wise "logical OR" operation. That is, $(c \vee c')_{ij} = 1$ if $c_{ij} = 1$ or $c'_{ij} = 1$. The number of paths between input i and output j of the M stage system is the corresponding entry of the matrix,

$$L^{(M)} = \prod_{m=1}^{M} \bigvee_{c \in \mathcal{C}_m} c \tag{5.2}$$

This can be proved by induction. For a single stage system, L_{ij} is 1 if input i and output j can be connected; otherwise it is 0 (hence L_{ij} is the number of paths between i and j). Suppose Eq. (5.2) holds for an $M-1$ stage system and the number of paths between I-O pair (i, l) of the $M-1$ stage system is the corresponding entry of,

$$L^{(M-1)} = \prod_{m=1}^{M-1} \bigvee_{c \in \mathcal{C}_m} c$$

Let us put another interconnection in cascade at the end of these $M-1$ stages, so that it is the Mth interconnection in the overall system. For the M stage system, the number of paths between input link i and output link j is equal to the number of paths between the I-O pairs of the form $(i, \cdot)$ of the $M-1$ stage system that can be extended to the jth output of the M stage system. Namely, it is the sum of the each entry of the ith row of $L^{(M-1)}$, for which there is a possible path

to the jth output of the system through the corresponding input of the Mth stage. Hence,

$$\begin{aligned} L^{(M)} &= L^{(M-1)} \left(\bigvee_{c \in \mathcal{C}_M} c \right) \\ &= \left(\prod_{m=1}^{M-1} \bigvee_{c \in \mathcal{C}_m} c \right) \left(\bigvee_{c \in \mathcal{C}_M} c \right) \\ &= \prod_{m=1}^{M} \bigvee_{c \in \mathcal{C}_m} c \end{aligned}$$

and the proof is complete.

Using (5.2), for the Banyan network given in Example 5.1, the number of end to end paths can be found to be,

$$\begin{aligned} L^{\text{Banyan}} &= \left(\bigvee_{c \in \mathcal{C}_1} c \right) \left(\bigvee_{c \in \mathcal{C}_2} c \right) \\ &= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{aligned}$$

Hence, there is exactly one path between any input-output pair. Next, consider a Benes network:

Example 5.3 *One way to build a Benes network is by cascading two Banyan networks which share one common stage. A 4×4 Benes network is illustrated in Fig. 5-5. The two Banyan networks share one common stage as shown in this figure. The number of paths between input and output*

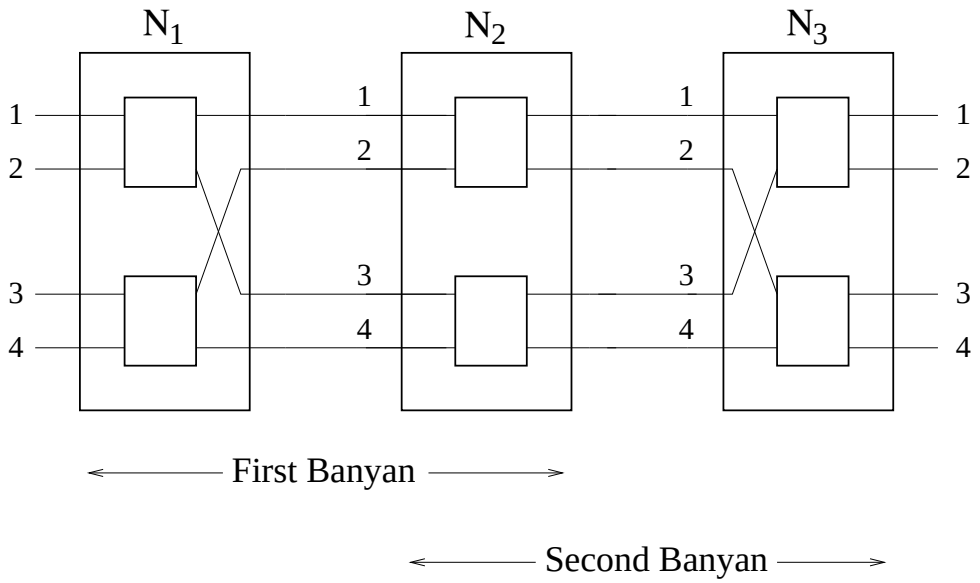

Figure 5-5: The two Banyan networks, $(\mathcal{N}_1, \mathcal{N}_2)$ and $(\mathcal{N}_2, \mathcal{N}_3)$ share $\mathcal{N}_2$ and form a Benes network.

links can be found similarly:

$$L^{Benes} = L^{Banyan}\left(\bigvee_{c \in \mathcal{C}_3} c\right)$$

$$L^{Benes} = L^{Banyan}\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

In general, for an $N \times N$ Benes network,

$$L^{\text{Benes}} = \frac{1}{2} N \vec{e} \vec{e}^{T}$$

where the T in the exponent represents matrix transpose and $\vec{e}^{T} = [1 \cdots 1]_{1 \times N}$. That is, there are $\frac{N}{2}$ paths between each I-O pair in an $N \times N$ Benes network.

Finally, consider Example 5.2. The L matrix for the architecture given in Fig. 5-4 can be found to be

$$\begin{aligned} L &= \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}^2_{N \times N} \\ &= N \vec{e} \vec{e}^T \end{aligned}$$

It may seem surprising that this architecture supports only N configurations as opposed to $N^{N/2}$ of the Banyan network (which is composed of 2×2 crossbars), but it has N paths between any input and output link pair whereas a Banyan network has only 1. In fact, this is plausible: The second interconnection does not increase the number of end to end configurations, but it does increase the number of paths between any input-output pair since an input can reach any output through any input of the second interconnection. We will exploit the property that the number of end to end paths of this architecture scales with N in quite a few of our switch designs later.

Similarly, each added stage to a Banyan network increases the number of end to end configurations at a high rate until it reaches the number of $N \times N$ permutation matrices, $N!$. This point is reached at stage $2 \log N - 1$ (i.e., Benes network), and the number of end to end configurations cannot increase any further. After this, with each additional stage, the number of end to end paths increases geometrically, doubling at each stage.

5.2.2 Traffic Contracts

So far, we have considered interconnections and multistage switches. Next, we introduce the traffic model and the main ideas of the routing and scheduling algorithms.

First, let us incorporate the time dimension. We assume that end users generate packet traffic at each input. At the input of a switch, these packets are partitioned into equal size cells. We assume that the interconnections are capable of changing their configurations and operate synchronously. Recall that we defined a service time slot as the period in which a cell can be transferred through an interconnection. Without internal queueing, a cell can be moved through the entire multistage switch within a time slot. Even though it is assumed that an interconnection can switch from one configuration to another in zero time, once set, a configuration remains for the period of time until a cell is transferred before it can be changed. Cell transfers are done synchronously over all the

inputs. In this chapter, we simply use a time slot instead of a service time slot. Unless mentioned otherwise, we assume no speedup, and hence, it takes a time slot to transmit a cell over a typical link.

There are two major alternatives for the *service* agreement between users and a network: connection oriented or connectionless. A connection oriented contract has a duration (in number of time slots) associated with it. Also, the number of cells that can be transferred between every I-O pair within the lifetime of the contract is specified. Thus, a rate, R_{ij}, can be associated with the input-output pair (i, j) as the ratio of the number of cells to be transferred between the pair to the lifetime of the contract. Thus, the rate, R_{ij} takes on values from the set $[0, 1]$. An admission controller makes sure that no input or output link is oversubscribed, i.e., $\sum_i R_{ij} \leqslant 1$ for all j and $\sum_j R_{ij} \leqslant 1$ for all i. Succinctly, we can represent R_{ij} as the (i, j) entry of a matrix R. The two admission control inequalities imply that R is a doubly sub-stochastic matrix. It was shown by von Neumann ([24]) that for every doubly sub-stochastic matrix, R, there exists a doubly stochastic matrix, Q such that $Q_{ij} \geqslant R_{ij},\ \forall i, j$. An algorithm that generates the doubly stochastic matrix is also given in [24]. In this chapter, we assume that the rate request matrix, R, is doubly stochastic.

In connectionless service, no rates are specified by the users. Congestion avoidance and control mechanisms are necessary to prevent buffers inside the network from overflowing. The service provided by the switches is best effort and only very limited quality of service guarantees (mostly probabilistic) can be given. We assume connection oriented service agreements throughout this chapter. It is only with connection oriented service that rate reservation based scheduling algorithms can be developed since such algorithms require the prior knowledge of the desired rates between I-O pairs.

Traffic Contracts without Internal Buffering

A switch is responsible for providing the desired number of service opportunities between each I-O pair. If the lifetime of a contract is T, then the switch will go through T configurations and at least TR_{ij} of them should connect input i to output j for the terms of the contract to be met. Suppose, for some switch schedule, the switch goes through configurations such that input i and output j are connected for $D_{ij}(t)$ time slots, by time $t \leqslant T$. We call the difference, $tR_{ij} - D_{ij}(t)$, the service lag for I-O pair (i, j) at time t. In many cases, some of the I-O pairs have positive service lag at the maturity $(t = T)$ of a contract. It may even be impossible to have all non-positive service lags.

In such cases, it is desirable that at least the service lag is upper bounded, i.e., does not increase with the duration, T.

The above arguments translate into our setting of multistage switches without internal buffers as follows. Suppose at some point in time, input i is connected to output j in an M-stage multistage switch. This will provide a service opportunity for a cell that lies at the corresponding virtual output queue (VOQ) to be transferred. Thus, for an agreement with a rate matrix R and a contract duration T to be fulfilled, the configurations of the interconnections, $c_1(t), \ldots, c_M(t)$, in period $(0, T]$ must satisfy the following:

$$\begin{aligned} TR &\leqslant \sum_{t=1}^{T} \prod_{m=1}^{M} c_m(t) \\ &= \sum_{t=1}^{T} c_{1 \to M}(t) \end{aligned} \tag{5.3}$$

where $\leqslant$ is entrywise.

Long Term Contracts

Definition 5.1 *A contract, (R, T), is called* supportable *if there exists a schedule of multistage switch configurations for the corresponding infinite duration contract, (R, ∞) for which the service lag remains upper bounded for all I-O pairs and for all t.*

If a schedule and a time t exist such that $tR_{ij} - D_{ij}(t) \leqslant 0$ for all pairs (i, j), then we say that the rate matrix R is perfectly supportable *at time t.*

If R is perfectly supportable at time t, then it is also perfectly supportable at times kt, $\forall k \in \mathbb{Z}^+$ since the switch can implement a periodic schedule which repeats itself every t seconds. By definition, supportability is relevant only for infinite duration contracts. Thus, instead of saying "the infinite duration contract with rate matrix R is supportable," we simply use, "R is supportable." Given R is supportable with the schedule $D(t), t > 0$,

$$\lim_{t \to \infty} \frac{D_{ij}(t)}{R_{ij} t} = 1$$

In an M-stage switch, for (5.3) to hold, R must be perfectly supportable at time T. Perfect

support may not be possible, even for admissible traffic. Let

$$D(t) = \sum_{\tau=1}^{t} c_{1\to M}(\tau) \tag{5.4}$$

For a multistage switch with no internal buffers, the set of rates R is supportable if there exists a set of configurations, $c_1(t), \ldots, c_M(t)$ such that $tR_{ij} - D_{ij}(t)$ is upper bounded for all t.

Theorem 5.1 *A contract with the rate matrix R is supportable by a multistage switch with no internal buffers if and only if R can be written as a convex combination of $c_{1\to M} \in \mathcal{C}_{1\to M}$.*

Proof: If there exists a set of non-negative coefficients, $\{\phi_1, \ldots, \phi_{|\mathcal{C}_{1\to M}|}\}$ such that

$$R = \sum_{l=1}^{|\mathcal{C}_{1\to M}|} \phi_l c_l \tag{5.5}$$

then, it was shown in [15] that the service lag remains upper bounded at all times using a packetized processor sharing schedule of the configurations with non-zero coefficients in the decomposition.

Conversely, suppose R is supportable. Then, there exists a scheduler and some $B < \infty$ such that $tR_{ij} - D_{ij}(t) \leqslant B$ for all pairs (i, j) and for all t. Hence,

$$\lim_{t\to\infty} [tR - D(t)] \leqslant B\vec{e}\vec{e}^{T} \tag{5.6}$$

Since $D(t)$ is a sum of t permutation matrices as given in Eq. (5.4) and B is not a function of t, (5.6) can be written as,

$$R = \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=1}^{t} c_{1\to M}(\tau) \tag{5.7}$$

The right side of the above equation (5.7) is a convex combination of at most $|\mathcal{C}_{1\to M}|$ permutation matrices, completing the proof.

5.3 Internal Buffers and Supportable Rates

Note that the formulation presented in the previous sections has the inherent assumption that a cell is transferred from an input to an output through the fabric without being kept in the intermediate stages. That is, no buffers are present inside the multistage switch between interconnections. The

converse (only if) part of Theorem 5.1 is only valid if there are no buffers inside. Indeed, Theorem 5.1 specifies the necessary and sufficient conditions for a given set of rates to be supportable by a multistage switch that has no buffers between its interconnections. Next, we raise the question of what happens if there are buffers between interconnections. We will present an example in which buffers inside a multistage switch expand the set of supportable rates.

5.3.1 An Example

Let $N = 3$. The following rate matrix with service duration, $T = 3$ is to be supported over the two stage cyclic shift architecture.

$$\frac{1}{3}\begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Each switch supports only the following 3 configurations:

$$c_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, c_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, c_3 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Note that, without buffers, one of the switches is redundant from circular symmetry. Obviously, R is not supportable in this scenario since it cannot be written as convex combination of c_1, c_2 and c_3.

If there are buffers at the input of the second stage, we have more freedom on which cell to schedule through the first stage since they do not have to leave the system within the same time slot. We can exploit this freedom to serve matrix R as illustrated in Fig. 5-6. In this figure, a schedule of switch configurations with which the set of rates can be supported is shown. The same schedule is also described in the following paragraph in detail. The important thing to note is that two cells with the same destination arriving at the input of the second switch at different times must be placed in a queue for a later transfer. For instance, in the second time slot, two cells both of which are destined to output 1 are transferred to the second interconnection. The one at the second input of this interconnection is queued for later trasfer. Also in the same time slot, no cells are transferred from the second input of the second stage to the output. Thus, it may seem that the given rates

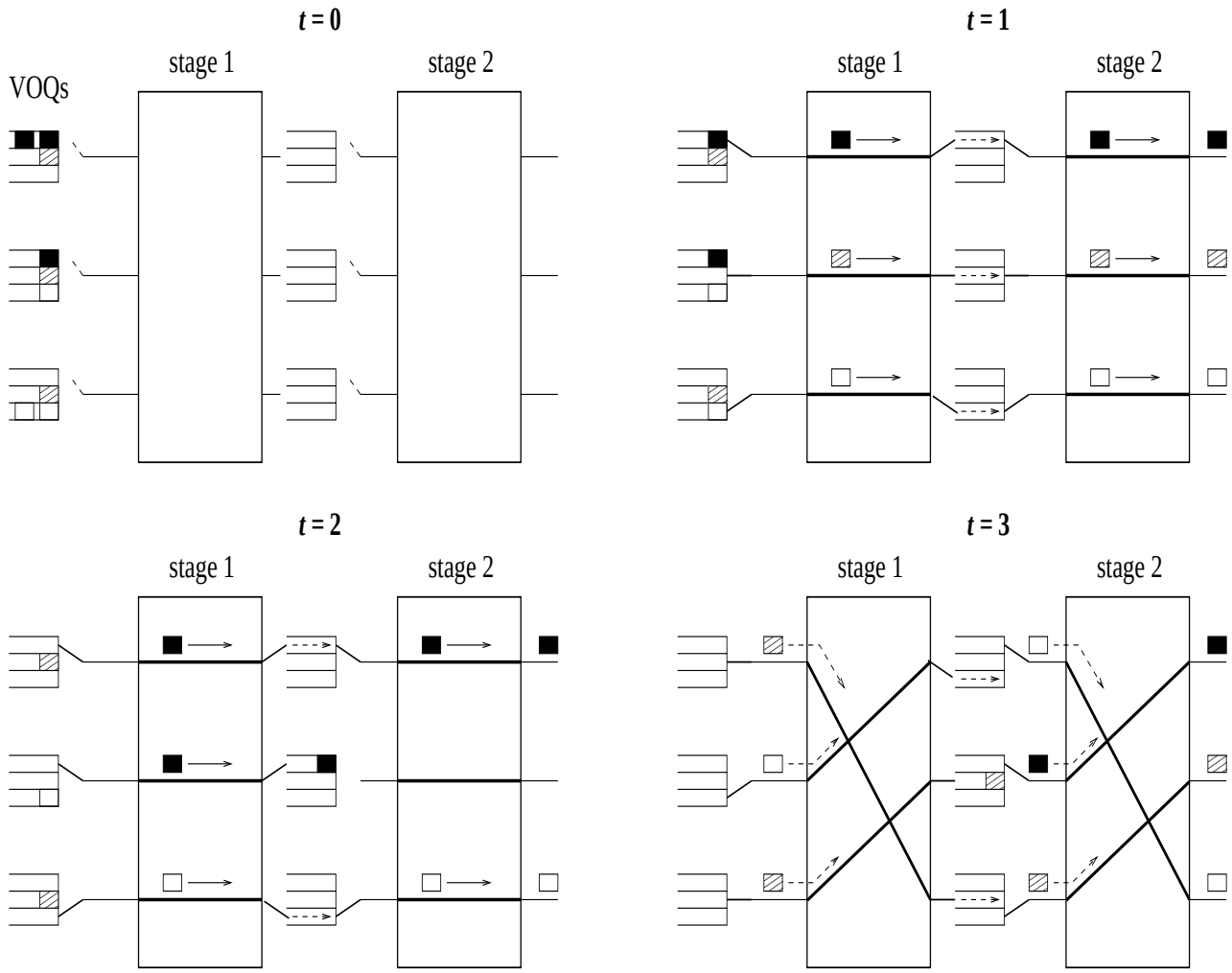

Figure 5-6: The evolution of the system in period [0,3] is illustrated. We assume that a cell can be transferred from the input queues to the output link in a time slot. In the second time slot, no cells are transferred from the second input of the second stage to the output, but if the same schedule of configurations is repeated all over, all the output links will be fully utilized.

cannot be supported due to cell accumulation if the same schedule of configurations is repeated. However, in the second time slot of the second cycle, there will be a cell to be transferred from the second input of the second stage to the output. Hence, all the output links will be fully utilized.

Description of the Schedule: In the first time slot, let c_1 be in effect in the first interconnection. Suppose, a service opportunity is given to the cells with I-O (ultimate destination) pairs, (1,1), (2,2) and (3,3) to be transferred to the input of the second interconnection. Let the same matrix, c_1, be kept in the second time slot; but this time let a cell with I-O pair, (2,1), be transferred from input 2 to output 2 of the first switch instead of that with (2,2), along with those cells with I-O pairs (1,1) and (3,3). At the end of the second time slot, either the cell with the I-O pair (2,1) , or that with the I-O pair (1,1) must be enqueued at the input of the second switch, since both cannot be transferred to their ultimate destinations simultaneously. In the third time slot, let c_3 be in effect to complete a cycle. In this time slot, let cells with I-O pairs (1,2), (2,3) and (3,2) be awarded with a service opportunity. These cells will get to the inputs 3, 1 and 2 respectively according to c_3. The time slots that service opportunities are awarded by the first switch are illustrated in the following matrix, $T^{(1)}$.

$$
\begin{array}{c} \\ \text{I1} \\ \text{I2} \\ \text{I3} \end{array}
\begin{array}{c}
\begin{array}{ccc} \text{O1} & \text{O2} & \text{O3} \end{array} \\
\left[\begin{array}{ccc} (1,2) & (3) & (-) \\ (2) & (1) & (3) \\ (-) & (3) & (1,2) \end{array} \right]
\end{array}
$$

where the (i, j) entry represents the slots during which i to j transfer is made on the first switch. For instance, if the (i, j) entry is (1,2), then cells with I-O pairs, (i, j) get scheduled in time slots 1 and 2. We call such matrices, schedule matrices. Entries with '-' require no service opportunities. Given that the first stage goes through the sequence of configurations c_1, c_1, c_3 with the schedule, the load at the input of the second stage can be computed as follows.

Input 1 Two of the cells are destined to output 1 and the other is destined to output 3.

Input 2 Two of the cells are destined to output 2 and the other is destined to output 1.

Input 3 Two of the cells are destined to output 3 and the other is destined to output 2.

The set of rates that the second interconnection is loaded with is thus,

$$R^{(2)} = \frac{1}{3}\begin{bmatrix} 2 & 0 & 1 \\ 1 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

which can be supported using c_1 and c_3 only. Therefore, if the second switch keeps c_1 in effect in the first two time slots and c_3 in effect in the third and repeats this periodically, R can be supported. Notice that there is always a cell in a buffer, either at input 1 or input 2 at the buffers of the second stage interconnection. Hence the service lag for one of the I-O pairs will be 1, while the others are 0.

This example illustrates that, in multistage switches, there exist sets of rates that are supportable in the presence of internal buffers, but not supportable without them. In particular, in this 2-stage example, without buffers at the input of the second interconnection, one of the cells must be dropped[3]. Buffers give us freedom to transfer two cells from the first stage to the second simultaneously, even if their ultimate destinations are the same. Finally note that, in the above example, each flow followed a unique path and thus no reordering of cells was necessary at the output of the switch.

5.3.2 Infinitely Divisible Traffic Model

In this section, we study a number of routing and scheduling algorithms for multistage switches. We will use "infinitely divisible fluids" to model the traffic. Fluid flows are extremely helpful in this work since they can be modeled by only one parameter, their rate, and as shall be shown, we can perfectly specify the traffic at each stage of a multistage switch using the input rate matrix and the set of configurations that each interconnection goes through. The fluid model eliminates the need to worry about queueing and cell scheduling which could complicate things highly as shown in the previous section.

The framework of fluid models has been used extensively in the literature (e.g., [48],[49]), mostly to obtain stability regions (set of supportable rates) of stochastic networks with very mild assumptions on input traffic. Similarly, we will use them to find the region of supportable rates

[3]In fact, it can be shown that a speedup of 3 is necessary to support all admissible traffic for this system without buffers.

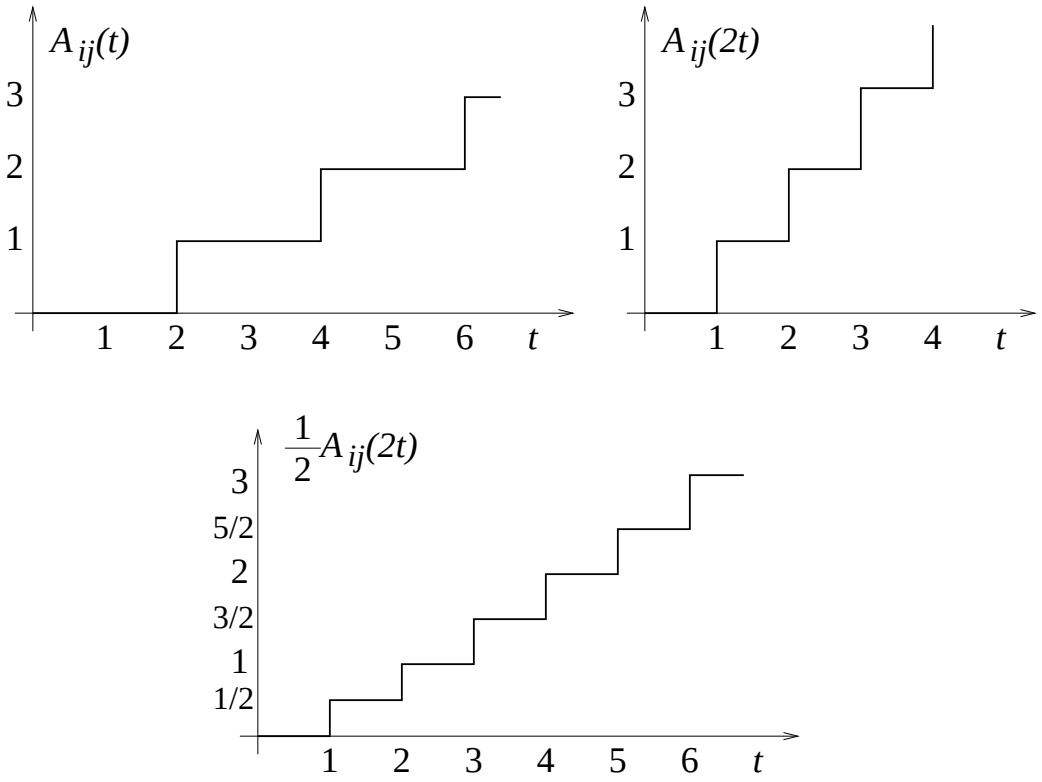

Figure 5-7: The graph on the top left is a typical (rate controlled) arrival process. In the top right graph, time axis is scaled with a factor of 2, i.e., $t' = 2t$. In the bottom one, both time and the magnitude is scaled with the same factor, $\alpha = 2$. As α is increased, the broken line starts to look like a tilted straight line.

of the multistage switches with the routing and scheduling algorithms we design. First, we give an introduction to fluid models, and study how such flows behave as they propagate through the interconnections of a multistage switch.

Traffic Flows

Let $A_{ij}(t)$ denote the cumulative number of cells that have arrived at VOQ (i, j) by time t. We assume that $A_{ij}(0) = 0$ and suppose the cell arrival processes, $\{A_{ij}(\cdot), 1 \leqslant i, j \leqslant N\}$ satisfy the following for all pairs, (i, j),

$$\lim_{t\to\infty} \frac{A_{ij}(t)}{t} = R_{ij} \tag{5.8}$$

We already defined $D_{ij}(t)$ to be the number of configurations for which input i is connected to output j by time t. Given that (5.8) holds, we can adopt the fluid model by scaling time and space

simultaneously as follows[4].

Suppose, we scale the size of a time slot and the cell size down by a factor, α, keeping everything else the same. Recall that a service time slot is the time it takes a cell to be transferred through an interconnection. In the new system, if we examine the number of arrivals, we can observe that it goes through the same sequence as before, except that the changes occur α times as fast as the original system as illustrated in the second graph given in Fig. 5-7. Hence, a set of rates is supportable in the original system if and only if it is supportable in the scaled version. Next, suppose we scale the ordinate by the same factor, α. As the scaling parameter, α is increased to very large values, the arrivals occur almost "continuously" and each arrival will cause an infinitesimal change in the state of the system, and in the limit,

$$\lim_{\alpha \to \infty} \frac{1}{\alpha} A_{ij}(\alpha t) = R_{ij} t \tag{5.9}$$

This limiting case is called the *fluid limit*. In this scenario, the traffic between each I-O pair will be called a flow, and each flow can be modeled only with its rate. For instance, let $A_{ij}(t)$ be the accumulated amount of flow between I-O pair (i, j) at time t. Then,

$$A_{ij}(t) = R_{ij} t \tag{5.10}$$

for all $t > 0$. We will be dealing with flows in the rest of this section. A crucial assumption that we make is that the flows are infinitely divisible. A flow can be divided into arbitrarily many subflows, the sum of whose rates is that of the initial flow.

The interconnections also operate in the fluid limit; thus $D_{ij}(t)$ takes on values from a continuous set. For instance, a switch which is composed of an interconnection can continuously alternate among arbitrarily many configurations for all $t > 0$, and the matrix, $D(t)$, will be the weighted sum of the corresponding permutation matrices. Hence, $\frac{1}{t} D(t)$ can be any convex combination of the configurations of the switch, for all $t > 0$. As the cell size gets reduced by some factor, service lag is also cut proportionally[5]. Therefore, in the fluid limit, for a supportable rate matrix, the service lag is 0 for all I-O pairs and for all t. In that sense, the service lag can be viewed as a metric for

[4]See [50] for a more detailed treatment of the transformation. It was shown that it can be generalized to all random arrival processes that satisfy the strong law of large numbers. In that case, Eq. (5.8) holds with probability 1.

[5]If the cell size is reduced by some factor, the service lag remains the same in number of cells. However, it gets reduced by that factor in number of bits.

the difference between the service provided by a certain scheduler and that of the corresponding (same set of rates) flow scheduler.

The above paragraphs lead to the following. Suppose a given rate matrix, R is supportable by a packet switch. Then, the following holds for the matrix, $D(t)$, of service opportunities provided by time t:

$$[tR - D(t)] \leqslant B\vec{e}\vec{e}^T$$

for some $B < \infty$ and all $t > 0$. Scaling the size of a time slot and the cell size down by a factor, α, we get

$$\begin{aligned} \frac{1}{\alpha}[\alpha t R - D(\alpha t)] &= tR - \frac{1}{\alpha}D(\alpha t) \\ &\leqslant \frac{1}{\alpha}B\vec{e}\vec{e}^T \end{aligned}$$

Hence, in the fluid limit, for all $t > 0$,

$$\frac{1}{t}D(t) = R \tag{5.11}$$

We just showed that a packet switch cannot support a contract with a rate matrix R, unless the flow traffic with the matrix of rates R is supportable in the corresponding flow switch. This will be our main motivation for looking into flow switches and developing routing algorithms for flow traffic. Then, we will transform these algorithms to the packet switching world.

Eq. (5.10) and (5.11) imply that at any point in time $A(t) = D(t)$. Thus, the required buffer size approaches to 0 in the fluid limit. This may mislead us to an incorrect conclusion that the set of supportable rates with internal buffering is identical to that without internal buffering. However, as illustrated by an example in the previous section, internal buffering expands the set of supportable rates. The critical point here is the following. Let β be the size of internal buffers and α be the scaling parameter. The following can be written for $\rho(\alpha, \beta)$, the region of rates supportable by a multistage switch, for a given (α, β) pair:

$$\lim_{\alpha \to 0} \lim_{\beta \to 0} \rho(\alpha, \beta) \neq \lim_{\beta \to 0} \lim_{\alpha \to 0} \rho(\alpha, \beta) \tag{5.12}$$

We dealt with the left side of (5.12) when we studied the case without internal buffering. We

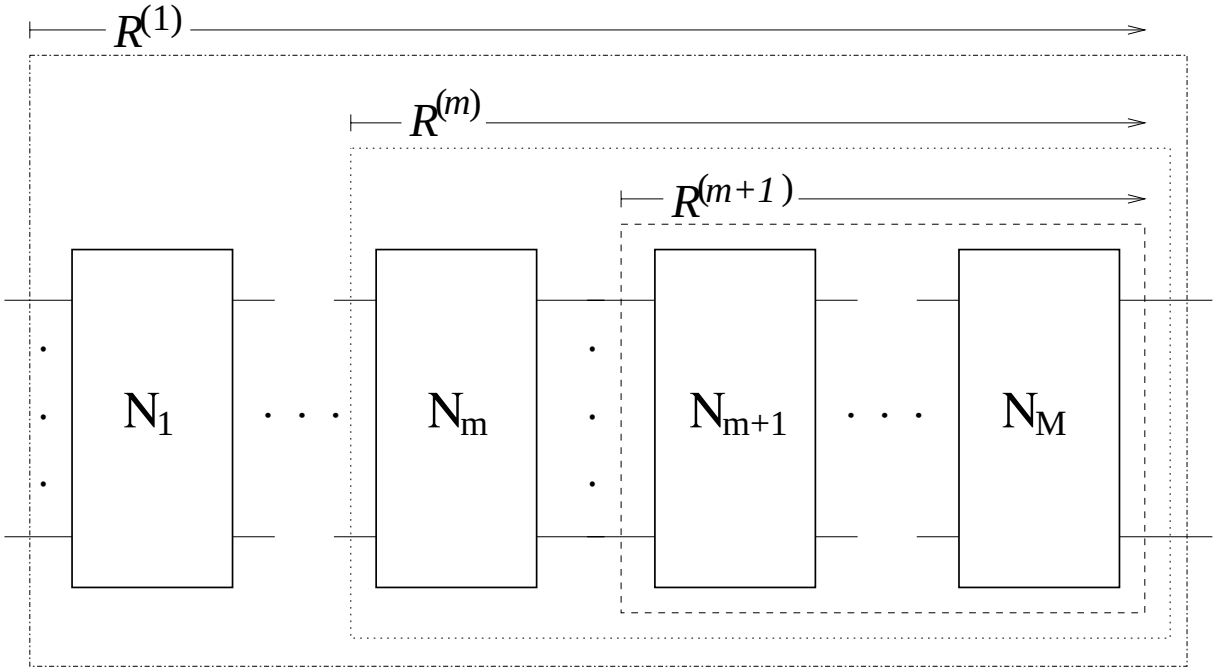

Figure 5-8: The mth multistage rate matrix is defined to be from the input of the mth stage to the output.

showed that the supportable rates lie the in the convex hull of the supportable configurations by the switch. On the other hand, in this section we studied the fluid approximation, $\alpha \to 0$. As shown in (5.11), necessary buffer size approaches 0 in the fluid limit as a by-product of the flow approximation. The region of rates depends on which limit is taken first.

Flows over Multistage Switches

In this section, we will evaluate the rates of the flows inside a multistage switch as a function of the rates at the input, and the set of configuations that the interconnections go through. We start by defining *multistage rate matrices*. Conventionally, entries of a rate matrix are defined to be end to end, i.e., from the input links to the output links of the system. For an M-stage switch, we will define M rate matrices -one per stage- and talk about their relation.

Let the entries of the mth matrix, $R^{(m)}$ represent the fraction of time that the input links of the mth interconnection need to be connected to the output links of the final interconnection (hence the system) as illustrated in Fig. 5-8. For instance, $R^{(1)}$ is the conventional (end to end) rate matrix, i.e., $R^{(1)}_{ij}$ is the fraction of the time that input i should be connected to output j.

Suppose the first interconnection is configured to some $c_1 \in \mathcal{C}_1$. Then, $R^{(2)}$ can be found as,

$$R^{(2)} = c_1^T R^{(1)} \tag{5.13}$$

Eq. 5.13 can actually verified by inspection. Suppose, the interconnection connects the kth input to the lth output, i.e., $c_{1,kl} = 1$. When multiplied with $R^{(1)}$ from the left, the lth row of the product, $R^{(2)}$, will be identical to the kth row of the original matrix, $R^{(1)}$. This is plausible, since whatever is in the kth input link of the first interconnection will now be at the lth input link of the second interconnection. Thus, the subsystem which is composed of stages $2, \ldots, M$ will have the rate matrix, $R^{(2)}$, at its input. We conclude that, $R^{(m)}$ is a function of the configurations of the first m interconnections and $R^{(1)}$.

Another interpretation of the above arguments is the following. Let us define $R^{(1)}$ as the rates of the (fluid) flows at the input of the multistage switch. For instance, $R_{ij}^{(1)}$ is the rate of the flow arriving at the ith input and destined to the jth output of the multistage switch. Then, if the first interconnection is set to a constant configuration, c_1, the rates of the flows at the input of the second interconnection, $R^{(2)}$ can be found using Eq. 5.13.

With this interpretation, we can extend Eq. 5.13 to the case where interconnections alternate among different configurations rather than remaining fixed. Suppose the mth interconnection in a multistage switch is configured to spend a fraction, π_k, of time in $c_m^{(k)}$, for $1 \leqslant k \leqslant K$. As the configuration of the interconnect alternates, the flows at the input are scheduled such that those arriving at input i are divided into K subflows and the kth subflow is sent to the jth output link of the fabric whenever $c_{ij}^{(k)} = 1$. The rate of the kth subflow is π_k of the total rate of the flow. In the fluid limit, the interconnection can provide this convex combination in any time span. With this described flow routing and scheduling, there is a relation between $R^{(m)}$ and $R^{(m+1)}$, similar to the single configuration scenario, i.e., Eq. 5.13:

$$R^{(m+1)} = \sum_{k=1}^{K} \left(\pi_k c^{(k)^T} \right) R^{(m)} \tag{5.14}$$

Suppose at stage m, the rate matrix (from the input of m to the end) is $R^{(m)}$. In this setting, a set of rates is supportable if and only if $R^{(m)}$ can be written as a convex combination of the elements of $\{c | c \in \mathcal{C}_{m \to M}\}$. Thus, rate matrices that are in the convex hull of $\mathcal{C}_{m \to M}$ can be supported. Hence, stages $1, \ldots, m-1$ must be configured so that $R^{(m)}$ has the desired form for it to be supportable

by the following stages.

5.4 Routing and Scheduling of Flows

In many multistage architectures, there are multiple paths between I-O pairs. In this section, we employ the infinitely divisible fluid model and discuss the impacts of routing flows over multiple paths. In particular, we specify supportable rate regions associated with different routing algorithms, with a focus on *full division of flows*. Indeed, we show that the two stage cyclic shift can support all admissible flow traffic, and even multicast can be made possible over the same architecture with full division.

5.4.1 The Two Stage Cyclic Shift Architecture and Supportable Rates

Let us consider the two stage cyclic shift system given in Example 5.2. Suppose the rates of the flows at the input constitute the entries of the matrix,

$$R^{(1)} = [\vec{r}_1 \ \vec{r}_2 \ \cdots \ \vec{r}_N]^T$$

and hence the entries of $\vec{r}_i$ represent the rates of the flows at input i.

1. Let the first fabric in the two stage cyclic shift architecture be configured to the first shift of the identity matrix, i.e.,

$$c_1 = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Then,

$$\begin{aligned} R^{(2)} &= c_1^T R^{(1)} \\ &= [\vec{r}_N \ \vec{r}_1 \ \vec{r}_2 \ \cdots \ \vec{r}_N - 1]^T \end{aligned}$$

Hence, all the flows are shifted by one link with the given configuration.

2. Let the first fabric in the two stage cyclic shift architecture go through all the possible (N) cyclic configurations, and let the fraction of time spent in each one be /identical: $\pi_1 = \cdots = \pi_N = \frac{1}{N}$. With this sequence of configurations, each flow is divided into $K = N$ subflows and the rate of each subflow is identical and equal to $\frac{1}{N}$ of that of the original flow. In a multistage switch, if each flow is divided identically over all possible end to end paths between its I-O pair, we say *full flow division* is implemented by the switch. A two stage cyclic architecture is sufficient, but not necessary for full flow division. We talk about other alternatives for full division later in this chapter. Applying (5.14),

$$\begin{aligned} R^{(2)} &= \frac{1}{N}\vec{e}\vec{e}^{T}R^{(1)} \\ &= \frac{1}{N}\vec{e}\vec{e}^{T} \end{aligned} \tag{5.15}$$

where (5.15) follows since $\vec{e}^{T}$ is a left eigenvector of every stochastic matrix. This implies that with full division of flows at a certain stage, the loading in the following stage is uniform.

One can observe that, $R^{(2)} = \frac{1}{N}\vec{e}\vec{e}^{T}$ is in the convex hull of $\mathcal{C}_{2\to 2}$. Indeed, if $\mathcal{N}_2$ spends equal time in all of its configurations just like $\mathcal{N}_1$, it can support $R^{(2)}$ regardless of the end to end rate matrix, $R^{(1)}$.

We just showed that the network given in Fig. 5-4 can support all admissible traffic given that each flow is divided equally over the inputs of the second stage. In implementing this idea in a packet switching system, queueing at the input of both interconnections is necessary. We will talk about issues on transforming the problem to the domain of discrete cells from the domain of flows, (e.g., queue sizing and the packet delay) associated with this scheme later in this chapter. We analyze the service lag and end to end cell delay in the two stage cyclic shift system for the case where cells between any given I-O pair is routed over all the possible paths between the pair for all admissible rates.

5.4.2 Multicast Support

Recall that we gave a detailed treatment of multicast support in Chapter 4. We showed that a speedup of $\sim \log N$ is necessary to support all admissible multicast rates over a single crossbar. In this section, we review the model, and make some modifications for fluid flows. Then, we show that the two stage cyclic shift architecture can support all admissible multicast flows.

In the previous section, we showed that with full flow division at one stage, load can be uniformized for the following stage. With uniform load, this stage can follow a simple schedule and support point to point traffic. In this section, we will show that the same approach also works in the presence of *multicast flows*. Indeed, it turns out that all admissible multicast flows are supportable by our two stage switch without speedup. It is a surprising result, since as will be shown, a single stage crossbar switch cannot support all admissible multicast flows without speedup, even when one input can be connected to multiple outputs simultaneously.

First, let us introduce some notation. A point to point traffic flow has a single destination, and there is only one flow between each I-O pair. In multicast, we refine this and assume that more than one flow may share the same I-O pair. Each flow does not necessarily have a single destination and two such flows need to be treated separately even if they may have some common destinations. For two flows to be counted as one, they should share the same input and the same set of destinations.

Let there be k_{ij} flows which arrive at input i and have output j as one of their destinations. Let the first one of these flows be unicast[6] and all the other $k_{ij}-1$ be multicast flows. Also let the number of destinations and the rate for the lth such flow be $n_{ij}(l)$ and $R_{ij}^{(1)}(l)$ respectively. The rate matrix, $R^{(1)}$, can be broken into multiple pieces, one for the unicast traffic and a number of others for the multicast flows. For instance, consider the following 3×3 system:

$$R^{(1)} = \underbrace{\begin{bmatrix} 0 & 0.2 & 0.5 \\ 0.6 & 0.4 & 0 \\ 0.1 & 0.1 & 0.4 \end{bmatrix}}_{1} + \underbrace{\begin{bmatrix} 0.2 & 0.2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{2} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}}_{3}$$

where the first matrix represents the unicast flows, and matrix 2 and 3 represent the rates for the two multicast flows. The multicast flow with rate given in the second matrix arrives at the first input link and it must be duplicated at some point inside the multistage switch and sent to both of the destinations, 1 and 2. The last multicast flow has a rate of 0.1 and it is destined to all the output links. Hence, $k_{12} = 2$ and $n_{12}(2) = 2$ for example. Constructing the rate matrix, $R^{(1)}$, in the presence of multicast flows, multiple entries of the matrix are increased by an amount identical to the rate of the multicast flow. For instance if a flow arrives at input 1 and has destinations 2

[6] There can be only one unicast flow with the same input and output pair.

and 3, both $R_{12}^{(1)}$ and $R_{13}^{(1)}$ are increased by the rate of the flow. Hence, it may be the case that, for some i,

$$\sum_{j=1}^{N} \sum_{l=1}^{k_{ij}} R_{ij}^{(1)}(l) > 1 \tag{5.16}$$

That is, the rate matrix is no longer doubly stochastic. If (5.16) holds for some i, then multicast flows arriving at input link i cannot be duplicated at the input. Otherwise, the ith input link would be oversubscribed. But the actual rate of flows at the ith input link is not the sum of the entries over the ith row. In the presence of admission control, the following holds:

$$\sum_{j=1}^{N} \sum_{l=1}^{k_{ij}} \frac{R_{ij}^{(1)}(l)}{n_{ij}(l)} \leqslant 1 \tag{5.17}$$

for all i. The summation on the left side of Ineq. (5.17) is, indeed the actual rate of flows at input link i. The rate of each flow is divided by the number of destinations (fanout) of that flow since each flow is represented that many times in the same row of the rate matrix. The admission control inequality for the jth output link is as follows,

$$\sum_{i=1}^{N} \sum_{l=1}^{k_{ij}} R_{ij}^{(1)}(l) \leqslant 1 \tag{5.18}$$

This time, we did not divide the rate of each flow by the fanout of the flow since each flow is already duplicated before arriving at the output links and each copy of a flow can be treated as a separate unicast flow.

Next, suppose we have a multicast rate matrix, $R^{(1)}$ to be supported over the two stage architecture given in Fig. 5-4. Flows have to be duplicated at some point in the switch since the interconnections are not capable of sending a cell to multiple output links simultaneously. In fact, this duplication process is the only difference between support of multicast flows and unicast flows. The operation of the two stages can be described as follows.

Stage 1: In this stage each flow is divided into N subflows. No duplication takes place. The function of the first stage is exactly the same as it was in the unicast case. Hence, all flows are treated the same. The service that the interconnection provides is uniform ($\frac{1}{N}\vec{e}\vec{e}^{T}$) as in the case where all the flows are unicast.

Stage 2: Each flow is classified according to its destination set. It is then duplicated as many times as the fanout and one copy is sent to each of the destinations.

Claim 5.1 *After all duplication, no input link at the second stage is oversubscribed. Moreover, loading at these input links is uniform, i.e.,*

$$R^{(2)} = \frac{1}{N}\vec{e}\vec{e}^T$$

Note that every flow, after duplication, is destined to a unique output link, and we can talk about a unicast rate matrix, $R^{(2)}$, since all the traffic is indeed unicast at this point.

Proof: Since every flow is divided equally over the output links of the first stage, after duplication,

$$R_{ij}^{(2)} = \sum_{m=1}^{N}\sum_{l=1}^{k_{mj}} \frac{1}{N}\frac{R_{mj}^{(1)}(l)}{n_{mj}(l)} n_{mj}(l) \tag{5.19}$$

$$= \frac{1}{N} \tag{5.20}$$

where the factor $n_{ij}(l)$ in (5.19) is the number of copies of a flow after duplication and Eq. (5.20) follows since each column of $R^{(1)}$ sums to 1. Hence, we complete the proof by noting that the total rate of subflows at the ith input of the second switch is

$$\sum_{i=1}^{N} R_{ij}^{(2)} = 1$$

and no input link of the second switch is oversubscribed.

We just showed that with full division in the first stage, the loading of the second stage is uniform after duplication. Thus, the architecture given in Fig. 5-4 supports multicast flows as well. Recall that, as shown in Chapter 4, all admissible multicast flows cannot be supported by a single crossbar switch. Next, we study switch architectures other than the two stage cyclic shift architecture. We present the implementation issues and the performance analysis in the presence of multicast flows later in this chapter.

5.4.3 Benes Architecture

An $N \times N$ switch can be constructed recursively, using $b \times b$ crossbars, where $\log_b N$ is an integer as illustrated in Fig. 5-9. We call the $b \times b$ crossbars at the input the input stage crossbars, and

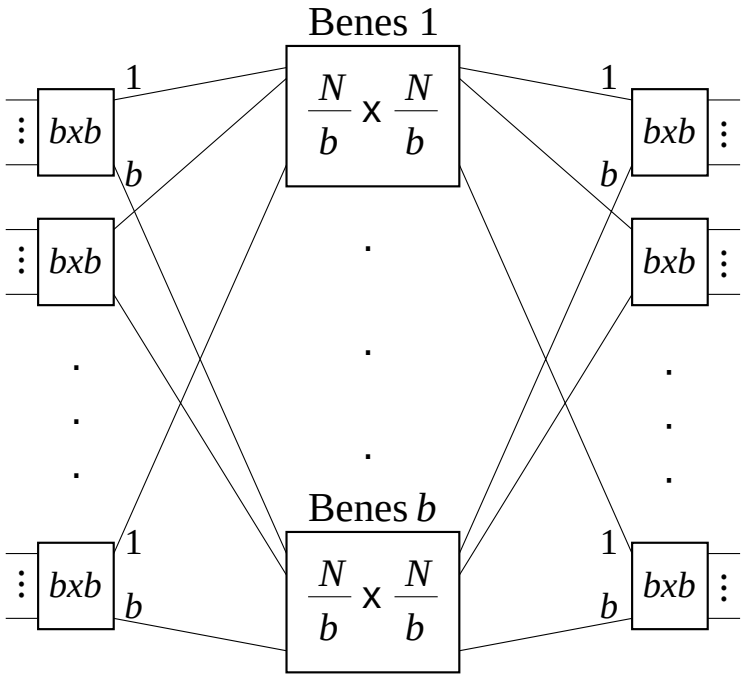

Figure 5-9: A Benes network can be constructed iteratively using smaller crossbars. There are $\frac{N}{b}$ input and output stage crossbars of size $b \times b$. There are b $\frac{N}{b} \times \frac{N}{b}$ Benes networks in the middle.

those at the output, the output stage crossbars. Together, input stage crossbars constitute an interconnection, and so do the output stage crossbars. Each $\frac{N}{b} \times \frac{N}{b}$ unit at the center is also a Benes network. Each such network has $b \times b$ input stage crossbars, $b \times b$ output stage crossbars and $\frac{N}{b^2} \times \frac{N}{b^2}$ Benes networks at the center. Notice that, ultimately the described network is composed of $b \times b$ crossbars only.

Suppose there are flows with rates, $R^{(1)}$. First assume they are all unicast flows, and thus, $R^{(1)}$ is doubly stochastic.

Claim 5.2 *Any doubly stochastic flow matrix is supportable over an $N \times N$ Benes switch which is composed of $b \times b$ crossbars, where $\log_b N$ is an integer.*

Proof: By induction. For $N = b$ ($\log_b N = 1$), the network is composed of a single crossbar, and hence any $b \times b$ doubly stochastic matrix is supportable. Suppose the claim holds for $N = b'$ ($\log_b N = \log_b b'$). We will show that it also holds for $N = bb'$ ($\log_b N = \log_b b' + 1$). A $bb' \times bb'$ Benes network can be recursively constructed using b $b' \times b'$ Benes networks as illustrated in Fig. 5-9. Let the input stage $b \times b$ crossbars divide each flow into b identical (rate) subflows and send each subflow to a different output link. The set of $b' \times b'$ middle stage Benes networks each has the same traffic matrix, although these matrices are not necessarily uniform. That is, it may be

the case that $R^{(2)} \neq \frac{1}{N}\vec{e}\vec{e}^T$. The total rate, $q_i^{(\text{in})}$, of traffic at the ith input of each $b' \times b'$ Benes network is

$$\begin{aligned} q_i^{(\text{in})} &= \sum_{l=1}^{N} \sum_{m=(i-1)b+1}^{ib} \frac{1}{b} R_{ml} \\ &= \sum_{m=(i-1)b+1}^{ib} \frac{1}{b} \\ &= 1 \end{aligned} \tag{5.21}$$

Similarly, the rate, $q_j^{(\text{out})}$, of traffic which is destined to the jth output stage crossbar is,

$$\begin{aligned} q_j^{(\text{out})} &= \sum_{m=1}^{N} \sum_{m=(j-1)b+1}^{jb} \frac{1}{b} R_{ml} \\ &= \sum_{m=(j-1)b+1}^{jb} \frac{1}{b} \\ &= 1 \end{aligned} \tag{5.22}$$

Equations (5.21) and (5.22) show that with full division at the input stage, the input links and the output links of the middle $b' \times b'$ Benes networks are not oversubscribed. Indeed, the $b' \times b'$ rate matrix for these networks are doubly stochastic and hence the proof is complete.

Examining the algorithm described in the proof carefully, one can realize that it is sufficient that the $b \times b$ input and the output stage crossbars go through b equally weighted configurations each of which connects each input to a distinct output. The identity configuration and its $b-1$ cyclic shifts is an example for such sets of b configurations.

Combining all these facts together, we see that all the $b \times b$ crossbars in the network except the ones located in the center (stage $\log_b N$) follow the N-connection full division switch schedule regardless of the end to end rate matrix (see Fig. 5-10 for an illustration with $b = 2,\ N = 8$). The crossbars at the center stage have to alter the configurations according to the changes in the rate matrix. They all go through the same configurations since they are identically loaded. We will analyze the performance of this switch under unicast packet traffic later.

Suppose, there are multicast flows as well as unicasts. Similar to the procedure described in Section 5.4.2, let us postpone duplication until the center stage and distribute multicast flows just like the unicasts prior to the center stage. Each one of the N/b middle stage $b \times b$ crossbars will

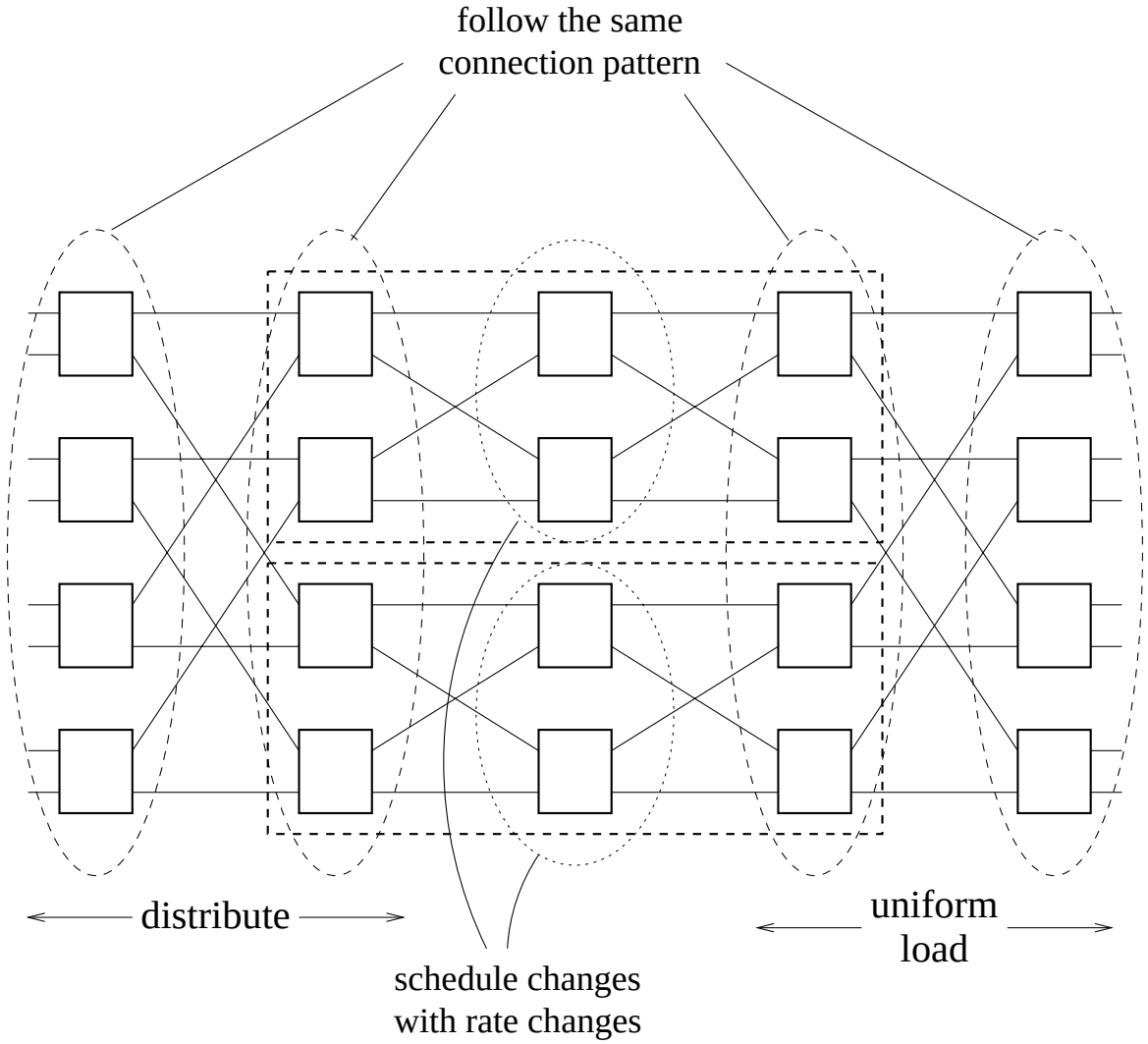

Figure 5-10: An 8×8 Benes network. Only the middle stage 2×2 units have non-trivial scheduling.

be identically loaded. At this point, those flows with multiple destinations must be duplicated. If we postponed the duplication further, the system would not be able to route some of the flows as desired since there is a single path between each output link and each center stage switch. Indeed, when $l \leqslant \log_b N$, there are Nb^{-l} paths between each input link of the lth stage interconnection and each output link of the system. For $l > \log_b N$, each $b \times b$ crossbar is connected to only a fraction $Nb^{-l} < 1$ of the output stage (stage $2\log_b N - 1$) crossbars, i.e., there are no paths to the other output links. Thus, the center stage crossbars must be able to provide multicast support for the Benes network to support all multicast rate matrices. It was shown earlier in this thesis that, for a crossbar to support all admissible multicast rate matrices, some speedup is necessary. Hence, $b \times b$ crossbars without speedup in the middle is not sufficient for the Benes network to support all multicast rate matrices. We will consider two alternatives for multicast support with or without speedup.

The first alternative is to put output queued $b \times b$ crossbars with a speedup of b in the center stage. In this setting, an input can be connected to <u>all</u> the outputs of the $b \times b$ crossbar simultaneously, regardless of the connections between other I-O pairs. Therefore, it can support all admissible multicast and unicast flows. There are certain practical issues with this architecture. While, a speedup proportional to the size of a crossbar may not be feasible for small b (e.g., 2) this speedup may be practical. A key advantage of this architecture is simple switch scheduling: Recall that each crossbar follows the uniform division schedule, i.e., $\frac{1}{t}D(t) = \frac{1}{b}\vec{e}\vec{e}^T$ for all $t > 0$. In the center interconnection, the crossbars have each input connected to all of its outputs at all times, independent of the set of rates. This middle stage has another advantage in that, in the corresponding packet switch, the delay experienced at the center stage is 0.

The second alternative is using the two stage cyclic shift systems of size $b \times b$. With this selection, the Benes network will become two Banyan networks in cascade. We study this architecture in the following section extensively.

5.4.4 Cascaded Banyan Networks

We showed in Section 5.4.2 that the two stage cyclic shift architecture supports all admissible multicast rate matrices. We can replace each stage $\log_b N$ crossbar with the two interconnection architecture given in Fig. 5-4 and the Benes network will be able to support all multicast rate matrices without speedup. With this expansion of number of stages by one, the system is no longer

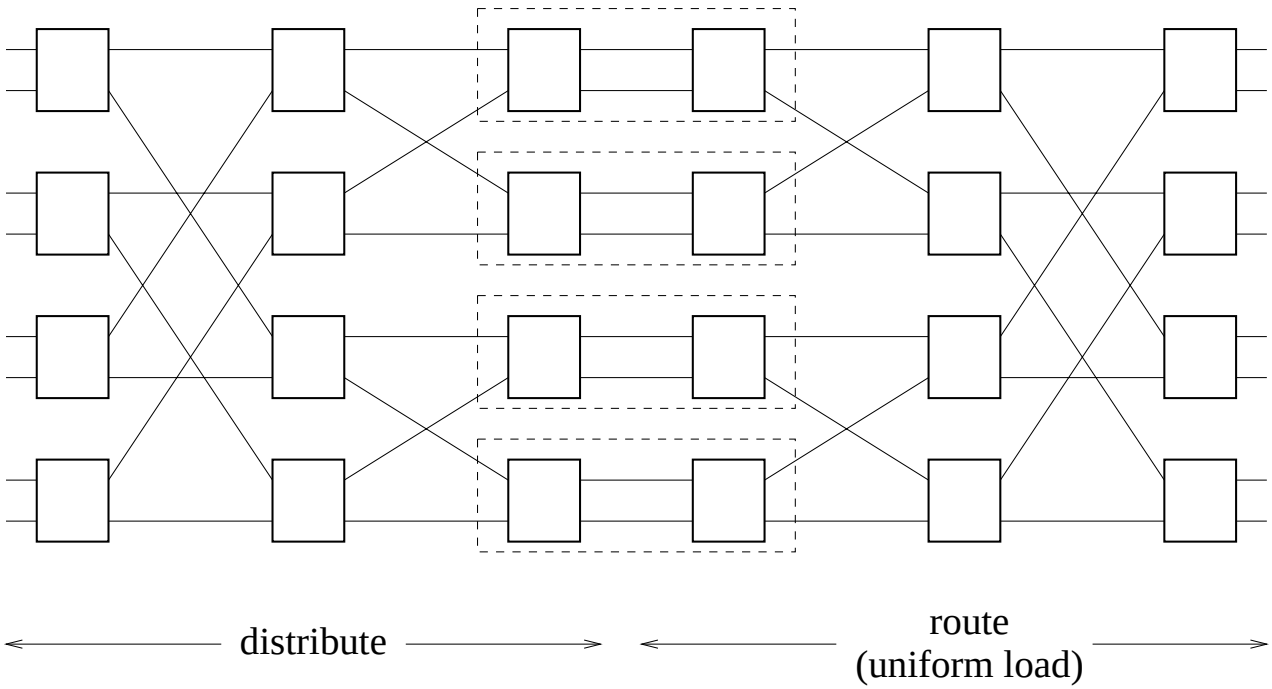

Figure 5-11: As marked in dashed boxes, each crossbar in the center is replaced with the two stage cyclic shift system. Overall, the system is composed of two Banyan networks put in cascade. The second one is an inverted Banyan which is isomorphic to a regular Banyan.

a Benes network but two Banyans put in cascade as illustrated in Fig. 5-11. Note that the ordering of the stages is inverted in the second Banyan; this is isomorphic to a regular Banyan network.

There are two key advantages of this system. First, no speedup is necessary for multicast support. Given that the duplication process is handled at the input of the second Banyan network (stage $\log_b N - 1$), the load at every $b \times b$ switch is uniform. Also, the first Banyan network is responsible for division of flows only. This leads to the second advantage of the system, that the switch scheduling is trivial. Each $b \times b$ crossbar will be configured to the full division schedule, i.e., $\frac{1}{t}D(t) = \frac{1}{b}\vec{e}\vec{e}^T$ regardless of the set of rates and this will be sufficient to accommodate all admissible unicast and multicast traffic.

On the other hand, compared to the system with speedup, the maximum end to end delay for this system is worse in the corresponding packet switch. We discuss the packet performance of this architecture later in this chapter.

Discussion: A cyclic shift fabric connects inputs to outputs in such a way that each input spends equal amount of time connected to each output. As mentioned before, a cyclic shift fabric is not necessary to achieve the *full division* property. Next, we show that a Banyan network can provide full division, and illustrate a schedule of configurations by which this can be achieved. This is also

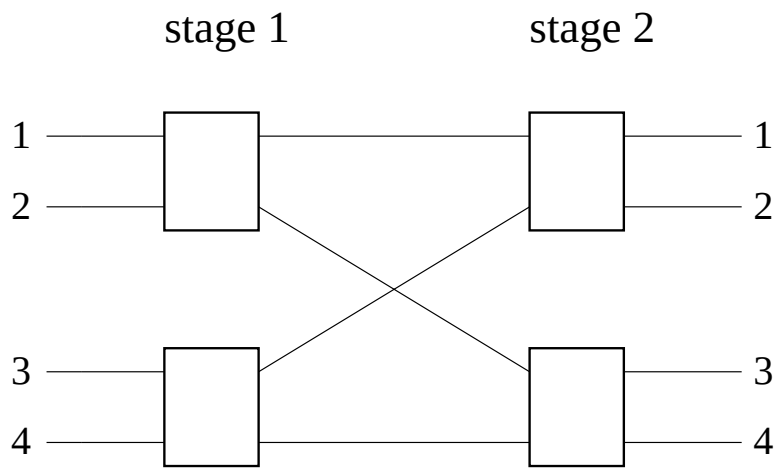

Figure 5-12: The 4×4 Banyan network

the place where we start discussing packet switching. We assume that a switch configuration is kept for a time slot.

Example 5.4 *Consider the 4×4 Banyan network given in Fig. 5-12. It can be seen that this network cannot support the identity configuration; e.g., if input link 1 is connected to output link 1, input link 2 cannot be connected to output link 2. In fact, if any two input links share the same local switch in the first stage, they cannot in the second stage.*

However, being able to configure the identity matrix and its cyclic shifts is not necessary for full division property. Any set of N configurations is sufficient if each input is connected to each output exactly once in these configurations. Let us represent each $N \times N$ permutation matrix with the corresponding permutation of $[12 \cdots N]$. Then the $N \times N$ matrix whose columns (or rows) are (permutations corresponding to) the N configurations must form a Latin square. For instance, for the 4×4 Banyan network, consider the following configurations:

$$\begin{bmatrix} 1 \\ 3 \\ 2 \\ 4 \end{bmatrix}, \begin{bmatrix} 3 \\ 1 \\ 4 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \\ 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

where in the first configuration, inputs 1, 2, 3, 4 are connected to outputs 1, 3, 2, 4 respectively.

We can see that,

$$\begin{bmatrix} 1 & 3 & 2 & 4 \\ 3 & 1 & 4 & 2 \\ 2 & 4 & 1 & 3 \\ 4 & 2 & 3 & 1 \end{bmatrix}$$

is a Latin square. Each permutation given above is configurable by the Banyan network, and the switch can alternate among these configurations for the desired operation.

In general, consider $b \times b$ fabrics that are connected to form of an $N \times N$ Banyan network as shown in Fig. 5-3 for $N = 4$ and $b = 2$. These $b \times b$ fabrics can also be Banyan networks by themselves, or any fabric with a set of b configurations which form a Latin square would suffice (e.g., the cyclic shift fabric). The N configurations by which every input is connected to every output exactly once can be provided as follows. Suppose the $N \times N$ Banyan network is configured to c_1 initially. In the next b time slots, the final stage (i.e., stage $\log_b N$) fabrics go through a set of b configurations that form a Latin square. After one such cycle, stage $\log_b N - 1$ fabrics alter their configurations and final stage fabrics repeat their cycle once again. This goes on until each stage $\log_b N - 1$ fabric completes one cycle going through b configurations which form a Latin square. Then, fabrics at stage $\log_b N - 2$ change their configurations and the cycle for the final two stages repeat again. This whole process continues until the first stage fabrics go through a complete cycle. The procedure can be summarized as follows:

Algorithm: Stage m fabrics change their configuration every $b^{\log_b N - m}$ time slots. The configurations that each fabric goes through forms a Latin square. Upon completion of a cycle, it is repeated. One complete cycle lasts for $b^{\log_b N} = N$ time slots as expected.

The set of configurations that the algorithm goes through has the desired property, namely, no input is connected to an output more than once. Indeed, every input is connected to every output exactly once. This can be proved as follows. As shown earlier, there is only one path between each I-O pair in a Banyan network. Within one cycle of period N, there exists no two time slots in in which the states of all $b \times b$ fabrics are identical along any end to end path. At least one $b \times b$ fabric alters its configuration along each path. If it were true that an input output pair of the $N \times N$ Banyan network is connected in two different time slots within a period of N time slots, it would mean that there actually are two paths between the pair, a contradiction. Thus, every I-O pair is

connected exactly once in a period of N time slots.

We just showed how to implement a full division schedule using a Banyan network. Thus, a network composed of two cascaded Banyan networks can be used to support all admissible unicast and multicast rate matrices instead of the two stage cyclic shift architecture. Buffering is necessary only at the input and output of each Banyan network. No buffering is needed inside of each Banyan network.

5.5 Routing Cells on Multiple Paths - Full Division

In this section, we will use the ideas and insights we have for the flow switching scenario and show how to apply them in multistage packet switches. We will present different ways of routing cells which share the same I-O pair over multiple paths. In particular, we will focus on *full division*, in which all possible end to end paths are used equally between any I-O pair to transfer cells from the input to the output. The architectures that will be analyzed are the cascaded cyclic shift (Fig. 5-4), and the cascaded Banyans (Fig. 5-11). We assume the presence of VOQs at the inputs of interconnections.

For a given contract, (R,T) we will study how different schedulers perform in each architecture. Our performance metric is the service lag and the corresponding cell delay. Indeed, for all admissible sets of rates, we will evaluate an upper bound for the maximum service lag and the corresponding delay. As shown earlier, in the fluid limit, the service lag for all supportable rates is 0 for all input-output pairs at any point in time since $\frac{1}{t}D(t)$ can be any convex combination of configuration matrices for all $t > 0$. In that sense, the service lag can be viewed as a metric for the difference between the service provided by a certain scheduler and that of the corresponding flow scheduler with the same set of rates.

Since different cells that belong to the same I-O pair follow different paths, they may get out of order. We assume that there are queues at the output of the switches to reorder cells and reassemble packets. We will evaluate the necessary buffer sizes both at the input of the interconnections and at the output for each architecture and the corresponding scheduler.

5.5.1 Cascaded Cyclic Shift Interconnections

First we consider the two stage cyclic shift architecture. We had proved that in the fluid limit, if each flow is broken into N identical subflows, each of which is routed on a different path, all

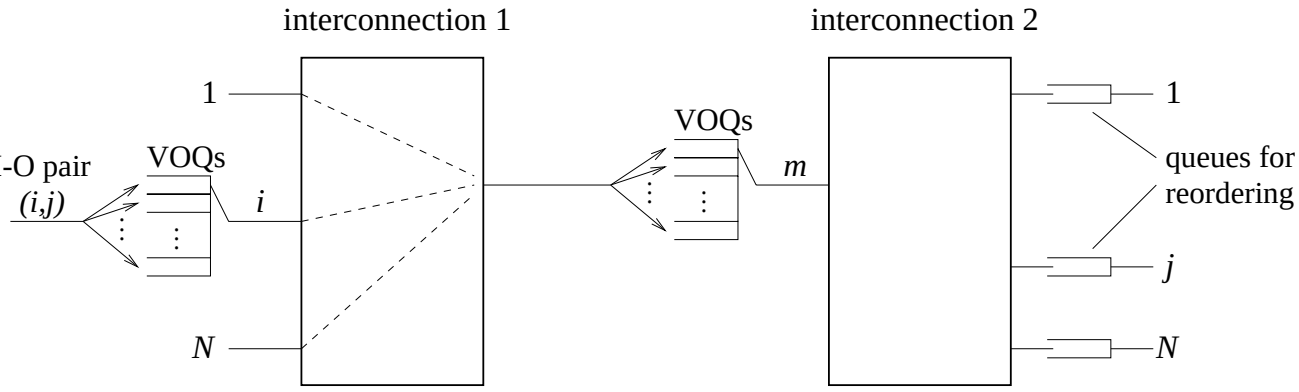

Figure 5-13: Cells with common I-O pairs are placed in multiple VOQs in the first stage so that they follow N different paths to get to their destinations. Connections shown with dashed lines are configured once every N time slots.

admissible flow traffic can be supported. In a packet switch, each packet is sliced into equal sized cells, and these cells cannot be further broken into multiple subcells since cells are not (infinitely) divisible like flows.

There are N VOQs at each input of an interconnection, and the lth VOQ at each link carries cells destined to the lth output link of the interconnection. To utilize every path between I-O pairs, we place consecutive cells of each I-O pair to consecutive VOQs at the input of the first interconnection. Namely, if a cell is placed in the lth VOQ, then the next cell with the same I-O pair is placed in VOQ $(l+1) \bmod N$. At the input of the second interconnection, each cell is placed in the VOQ through which it can be routed to its ultimate destination. Cells are reordered and packets are reassembled at queues at the output of the system. The whole process is summarized in Fig. 5-13. Regardless of how cells are scheduled at each VOQ, some service lag is experienced due to the discrete nature of cell switching. For each scheduler, we will find upper bounds on the maximum service lag and the corresponding cell delay between all the I-O pairs for all admissible rates. This will also justify that all admissible rates are supportable. We will treat multicast separately, and unless mentioned otherwise, we deal with unicast rates and thus, doubly stochastic matrices.

As a warm-up exercise for the performance analyses, consider an infinite duration contract with the rate matrix, $R^{(1)}$. Let $R_{ij}(m)$ be the rate of (i,j) traffic going through the mth intermediate link. Since (i,j) traffic is divided equally over all intermediate links,

$$R_{ij}(m) = \frac{1}{N} R_{ij}^{(1)}$$

where the factor, $\frac{1}{N}$ is due to the fact that the contract is of infinite duration and the ratio of the number of cells served through each VOQ to the total number of cells approaches $\frac{1}{N}$. Thus, the total rate of the traffic at intermediate link m coming from input link i is

$$\begin{aligned}\sum_{j=1}^{N} R_{ij}(m) &= \sum_{j=1}^{N} \frac{1}{N} R_{ij}^{(1)} \\ &= \frac{1}{N}\end{aligned}$$

Similarly, the total rate of output j traffic at the mth intermediate link is

$$\begin{aligned}R_{mj}^{(2)} &= \sum_{i=1}^{N} R_{ij}(m) \\ &= \sum_{i=1}^{N} \frac{1}{N} R_{ij}^{(1)} \\ &= \frac{1}{N}\end{aligned}$$

Thus, at the input and output of both the first and the second interconnection, there are cells with N different I-O pairs of total rate N^{-1} competing for each output link.

Both interconnections have a uniform schedule of configurations. Within one schedule period of N time slots, each input gets connected to each output for one time slot. This process continues regardless of the set of rates desired between I-O pairs. We will evaluate the service lag in what follows for different cell schedulers. We will consider two such schedulers, worst case fair weighted fair queueing (WF^2Q) and the simple first in first out (FIFO).

WF^2Q Scheme

The first alternative we consider is worst case fair weighted fair queueing (WF^2Q) which is proposed in [23]. If in a packetized processor sharing system, a constant rate resource is shared by a number of users, each of which ask for a certain fraction of that resource, WF^2Q scheduler generates a schedule for the order that these users get served. Consider the system given in Fig. 5-14. It was shown in [23] that the number of service opportunities, $D_m(t)$, given to the mth user by some given time, t, satisfies the following:

$$\left|\left[D_m(t') - D_m(t)\right] - (t'-t)\phi_m\right| < 1 \tag{5.23}$$

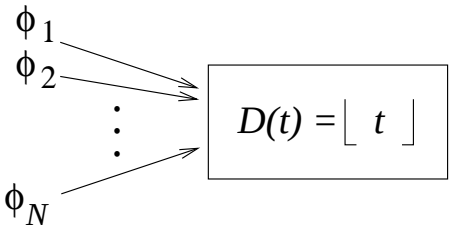

Figure 5-14: A constant bit rate service is shared by N users. One cell is served every time slot.

where ϕ_m is the weight of the mth user. We call this inequality, the WF^{2}Q service inequality. This means, within any two points in time, the number of service opportunities provided to a user is within 1 of what it asks for.

The entire system with WF^{2}Q is illustrated in Fig. 5-15. In the first stage, cells with the same I-O pair are divided among the N VOQs at that link as described before. Each VOQ can be viewed as a combination of N subqueues. Cells with the same ultimate destination share the same subqueue. The WF^{2}Q schedule is generated for each VOQ where the weights assigned to each subqueue are identical to the rates specified in the rate matrix[7]. For instance, the jth subqueue of the mth VOQ at input link i carries the (i, j) cells which are routed through the mth intermediate link. The weight assigned to this subqueue is $R_{ij}^{(1)}$. Service opportunities are awarded to each subqueue whenever the appropriate switch configuration is set up, i.e., once every N time slots. The division process is illustrated in Fig. 5-16 and the mth VOQ at input i along with its scheduler is illustrated in Fig. 5-17. To repeat, we visualize each VOQ to be formed of N separate subqueues, the jth of which carries the cells with ultimate destination j. We will use this model in the rest of this section.

Let us define $D[t, t']$ to be the service opportunities provided in the period $[t, t']$. Applying the WF^{2}Q service inequality (5.23) to the jth subqueue of the mth scheduler at the ith input link in the the first stage, we get:

$$D_{ij,m}^{(1)}[t, t'] > (t' - t)\frac{R_{ij}}{N} - 1 \tag{5.24}$$

Suppose a cell arrives at the head of this subqueue at time t and it gets scheduled to leave at time t'. The following relation can be written between the delay, $(t' - t)$, experienced at the head of the

[7]One can realize that the schedule is identical for each VOQ.

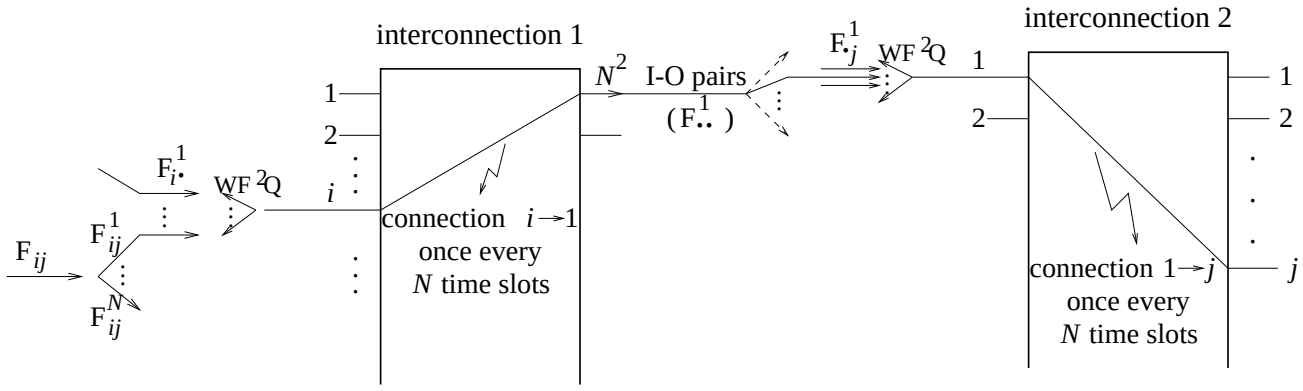

Figure 5-15: Cells of each I-O pair are placed in N VOQs and those sharing the same queue are scheduled with a WF^{2}Q scheduler. In the second stage, subflows are grouped according to their ultimate destinations and those sharing the same destination are again scheduled by a WF^{2}Q scheduler. A total of N schedulers per link is necessary.

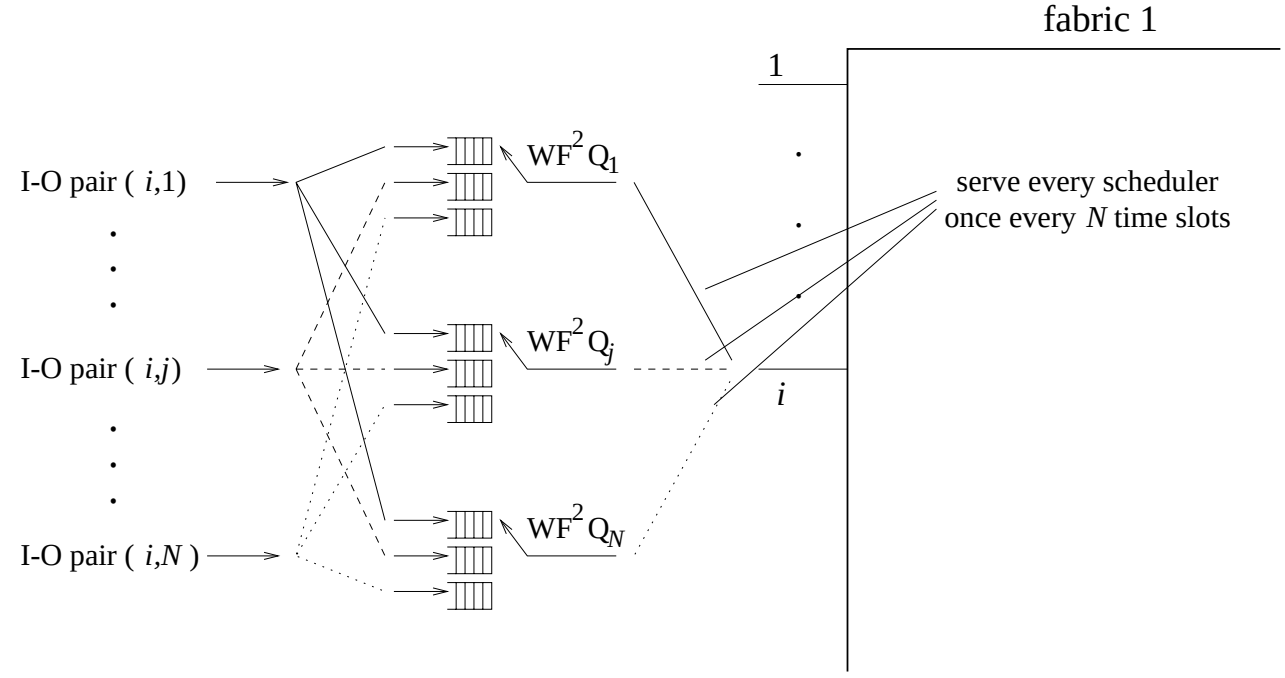

Figure 5-16: For the division process, N WF^{2}Q schedulers are used per link. If a cell with I-O pair (i, j) is placed into the queue to be scheduled by the mth WF^{2}Q scheduler, the next incoming cell with the same I-O pair will be served by scheduler $(m + 1)$ mod N.

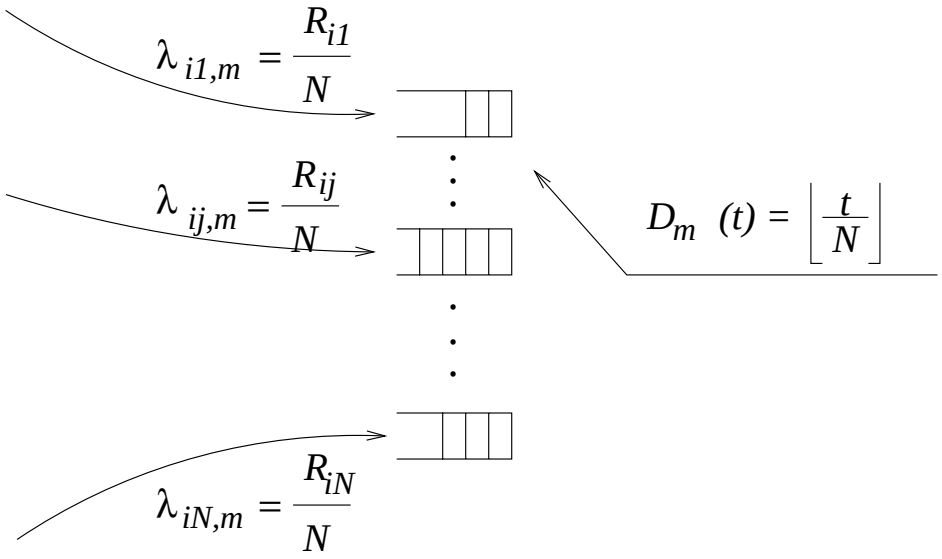

Figure 5-17: The mth VOQ at the ith input link and its scheduler are illustrated. Each VOQ is composed of N subqueues. The scheduler uses the entries of the ith row of the matrix as the weights; $\lambda_{ij,m}$ is the weight used by the scheduler to construct the schedule for providing service opportunities to the cells with I-O pair (i, j).

queue by this cell:

$$\begin{aligned} 1 &= D^{(1)}_{ij,m}[t, t'] \\ &> (t' - t)\frac{R_{ij}}{N} - 1 \end{aligned}$$

hence,

$$t' - t < \frac{2N}{R_{ij}} \tag{5.25}$$

In fact, the delay experienced by a cell at the head of its subqueue is the relevant parameter for measuring the delay of a cell at the input stage. This can be clarified as follows. If we sum (5.24) over all VOQs for I-O pair (i, j), we get:

$$\begin{aligned} \sum_{m=1}^{N} D^{(1)}_{ij,m}[t, t'] &> \sum_{m=1}^{N} (t' - t)\frac{R_{ij}}{N} - 1 \\ D^{(1)}_{ij}[t, t'] &> (t' - t)R_{ij} - N \end{aligned} \tag{5.26}$$

Hence for $t = 0$,

$$D_{ij}^{(1)}(t') > t' R_{ij} - N \tag{5.27}$$

Thus, at any point in time, the number of service opportunities provided for the cells of an I-O pair cannot be more than $N - 1$ behind that desired by that pair in that period. Suppose a cell with I-O pair (i, j) arrives at its subqueue, m, at time t and another cell arrives at the same queue at some $t' > t$ when the first one is still in the queue. Then, the following relation can be written between the number of arrivals and service opportunities provided for the cells with I-O pair (i, j) at the mth VOQ in period $[t, t']$.

$$A_{ij,m}^{(1)}[t, t'] - D_{ij,m}^{(1)}[t, t'] \geqslant 2 \tag{5.28}$$

Thus,

$$\begin{aligned} A_{ij,m}^{(1)}[t, t'] &\geqslant 2 + D_{ij,m}^{(1)}[t, t'] \\ &> 1 + (t' - t)\frac{R_{ij}}{N} \end{aligned} \tag{5.29}$$

where (5.29) follows from (5.24). Since the cells for an I-O pair are placed at VOQs in an alternating fashion (covering all queues with every N arrivals), the following can be written for any $1 \leqslant m, m' \leqslant N$ such that $m \neq m'$:

$$\left| A_{ij,m}^{(1)}[t, t'] - A_{ij,m'}^{(1)}[t, t'] \right| \leqslant 1 \tag{5.30}$$

Hence,

$$\begin{aligned} A_{ij}^{(1)}[t, t'] &= \sum_{l=1}^{N} A_{ij,l}^{(1)}[t, t'] \\ &= A_{ij,m}^{(1)}[t, t'] + \sum_{l \neq m} A_{ij,l}^{(1)}[t, t'] \\ &\geqslant A_{ij,m}^{(1)}[t, t'] + (N - 1)\left(A_{ij,m}^{(1)}[t, t'] - 1 \right) \qquad (5.31) \\ &> 1 + (t' - t) R_{ij} \qquad (5.32) \\ &\geqslant \lceil (t' - t) R_{ij} \rceil \end{aligned}$$

where $A_{ij}^{(1)}[t,t']$ is the number of cells that arrive in time period $[t,t']$ with I-O pair (i,j); Ineq. (5.31) follows from (5.30) and Ineq. (5.32) follows from (5.29).

This result can be interpreted as follows. If a cell that arrives at time t' observes a non-empty subqueue at the input of the first stage, then there exists some $t < t'$ such that the number of arrivals of cells with the same I-O pair in $[t,t']$ is greater than what the switch is contracted to support. In other words, within that period, more cells are inserted into the switch than it is supposed to serve, and the extra delay experienced in the subqueues other than at the head of it, is not due to the service lag of the service provisioned by the switch. Thus, in the first stage, the portion of the delay which is relevant for measuring the quality of service provided is that experienced at the head of the subqueues at each VOQ.

Next, we study the second stage where cells are routed to their ultimate destinations. For each output link, a WF^{2}Q scheduler decides which subqueue of each VOQ to provide a service opportunity with, when the corresponding configurations are set. The process is illustrated in Fig. 5-18. Each scheduler provides an opportunity once every N time slots to a subqueue following the schedule which is a function of the rates of the I-O pairs sharing the VOQ. The jth scheduler for the mth input link of the second interconnection is illustrated in Fig. 5-19. Each VOQ is composed of subqueues, one for the cells coming from each input.

Applying the WF^{2}Q service inequality (5.23) to the jth scheduler at intermediate link m, we get:

$$D_{ij,m}^{(2)}[t,t'] > (t'-t)\frac{R_{ij}}{N} - 1 \tag{5.33}$$

for all i and m. One can also see that the number of cell arrivals at the ith per flow queue of the jth scheduler at link m is upper bounded by the number of service opportunities given to the corresponding subqueue in the first stage:

$$\begin{aligned} A_{ij,m}^{(2)}[t,t'] &\leqslant D_{ij,m}^{(1)}[t,t'] \\ &< 1 + (t'-t)\frac{R_{ij}}{N} \end{aligned} \tag{5.34}$$

where Ineq. (5.34) follows from the WF^{2}Q service inequality. Combining (5.33) and (5.34), we get:

$$A_{ij,m}^{(2)}[t,t'] - D_{ij,m}^{(1)}[t,t'] < 2 \tag{5.35}$$

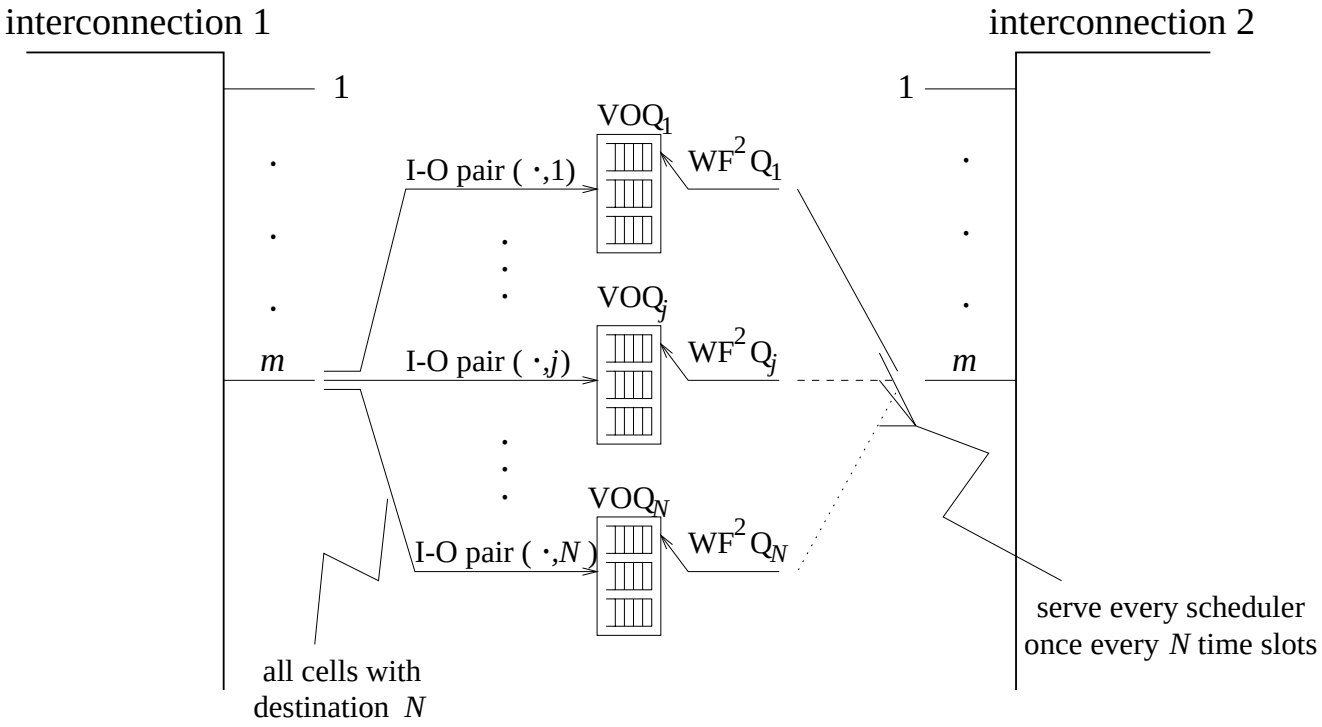

Figure 5-18: For the routing process, N WF^{2}Q schedulers are used per link. Cells are grouped according to their ultimate destinations and a cell destined for output j is scheduled by WF^{2}Q$_j$.

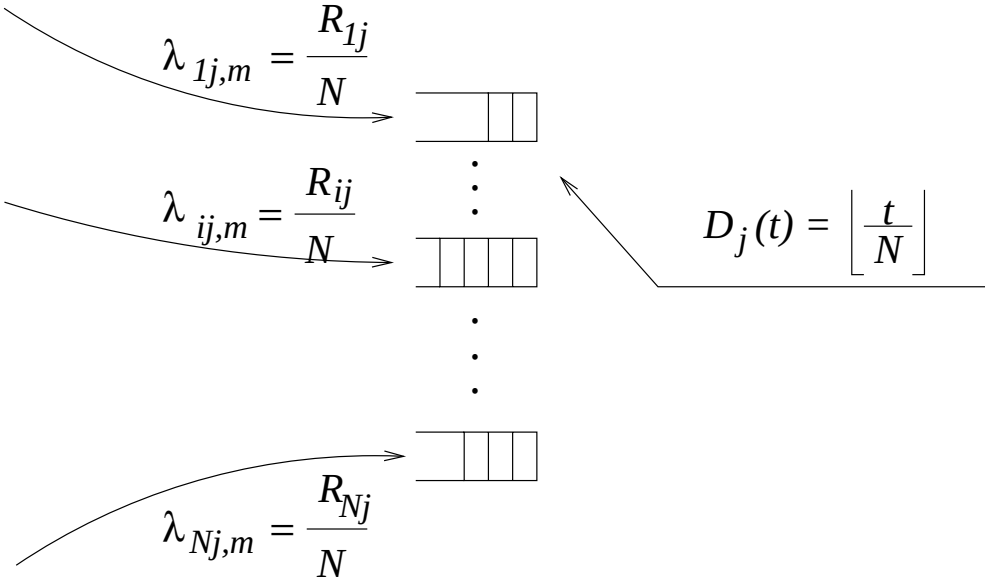

Figure 5-19: The jth scheduler at the mth input link of the second switch is illustrated. Schedulers are not necessarily symmetric; $\lambda_{ij,m}$ is the weight assigned to the jth subqueue where cells with I-O pair (i, j) arrive.

where the difference on the left side of Ineq. (5.35), gives us the accumulation, $Q^{(2)}_{ij,m}[t,t']$, of cells in the subqueue in the time period. Assuming all queues initially empty, the size of a subqueue at the input of the second interconnection never exceeds 1 cell. It may be the case, though, that a cell leaves its queue just before a new cell arrives there.

We can also calculate the worst case delay similar to the first stage. Suppose a cell arrives at its second stage subqueue at time t. We just showed that it is placed to the head of the queue upon arrival. Suppose it gets transferred to the output of the system at time t'. Then,

$$\begin{aligned} 1 &= D^{(2)}_{ij,m}[t,t'] \\ &> (t'-t)\frac{R_{ij}}{N} - 1 \end{aligned}$$

and hence,

$$t' - t < 2\frac{N}{R_{ij}} \tag{5.36}$$

is the bound on the maximum delay in the intermediate links.

Finally, we will talk about the delay and the maximum possible size of reordering queues[8] at the output of the multistage switch. We showed that the maximum delay that a cell with the I-O pair (i,j) experiences from the input to the output queues of the switch is $2\left(\frac{2N}{R_{ij}}\right) = \frac{4N}{R_{ij}}$ which is a function of the rate, R_{ij}.

Note that if a cell experiences maximum delay at the first stage and the input of the second stage, it does not need to be delayed at the output queues for reordering at all. Even if the cells that are ahead of our cell experience the maximum delay, our cell cannot get ahead of them. Therefore, the maximum end to end delay is the sum of the maximum delays experienced in the first stage and the input of the second:

$$\text{Maximum Total Delay} = \frac{4N}{R_{ij}}$$

Suppose a cell with the I-O pair (i,j) experiences maximum delay and a number of following cells do not experience as high a delay. They must be kept at the reordering queues and cannot be put onto the output links until our cell gets transferred there. The accumulation at the output

[8]The reordering queues must be push in push out (PIPO), which enables cells to be inserted and read from arbitrary locations of the buffer, not necessarily the two ends.

queues can be as high as

$$\begin{aligned} Q_{ij}^{(\text{out})}[t,t'] &\leqslant \frac{4N}{R_{ij}}R_{ij}+1 \\ &= 4N+1 \end{aligned} \tag{5.37}$$

where the extra 1 is for our cell itself. Note that there are, in fact, N subqueues per VOQ and a cell which shares the same subqueue (and hence the same route) as our cell cannot get to the output queues ahead of our cell if it arrived at the switch later. Thus, the above bound can be improved as follows:

$$\begin{aligned} Q_{ij}^{(\text{out})}[t,t'] &\leqslant 4N\frac{N-1}{N}+1 \\ &= 4N-3 \end{aligned} \tag{5.38}$$

To summarize, with WF^{2}Q scheduling in every VOQ, the total amount of queueing per link and the maximum delay experienced by a cell of an I-O pair with rate r are given in the following table.

	Input	*Intermediate*	*Output*	*Total*
Queue Size (per link)	N^2	N^2	$(4N-3)N$	$\sim 6N^2$
Maximum Delay	$2N/r$	$2N/r$	-	$4N/r$

Using the result about the maximum delay, the service lag can be found to be $4N$:

$$\begin{aligned} D_{ij}[t,t'] &\geqslant \left(t'-t-\frac{4N}{R_{ij}}\right)R_{ij} \\ &= (t'-t)R_{ij}-4N \end{aligned} \tag{5.39}$$

We finally talk about the delay and service lag in the presence of multicast. Recall that when we talked about flow switches, we assumed that there may be multiple flows with the same I-O pair. Similarly, in the packet switch, we assume that there may be more than one *class* of cells with the same I-O pair. If two cells arrive at the same input and have an identical set of outputs, they are defined to be of the same class. Namely, a class of traffic is the traffic defined by an input and either a single output or a set of multicast destinations. A subqueue is kept in each VOQ for each class present at the link. The structure of the rate matrix is as described earlier in the context of

flows.

Each class of cells follow N different paths, and therefore if a cell of a certain class is placed in a VOQ, the next cell of the same class is placed in the next VOQ. The delay bounds developed for the first stage are still valid in the multicast scenario. Since cells are not duplicated in the first stage, the division process is no different from the unicast case, i.e., (5.25) is still valid.

Recall that we defined k_{ij} to be the number of flows between input i and output j and $n_{ij}(l)$ to be the fanout of the lth such class. We define them similarly in the packet switch, i.e., k_{ij} is the number of classes with I-O pair (i,j) and $n_{ij}(l)$ is the fanout of the lth such flow. At the input of the second stage, each cell is duplicated and the number of copies is equal to the fanout of its class. Each copy is placed in a different subqueue. Thus, there are

$$\begin{aligned} K &= \sum_{i \leqslant N} \sum_{j \leqslant N} k_{ij} \\ &\geqslant N^2 \end{aligned}$$

subqueues at each input link of the second stage. The duplication process is illustrated in Fig. 5-20.

During the duplication process, each cell coming from mth subqueue, of the lth class with I-O pair (i,j) is replicated $n_{ij}(l)$ times. There will still be N WF^{2}Q schedulers, but this time each will handle more than N subqueues. In fact, the number of subqueues that a scheduler operates with differs for different outputs. After duplication, the number of subqueues per scheduler is

$$\begin{aligned} \frac{K}{N} &= \frac{1}{N} \sum_{i \leqslant N} \sum_{j \leqslant N} k_{ij} \\ &\geqslant N \end{aligned}$$

The WF^{2}Q service inequality (5.23) is independent of the number of flows that share the pipe. Therefore, the bounds derived for the cell delay at the input of the second stage hold in the multicast scenario as well. However, bounds on the buffer size necessary for this process scale with the sum of the fanouts. For instance, if there is one multicast class per input output pair (along with unicast) with an average fanout of 2, the queueing requirement will scale with a factor of 3. Besides, each scheduler will have to deal with $3N$ subqueues, compared to N with all unicast.

Similarly, bounds on the size of the reordering queues at the output need to be expanded

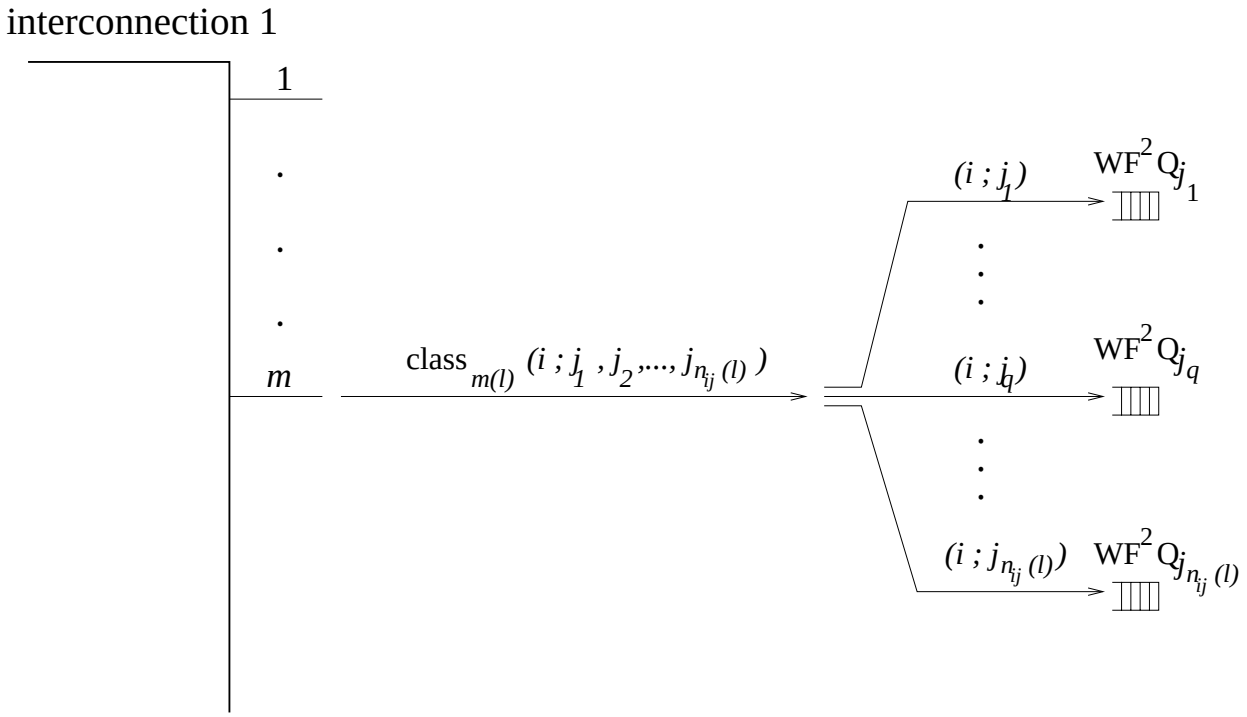

Figure 5-20: After duplication, cells at the mth subqueue, $m(l)$, of the lth class of cells with I-O pair (i, j) is replicated $n_{ij}(l)$ times.

proportional to the number of flows. Hence, there should be a total of $4N\sum_i\sum_j k_{ij}$ queues at the output of the system which corresponds to an average of $4\sum_i\sum_j k_{ij}$ per link. The buffering requirement at each output link is a function of the fanouts. Hence, there is no way of knowing the exact queueing requirement without knowing the rate matrix. Conservatively, one can use a buffer of size much larger than $4\sum_i\sum_j k_{ij}$ per link. Also, the admission controller can impose a limit on the number of classes per output link and/or fanout of each flow as well as making sure that no link is oversubscribed.

Similar to the delay, the service lag is unchanged in the presence of multicast in our model. Note that the service lag is constant (independent of the rates). On the other hand, the delay experienced by a cell is inversely proportional to its rate. These two properties are natural in some sense. For example, a G/G/1 queueing system has the same property that if the service and arrival rates are scaled by a factor, the (steady state) expected unfinished work is cut by the same factor.

We just completed the analysis of the two stage cyclic shift architecture with WF^{2}Q schedulers. We showed that with WF^{2}Q scheduling over the two stage cyclic shift architecture a service lag of $\sim 4N$ is experienced by all I-O pairs. This value is a factor O(N) improvement over the service

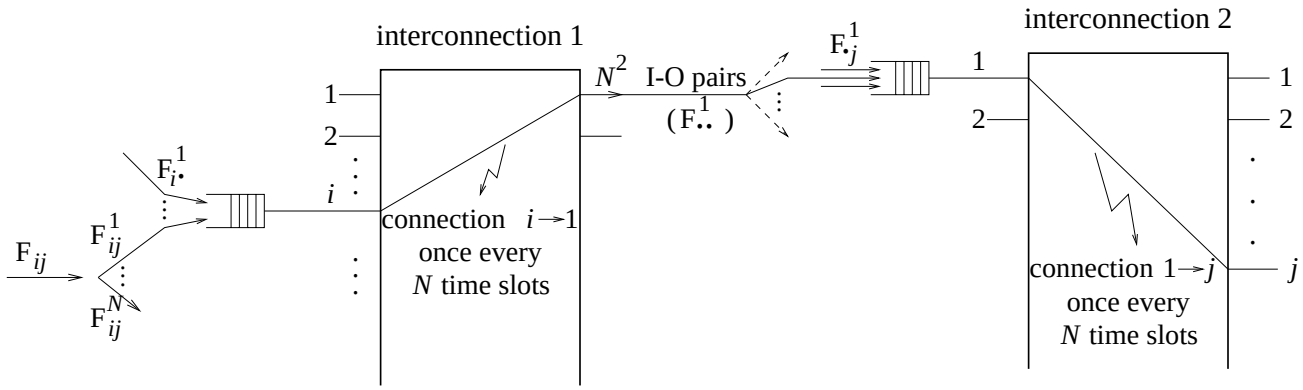

Figure 5-21: Each VOQ is served in a first in first out manner.

lag of single stage plain Birkhoff switch. Next, we derive bounds on the service lag and queue sizes with first in first out schedulers for the same architecture under both unicast and multicast traffic.

First in First out Scheme

The complete system with FIFO scheduling is illustrated in Fig. 5-21. In the first stage, each flow is divided into N subflows and those which are destined to the same intermediate link are grouped. Each group share the same FIFO VOQ and a service opportunity is awarded to the cell that is at the head of the queue whenever the appropriate switch connection is configured. Note that since there is no separation of cells with different I-O pairs, we assume the presence of a rate controller along with admission control to avoid domination by a user exceeding its rate and hurting others specified in the contract. A rate controller can be a simple leaky bucket controller whose parameters are customized for each I-O pair: The token rate is assumed to be identical to the rate specified in the contract for the I-O pair, and the bucket size is specified according to the requirements of the I-O pair. For the time being, we assume that the bucket sizes are 0 and hence the rate controllers are simple single server queues. The rate controller for the pair (i, j) should transfer a cell every R_{ij}^{-1} slots, but reciprocals of the rates are not necessarily integers. Let the controller provide a service opportunity at times $\lceil lR_{ij}^{-1} \rceil$, $l \in \mathbb{Z}^+$. It can be shown that, for any two points, t, t' in time, the number of cells, $A_{ij}^{(1)}[t, t']$, with I-O pair (i, j) emitted by the controller satisfies,

$$\left| A_{ij}^{(1)}[t, t'] - (t' - t)R_{ij} \right| < 1 \tag{5.40}$$

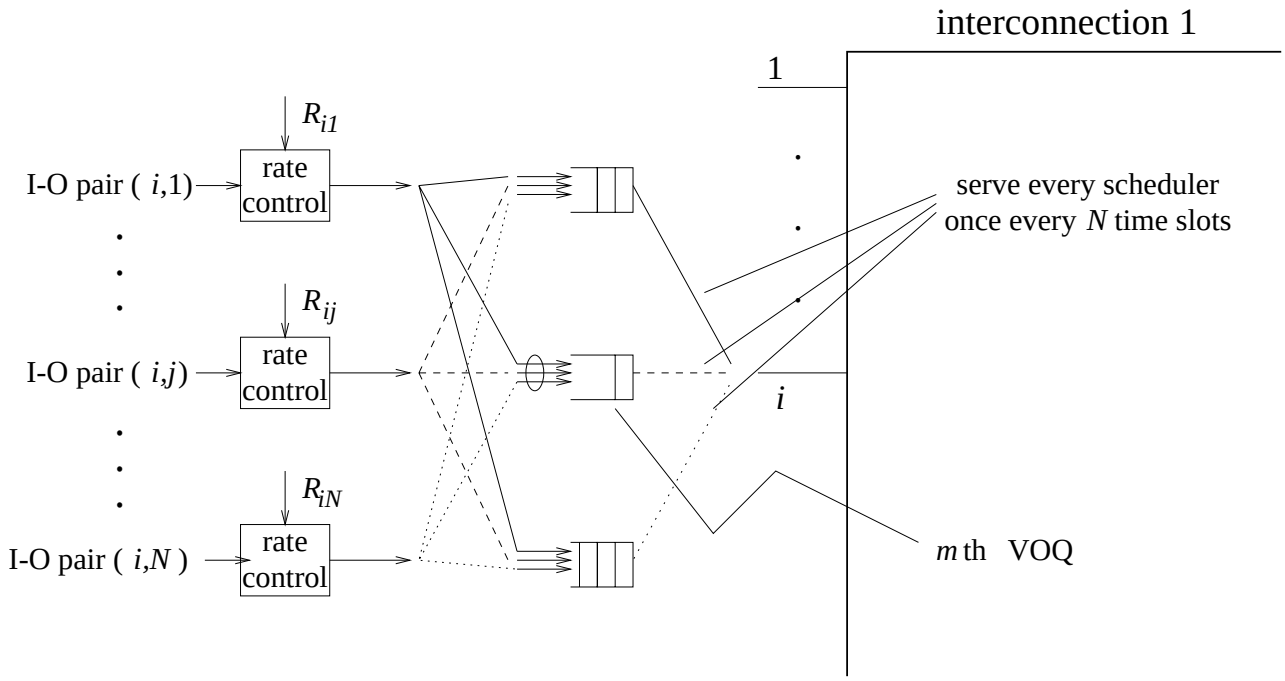

Figure 5-22: For the division process, N FIFO queues are used per link. If a cell with I-O pair (i, j) is placed into the mth VOQ, the next incoming cell of the same I-O pair will be placed in the $(m+1)$st VOQ.

Since consecutive cells are sent to consecutive schedulers as illustrated in Fig. 5-22, for any pair, m, m' such that $m \neq m'$,

$$\left| A^{(1)}_{ij,m}[t,t'] - A^{(1)}_{ij,m'}[t,t'] \right| \leqslant 1 \tag{5.41}$$

Thus,

$$\begin{aligned} (t'-t)R_{ij} + 1 &> A^{(1)}_{ij}[t,t'] && (5.42)\\ &= \sum_{l=1}^{N} A^{(1)}_{ij,l}[t,t'] \\ &= A^{(1)}_{ij,m}[t,t'] + \sum_{l \neq m} A^{(1)}_{ij,l}[t,t'] \\ &> A^{(1)}_{ij,m}[t,t'] + (N-1)\left(A^{(1)}_{ij,m}[t,t'] - 1 \right) && (5.43) \end{aligned}$$

where (5.42) follows from (5.40) and (5.43) follows from (5.41). Combining (5.42) and (5.43), we get:

$$A^{(1)}_{ij,m}[t,t'] < (t'-t)\frac{R_{ij}}{N} + 1 \tag{5.44}$$

The following holds for $A^{(1)}_{i,m}[t,t']$, the number of arrivals into the mth VOQ of the ith input link in $[t,t']$:

$$\begin{aligned} A^{(1)}_{i,m}[t,t'] &= \sum_{j=1}^{N} A^{(1)}_{ij,m}[t,t'] \\ &< \sum_{j=1}^{N} \left[(t'-t)\frac{R_{ij}}{N} + 1 \right] \qquad (5.45) \\ &= (t'-t)\frac{1}{N} + N \qquad (5.46) \end{aligned}$$

where (5.45) follows from (5.44). The first interconnection transfers a cell from the head of each VOQ to any given input of the second interconnection once every N time slots. Thus,

$$D^{(1)}_{i,m}[t,t'] > (t'-t)\frac{1}{N} - 1 \tag{5.47}$$

where $D^{(1)}_{i,m}$ is the number of service opportunities given to the mth VOQ of the ith input link. Combining (5.46) and (5.47), we get:

$$\begin{aligned} Q^{(1)}_{i,m}[t,t'] &= A^{(1)}_{i,m}[t,t'] - D^{(1)}_{i,m}[t,t'] \\ &< N+1 \qquad (5.48) \end{aligned}$$

where $Q^{(1)}_{i,m}[t,t']$ is the number of cells accumulated in the mth VOQ of the ith input link in period $[t,t']$. Assuming all queues empty initially, the number of cells in each does not exceed N at any point in time. Suppose a cell arrives at the mth VOQ of the ith input of the first stage at time t' and leaves the first stage at time t''. Then,

$$\begin{aligned} N &\geqslant D^{(1)}_{i,m}[t',t''] \\ &> (t''-t')\frac{1}{N} - 1 \end{aligned}$$

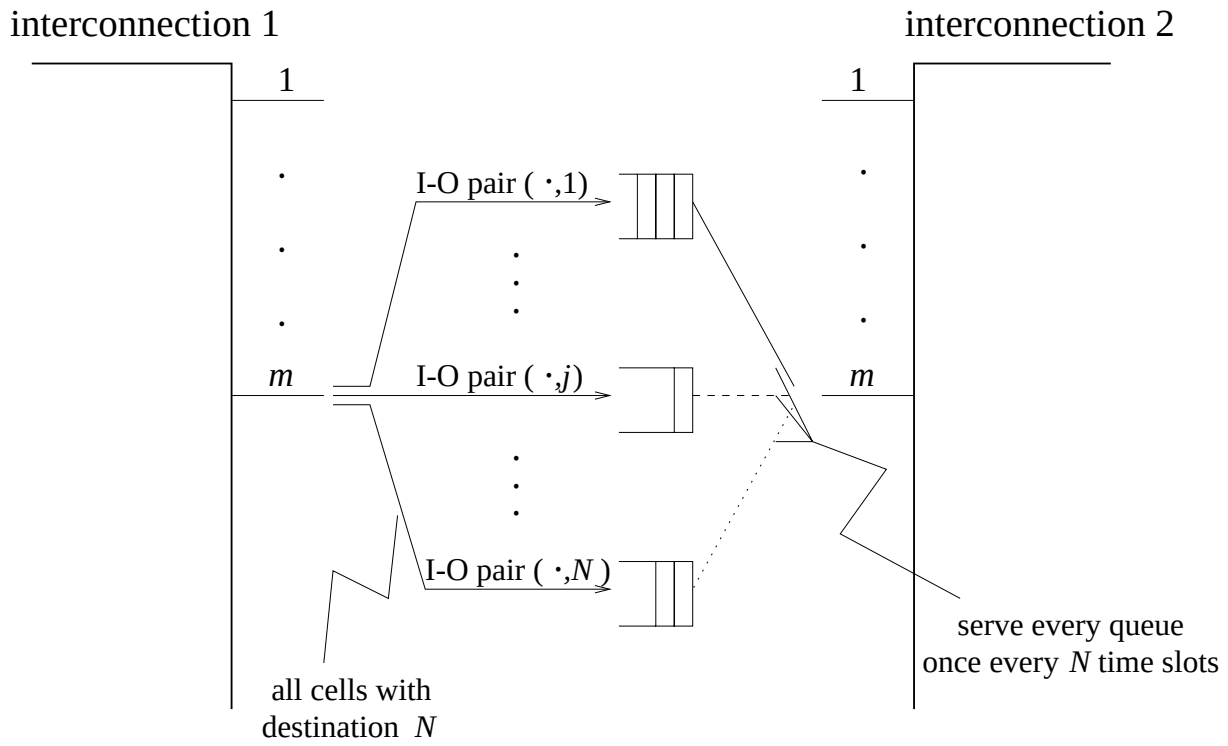

Figure 5-23: For the routing process at the input of the second stage, N FIFO queues are used per link. Cells that share the same destination share a queue.

and therefore,

$$(t'' - t') < N^2 + N$$

which implies that the delay experienced by a cell in the first stage is no more than $N^2 + N - 1$ time slots.

Next, we talk about the second stage through which cells are transferred to their ultimate destinations. The input of the second stage is illustrated in Fig. 5-23.

From some given t', let t be the last time the system was empty. The number of cell arrivals into the jth queue at link m from the ith input of the first interconnection is upper bounded by the number of service opportunities given to such cells by the first interconnection:

$$\begin{aligned} A^{(2)}_{ij,m}[t,t'] &\leqslant D^{(1)}_{ij,m}[t,t'] \\ &\leqslant A^{(1)}_{ij,m}[t,t'] \\ &< (t'-t)\frac{R_{ij}}{N} + 1 \end{aligned} \tag{5.49}$$

where $t' > t$. Hence, the total number of arrivals into the jth queue can be bounded as follows.

$$\begin{aligned} A^{(2)}_{j,m}[t,t'] &= \sum_{i=1}^{N} A^{(2)}_{ij,m}[t,t'] && (5.50) \\ &< (t'-t)\frac{1}{N} + N && (5.51) \end{aligned}$$

The number of cells served, $D^{(2)}_{j,m}[t,t']$, in that queue in the same period satisfies:

$$D^{(2)}_{j,m}[t,t'] > (t'-t)\frac{1}{N} - 1 \qquad (5.52)$$

Combining (5.51) and (5.52),

$$\begin{aligned} Q^{(2)}_{j,m}[t,t'] &= A^{(2)}_{j,m}[t,t'] - D^{(2)}_{j,m}[t,t'] \\ &< N+1 && (5.53) \end{aligned}$$

Note that $Q^{(2)}_{j,m}[t,t']$ is, in fact, the accumulation in period $[t,t']$. But, since we assumed that the system is empty at time t, Ineq. (5.53) also gives us the upper bound on the queue size for all $t' > 0$. Suppose a cell arrives at the jth queue of the mth input of the second interconnection at time t' and transfers to the output at time t''. Then,

$$\begin{aligned} N &\geqslant D^{(2)}_{j,m}[t',t''] \\ &> (t''-t')\frac{1}{N} - 1 \end{aligned}$$

and therefore,

$$t'' - t' < N^2 + N$$

which implies that the delay experienced by a cell at the input of the second stage is no more than $N^2 + N - 1$ time slots.

Finally, we talk about the delay, in the presence of multicast. Similar to WF^{2}Q scheduling, the delay performance with FIFO queueing is the same with multicast as in unicast: Bounds derived for the first stage are still valid for the multicast scenario. Since cells are not duplicated in the first stage, the scheduler is no different from that with all unicast cells, i.e., (5.48) is still valid.

Even though more than one cell arrival can happen at a time at an input of the second interconnection because of duplication, at most one arrival can occur for each VOQ. When a multicast cell arrives, the content of each VOQ increases by at most 1 cell, even though a number (the fanout of the cell) of different VOQs are incremented simultaneously. Thus, the equality in (5.50) still holds and the queue sizes do not exceed N cells.

In summary, we see that with FIFO scheduling, the delay is constant over all I-O pairs and it is upper bounded by $2\left(N^2+N-1\right)$ time slots. This is an $\mathrm{O}(N)$ improvement over the delay experienced with a plain Birkhoff switch. As explained earlier, if a cell experiences maximum delay in the first two stages, it is not delayed at the output queues to be reordered. This time, the service lag for a cell is a function of the rate of the corresponding I-O pair. For a cell with pair (i,j), the service lag can be found as follows. The number of service opportunities provided to this input-output pair between times t and t' satisfy

$$\begin{aligned} D_{ij}[t,t'] &\geqslant \left[(t'-t)-2\left(N^2+N-1\right)\right]R_{ij} \\ &= (t'-t)R_{ij}-2R_{ij}\left(N^2+N-1\right) \end{aligned}$$

and hence the maximum service lag for this pair is $2R_{ij}\left(N^2+N-1\right)$. The size of the reordering queues must be at least

$$\begin{aligned} Q_j^{(\mathrm{out})}[t,t'] &\leqslant \sum_{i=1}^{N} 2R_{ij}\left(N^2+N-1\right) \\ &= 2\left(N^2+N-1\right) \end{aligned}$$

for the jth output link.

Note that with WF^{2}Q scheduling, the service lag is constant, whereas with FIFO scheduling, the delay is constant over all I-O pairs. A careful reader may realize that there is a factor of 2 difference between the bounds on delay between the two schedulers we studied when they are averaged over all input-output pairs. That is, for a pair with rate $1/N$, the maximum delay with FIFO scheduling is half of that with WF^{2}Q scheduling. This can be explained as follows. Let us rewrite the WF^{2}Q service inequality:

$$(t'-t)\phi_m - 1 < D_m(t') - D_m(t) < (t'-t)\phi_m + 1$$

for any t, t' such that $t \leqslant t'$. For any given user m, even if one of the two bounds given above for provided service with WF^{2}Q may be tight individually, the other one is not. In fact, for some given t, if we sketch the graph of $[D_m(t') - D_m(t)] - (t' - t)\phi_m$ as a function of t', we would observe a difference of ~ 1 between the peaks. We use both inequalities to derive the delay bounds for the two stage cyclic shift system with WF^{2}Q schedulers. This implies that we overestimate the individual service lags by a factor of ~ 2, which is the factor 2 difference between the delay bounds with FIFO and WF^{2}Q schedulers.

5.5.2 Cascaded Banyan Networks

In this section, we present the performance analysis for the architecture composed of two Banyan networks in cascade as illustrated in Fig. (5-11). We assume that the network is composed of $b \times b$ crossbars. Each $b \times b$ crossbar has input buffers and there is no coordination between different crossbars. We also assume that each crossbar goes through b configurations, the identity and its circular shifts (or any other equivalent set of b configurations). The purpose of this analysis is to see whether the performance can be improved if the full division and reassembly process is implemented over multiple stages with smaller $(b \times b)$ fabrics instead of using the two stage cyclic shift architecture.

We use WF^{2}Q schedulers in each of b VOQs at the input of each crossbar. Therefore, our analysis consists of several applications of the delay bounds we derived when we analyzed full division with WF^{2}Q over the two stage cyclic shift system (5.25, 5.36 and 5.39). We show that the delay performance of the cascaded buffered Banyan architecture is worse than that of the two stage cyclic architecture. We will not give the analysis of the delay with FIFO schedulers for this architecture, but a similar degradation in the delay performance is experienced with FIFO schedulers also.

As described in the flow switching scenario, in each crossbar of the first Banyan network, cells are scheduled in such a way that those with the same I-O pair are divided over all the outputs of the crossbar and ultimately over all the output links of the first Banyan. If a cell with a given I-O pair is placed in a VOQ, the next one with the same I-O pair will be placed in the next. This process was illustrated in Fig. 5-16 for the two stage cyclic shift architecture. The difference here is that the number of subqueues scales with the number of stages. The first Banyan network (first $\log_b N$ stages) is illustrated in Fig. 5-24.

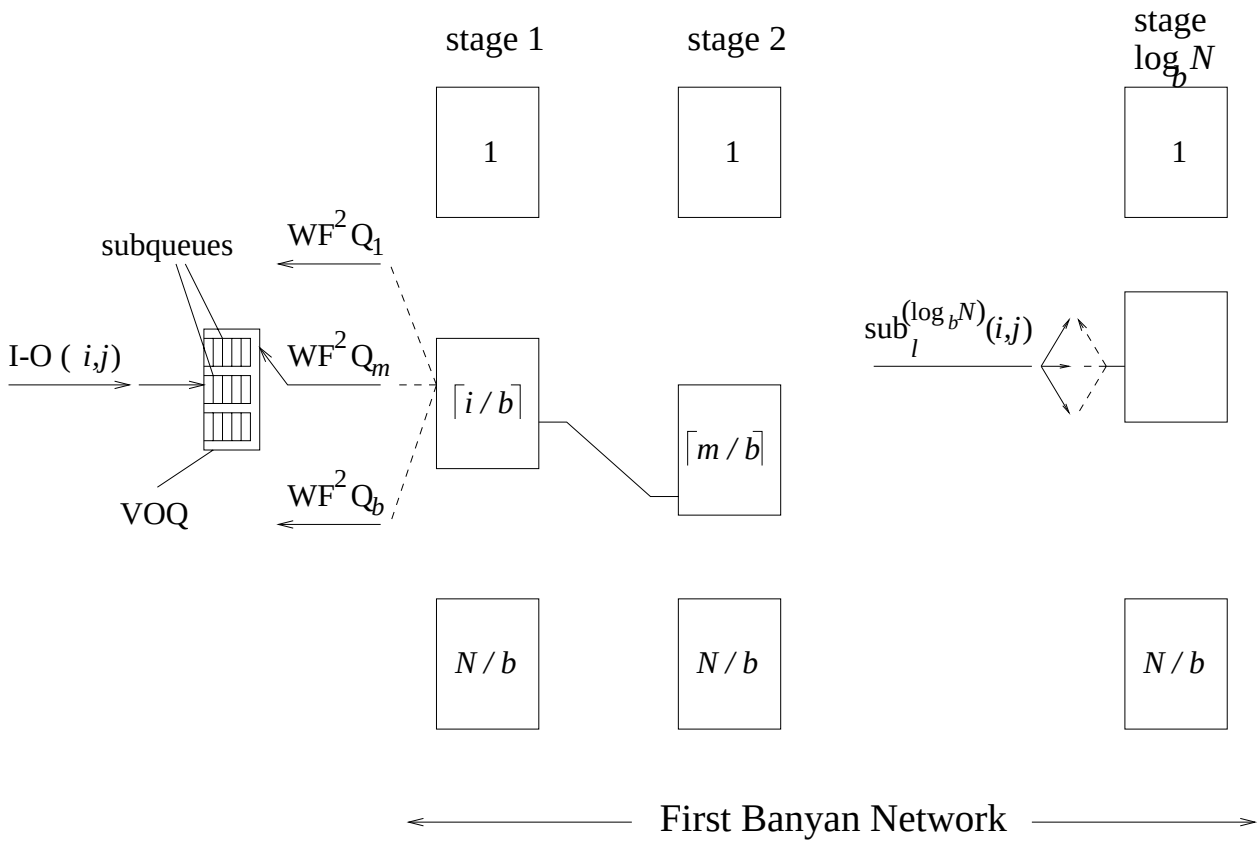

Figure 5-24: If a cell with an I-O pair is placed in a VOQ, the next one with the same pair will be placed in the next VOQ, so that cells are routed to their ultimate destinations over all the possible paths. There are b VOQs and thus, b WF^{2}Q schedulers per input link at each stage. The number of subqueues follow a geometric sequence with the number of stages.

At the lth stage of the first Banyan, each I-O pair, already split b^{l-1} ways is further split extra b ways. Hence, there are Nb^{l-1} subqueues in each VOQ at stage l, i.e., the number of subqueues follow a geometric sequence in the number of stages. Using the bound, (5.25), we found for the cell delay in the first stage of the two stage cyclic shift architecture, the following can be written for the delay at the mth subqueue that carries cells with I-O pair (i,j) at qth crossbar of stage $l \leqslant \log_b N$:

$$\tau_{ij,m}^{(l,q)} < \frac{2b^l}{R_{ij}}$$

by just replacing the N in inequality (5.25) with b^l. Thus, the total delay in the first Banyan network can be found by summing the delays in each stage on the way:

$$\begin{aligned} \tau_{ij,\mathrm{tot}}^{\text{Banyan I}} &= \sum_{l=1}^{\log_b N} \tau_{ij,m}^{(l,q)} \\ &< \frac{2b(N-1)}{R_{ij}(b-1)} \end{aligned}$$

In the second Banyan network, cells that share the same I-O pair are recombined back together as they are placed in the same subqueue in the appropriate VOQ. Note, however, that reordering is postponed until the output of the multistage switch, i.e., stage $2\log_b N$. This time the number of subqueues in each stage scale down by a factor b in each stage. Similarly, with the appropriate modifications, the delay in the second Banyan can be found using the bound we found for the cell delay in the second stage of the two stage cyclic shift architecture (5.36). The delay experienced by a cell of with I-O pair (i,j) in the qth crossbar of stage $l > \log_b N + 1$ is upper bounded as follows:

$$\tau_{ij,m}^{(l,q)} < \frac{2N^2 b^{-l+1}}{R_{ij}}$$

Thus, the following can be written for the total delay experienced in the second Banyan:

$$\begin{aligned} \tau_{ij,\mathrm{tot}}^{\text{Banyan II}} &= \sum_{l=\log_b N+1}^{2\log_b N} \tau_{ij,m}^{(l,q)} \\ &< \frac{2b(N-1)}{R_{ij}(b-1)} \end{aligned}$$

Similar to the two stage cyclic shift architecture, if maximum delay is experienced earlier, a cell is no further delayed at the reordering queues of the output of the switch. The total delay experienced

by a cell with the pair (i,j) is, hence,

$$\tau_{ij,\text{tot}} < \frac{4b(N-1)}{R_{ij}(b-1)} \tag{5.54}$$

The corresponding service lag can be found as,

$$\begin{aligned} D_{ij}[t,t'] &> \left[(t'-t) - \frac{4b(N-1)}{R_{ij}(b-1)}\right] R_{ij} \\ &= (t'-t)R_{ij} - \frac{4b(N-1)}{b-1} \end{aligned} \tag{5.55}$$

and the buffers at the output of the switch must be of size $\frac{4b(N-1)}{b-1}$ per I-O pair, i.e., $\approx \frac{4N^2b}{b-1}$ per link.

As mentioned earlier, the two stage architecture we analyzed is a special case of the cascaded Banyan architecture where $b = N$. Indeed, the bound derived for the service lag in the two stage cyclic shift architecture (5.39) can be found if the b in (5.55) is substituted with an N.

The delay experienced in this architecture depends on the size of the crossbars, i.e., choice of b. Indeed, it is proportional to $\frac{b}{b-1}$ for a given N. For $b = 2$, the delay bound is doubled compared to the two stage architecture ($b = N$); on the other hand, for small b, the crosspoint complexity (see e.g., [37]) of the switch decreases.

5.6 Routing Cells on Multiple Paths - Partial Division

In the previous section, cells with an I-O pair are transferred to their ultimate destination over all the possible paths between the pair. In this section, we will present a method by which cells are routed over a subset of the paths between I-O pairs rather than over all the paths. We will use the the two stage cyclic shift architecture (or any other architecture with N configurations that form a Latin square). Cell scheduling will not be as simple as in the full division scenario, since schedules need to be changed as a function of the rates, unlike the full division. We will show that the delay performance is improved as the number of paths decrease. However, support of multicast rates is no longer possible.

5.6.1 The Algorithm

First, let us explain what we mean by partial division. We assume that rates take on values from the discrete set[9] of rational numbers which are integer multiples of $0 < s \leqslant 1$. For a given s, we show that $R_{ij}^{(1)}/s$ is an upper bound on the maximum number of paths over which cells with I-O pair (i, j) are routed within the algorithm.

Let us study the two stage cyclic system closely. Let us represent the ith intermediate link (ith input of the second interconnection) by m_i, as illustrated in Fig. 5-4. Every time slot, m_i is connected to an input of the multistage switch through the first interconnection, and to an output of the switch through the second interconnection. In a cycle of N time slots, it gets connected to each input and output for exactly one time slot. Hence, within a cycle it has the opportunity to get one cell from each input and send one cell to each output. Also, a cell can be kept in a queue at an intermediate link until the desired output is connected. For instance, cells coming from input 1 can be kept until m_i is connected to output 3 through the second switch and then transmitted at that time, cells coming from input 2 can be kept until m_i gets connected to output $N - 1$ and so on. Hence, with some delay, any input link can transfer one cell to any output link via any given intermediate link within a *configuration cycle* of N time slots. However, within such a configuration cycle, at most one cell can be transferred to an output link from any given intermediate link. Similarly, within a cycle, each intermediate link can receive at most one cell from any given input link. That is, at most one cell can be transferred from an input link to an output link through a given intermediate link within a cycle. A sample connection pattern for link m_i during a span of N time slots is given in the following table and illustrated in Fig. 5-25.

inputs:	1	2	3	$\cdots$	N
	$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$
outputs:	3	$N-1$	6	$\cdots$	2

We see that this set of connections made through the intermediate link m_i can be represented with a permutation matrix, P_i. If the (l, k) entry is 1, then input l is connected to output k of the system through intermediate link m_i. Hence, the total service provided through all the intermediate links within a configuration cycle can be represented as the sum of N permutation matrices, $\{P_i\}_{i=1}^{N}$.

Next, suppose we have a rate matrix, $R^{(1)}$, whose entries are all integer multiples of $s = 1/N$.

[9] We will relax this assumption as we describe another architecture in the next section.

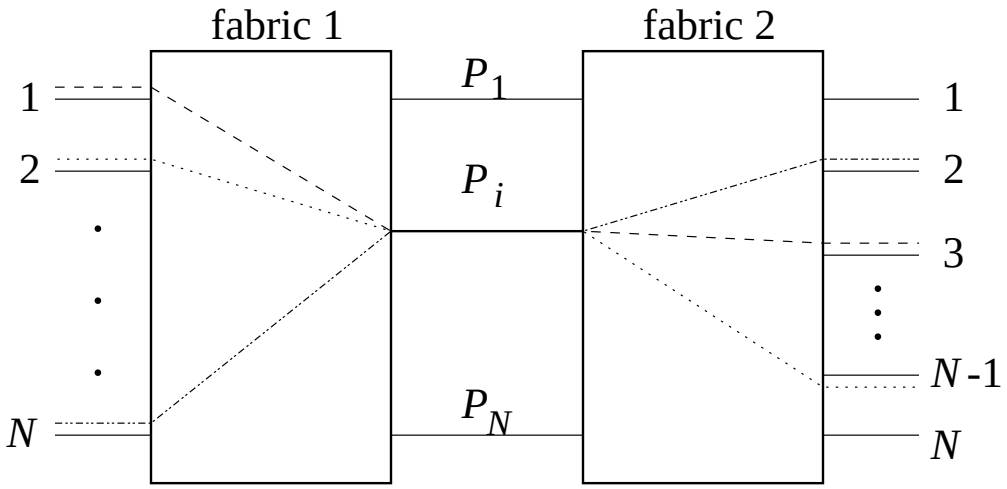

Figure 5-25: Within a cycle of interconnection configurations, the service provided by the second stage through its ith input link is P_i.

Then, we can write $R^{(1)}$ as a convex combination of N permutation matrices. Moreover, the coefficient of each permutation matrix in the decomposition is N^{-1}, i.e.,

$$\sum_{l=1}^{N} P_l = NR^{(1)} \tag{5.56}$$

To support $R^{(1)}$, link m_i can be assigned the task of providing the set of connections represented by P_i, for all $i \leqslant N$. By intermediate queueing, the two stage cyclic shift architecture can represent N permutation matrices, one each link, within a configuration cycle. This procedure is illustrated in the following example.

Example 5.5 *Suppose the switch of size* 4×4 *is to serve the set of rates given by:*

$$
\begin{aligned}
R &= \frac{1}{4}\begin{bmatrix} 1 & 0 & 2 & 1 \\ 2 & 2 & 0 & 0 \\ 0 & 2 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix} \\
&= \frac{1}{4}\left\{ \underbrace{2\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{P_1, P_2} + \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{P_3} + \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{P_4} \right\}
\end{aligned}
$$

According to this decomposition, we assign link $m_1 - m_4$ *the task of making the connections represented by* $P_1 - P_4$*. That is,* m_1 *and* m_2 *enqueue cells coming from input 1 to be released when switch 2 connects these links to output 3, and so on.*

We can represent the intermediate links through which cells are routed as the entries of a routing matrix, Ξ. The (i, j) entry Ξ_{ij} is a list of intermediate links through which cells with I-O pair (i, j) are transferred. For instance, if I-O pair (1,1) uses link m_j, i.e., $P_{11,j} = 1$, then the list in Ξ_{11} includes m_j. Let us use the link numbers instead of link IDs for convenience, i.e., use i directly instead of m_i. In Example 5.5, Ξ can be formed as:

$$
\Xi = \begin{array}{c} \\ \text{I1} \\ \text{I2} \\ \text{I3} \\ \text{I4} \end{array}
\begin{array}{c}
\begin{array}{cccc} \text{O1} & \text{O2} & \text{O3} & \text{O4} \end{array} \\
\begin{bmatrix} (3) & (-) & (1,2) & (4) \\ (1,2) & (3,4) & (-) & (-) \\ (-) & (1,2) & (4) & (3) \\ (4) & (-) & (3) & (1,2) \end{bmatrix}
\end{array}
$$

Note that, cells between I-O pair (i, j) follow $NR_{ij}^{(1)}$ paths, e.g., cells between pair $(1, 3)$ are transferred over m_1 and m_2. The entries which has a '(-)' correspond to 0 entries of matrix $R^{(1)}$.

Next we discuss some details. We know that the first interconnection periodically goes through N configurations starting with the identity configuration. We can generate a schedule matrix, $T^{(1)}$, whose entries signify the times when to schedule a service opportunity for an I-O pair within

a connection cycle for it to be routed to the appropriate intermediate link. Suppose the first interconnection is set up to have the identity connection initially, i.e., time slot 1. It connects input link i to intermediate link m_j at time $1 + [(j - i) \bmod \mathrm{N}]$ once every cycle, and thus the entries of the schedule matrix can be found using the following operation:

$$T_{ij}^{(1)} = 1 + [\Xi_{ij} - i \pmod{\mathrm{N}}] \tag{5.57}$$

In example 5.5, the schedule matrix can be found to be,

$$T^{(1)} = \begin{bmatrix} (3) & (-) & (1,2) & (4) \\ (4,1) & (2,3) & (-) & (-) \\ (-) & (3,4) & (2) & (1) \\ (1) & (-) & (4) & (2,3) \end{bmatrix}$$

The first interconnection can meet this schedule since every number between 1 and N is present in each row exactly once. Within a configuration cycle of the first interconnection, each intermediate link receives exactly one cell for each output link. Thus, the second interconnection is also able to handle the load at the intermediate links by going through the N configurations cyclically, starting with the identity configuration. We will give a more detailed explanation of these when we present the proof of the correctness of the algorithm. Before that, we would like to generalize the algorithm for some set of values of s other than $\frac{1}{N}$.

The algorithm can easily be modified to support the set of admissible rates which are integer multiples of $s = \frac{1}{kN}$ where k is an integer. Similarly, we can decompose $R^{(1)}$ as follows,

$$kNR^{(1)} = \sum_{l=1}^{kN} P_i$$

This time, we assign k permutations to each intermediate link, rather than just one. For instance, link m_i will handle matrices $P_{k\cdot(i-1)+1}$ through $P_{k\cdot i},\ i \leqslant N$. Therefore, both interconnections need to complete k configuration cycles to meet the rate requests. Hence, we define a period of kN time slots as a schedule cycle.

Also, note that, the cells with I-O pair (i, j) follow at most $kNR_{ij}^{(1)}$ paths. Thus, each interme-

diate link carries no more than

$$\frac{1}{N}\sum_{i,j} kNR_{ij}^{(1)} = \frac{1}{N}kN^2 = kN\left(=\frac{1}{s}\right)$$

I-O pairs.

Next, we present the algorithm formally and give a proof of correctness. The steps of the algorithm are as follows. Recall that the desired rates are all integer multiples of some $0 < s \leqslant 1$ where $k = \frac{1}{sN}$ is an integer.

1. The rate matrix, $R^{(1)}$ is written as a convex combination of permutation matrices and the coefficients of each permutation matrix is s:

$$R^{(1)} = s\sum_{l=1}^{kN} P_l \tag{5.58}$$

   Note that in Eq. (5.58), it is possible that $P_l = P_{l'},\ l \neq l'$.

2. Generate the matrix, Ξ, where $l \in \Xi_{ij}$ if $P_{l,ij} = 1$. Hence, each entry can be either empty or a list of numbers which take on integer values in $[1, kN]$. There will be kN elements at each row and column of Ξ.

3. Generate the schedule matrix, $T^{(1)}$, whose entries give the time slots that the input-output pairs are served within a schedule cycle as follows: If $P_{q,ij} = 1$, then, pair (i, j) should be granted a service opportunity when input i of the first interconnection is connected to input $1 + [q - 1(\text{mod N})]$ of the second, once every k switch cycles (kN time slots). Given that the first switch starts at the identity connection,

$$\vec{T}_{ij}^{(1)} = 1\vec{e} + \left[\vec{\Xi}_{ij} - i\vec{e}\ (\text{mod k})\right] \tag{5.59}$$

   where $\vec{e}$ is the vector with all unit entries. This schedule repeats every kN time slots. Note that the number of paths that the cells with each I-O pair is routed on is identical to the size of the list that is located at the corresponding position in the Ξ matrix. The list of intermediate links, O_{ij} that cells with I-O pair (i, j) is routed on can be found as follows.

$$\vec{O}_{ij} = 1 + \left[\vec{\Xi}_{ij} - 1\vec{e}\ (\text{mod N})\right] \tag{5.60}$$

Before we get to the correctness, let us illustrate this general form of the algorithm. Let

$$\begin{aligned} R^{(1)} &= \frac{1}{6}\begin{bmatrix} 1 & 2 & 3 \\ 3 & 0 & 3 \\ 2 & 4 & 0 \end{bmatrix} \\ &= \frac{1}{6}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \frac{2}{6}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} + \frac{3}{6}\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{aligned}$$

Then,

$$\Xi = \begin{bmatrix} (1) & (2,3) & (4,5,6) \\ (4,5,6) & (-) & (1,2,3) \\ (2,3) & (1,4,5,6) & (-) \end{bmatrix}$$

and

$$\begin{aligned} T^{(1)} &= \begin{bmatrix} (1) & (2,3) & (4,5,6) \\ (3,4,5) & (-) & (6,1,2) \\ (6,1) & (5,2,3,4) & (-) \end{bmatrix} \\ O &= \begin{bmatrix} (1) & (2,3) & (1,2,3) \\ (1,2,3) & (-) & (1,2,3) \\ (2,3) & (1,1,2,3) & (-) \end{bmatrix} \end{aligned}$$

Theorem 5.2 *The* partial division *algorithm supports any doubly stochastic rate matrix,* $R^{(1)}$ *with entries, each of which is an integer multiple of* $\frac{1}{kN}$, $k \in \mathbb{Z}^+$ *over the two stage cyclic shift switch.*

We will prove the statement in two steps. In the first step, we prove that the N configurations that the first interconnection goes through is sufficient to meet the schedule, $T^{(1)}$ generated by the algorithm. In the second part, we show that the *partial division* algorithm guarantees uniform loading for the second stage interconnection, whose N configurations is sufficient to support the uniform load.

Proof: *Step 1:* Since a permutation matrix has a single 1 in every row and column, every integer, $q \in [1, kN]$, can be found exactly once in each row and column of the matrix Ξ. Thus, every row of

matrix O contains each link, $m_q,\ q \leqslant N$ exactly k times. Hence, each input must be connected to each output exactly k times in a schedule cycle. Since a schedule cycle is kN time slots long and each configuration is kept $1/N$ of a cycle, the first switch will be able to meet the schedule.

Step 2: As mentioned, every row and column of the matrix, O, contains each link, $m_q,\ q \leqslant N$ exactly k times. Hence, the cells destined to a particular output, say j, are divided equally over the inputs of the second stage switch, since, as we just proved, the first switch is successful in supporting the flows according to the schedule, $T^{(1)}$. Thus, each input in the second stage is scheduled to receive exactly one cell every N time slots destined to a particular output, completing the proof.

5.6.2 Performance

In this part, we analyze the quality of service provided by the algorithm described in the previous section. In particular we will find bounds on the maximum service lag, end to end cell delay and the queue size necessary to prevent cell losses.

The most important observation is that the number of paths followed by the cells for a given I-O pair is no more than $\min\left\{N, \frac{r}{s}\right\}$, where r is the rate specified between the pair, whereas each I-O pair used all the (N) paths apriori with full division. This is the main difference between the two cases and we will show that the performance is enhanced due to the decrease in the number of paths used. In fact, there is an inverse linear relation between the number of paths used and the bounds on the end to end switch delay for a given pair.

Delay in the First Stage

Cells with an I-O pair (i, j) are divided into at most $\min\{N, R_{ij}^{(1)}/s\}$ VOQs. Let us define the service opportunities provided to cells with pair (i, j) by the first stage interconnection within the time period (t, t') to be $D_{ij}^{(1)}(t, t')$. We know that

$$D_{ij}^{(1)}((m-1)kN, mkN] = R_{ij}^{(1)}kN \tag{5.61}$$

for all $m \in \mathbb{Z}^+$. Hence, every kN time slots, the number of service opportunities provided to each I-O pair is identical to the number desired in that period. However, it may be the case that the service opportunities are provided in a burst rather than smoothly, which contributes to the service lag. To begin the analysis, we will assume that initially VOQs are empty; however,

service opportunities are given, regardless of whether VOQs are empty or not. Thus, some of the service opportunities provided in the first kN time slots may be wasted, simply because of the unavailability of cells in the VOQs. Within the first schedule cycle, at most $\frac{R_{ij}^{(1)}}{s}$ cells with I-O pair (i, j) can accumulate in the first stage. This is, in fact, the worst case since it happens if all the service opportunities are wasted within the scheduling period. Therefore, in this scenario, the delay experienced by a cell may be as high as kN time slots in the first schedule cycle.

If all the service opportunities are wasted by (i, j) cells in the first schedule cycle, no service opportunities is wasted in the next cycle since its VOQs are full. There are cells in $\min\{N, R_{ij}^{(1)}/s\}$ VOQs with I-O pair (i, j). If $R_{ij}^{(1)}/s = n$ and $n > 1$, then the first service opportunity is given to a cell with this I-O pair before time $2kN$. Indeed, one of the cells is served within $2kN - n$ time slots. The worst case delay for a cell only in the second cycle by itself is thus, $kN - \min\{N, R_{ij}^{(1)}/s\}$. Combining this with the worst case delay experienced in the first cycle, the total first stage delay is upper bounded by $2kN - 1$ using the fact that $\min\{N, R_{ij}/s\} \geqslant 1$.

Delay in the Second Stage

Let us define an interconnection cycle as the time in which an interconnection goes through all of its configurations, spending 1 time slot in each, i.e., an interconnection cycle is N time slots. Within an interconnection cycle, up to N cells arrive at each intermediate link from the first stage. If $k = 1$, then for any given intermediate link there is exactly one arrival at each VOQ by the end of first interconnection cycle. Each cell can be transferred to its output link within the next interconnection cycle, if not already transferred in the first. If $k > 1$, there will be k cell arrivals at each VOQ of a given link within a schedule cycle. There are still N cell arrivals at each intermediate link per interconnection cycle; however, these N cells are not necessarily uniform over the VOQs of the link, i.e., some VOQs may receive more than one cell and some of them may receive none. As illustrated in Fig. 5-26, all the k cells that a VOQ is supposed to receive in a schedule cycle may arrive within one interconnection cycle.

We showed that at certain points in time, the number of cells in a VOQ can be as high as k cells. It can easily be seen that this number cannot exceed k: Each VOQ in an intermediate link cannot receive more than k cells within any given period of kN time slots. On the other hand, the second interconnection transfers one cell from a VOQ once every N time slots. Thus, if there are k cells in a VOQ of an intermediate link, no new cell arrival can occur until all these k cells are

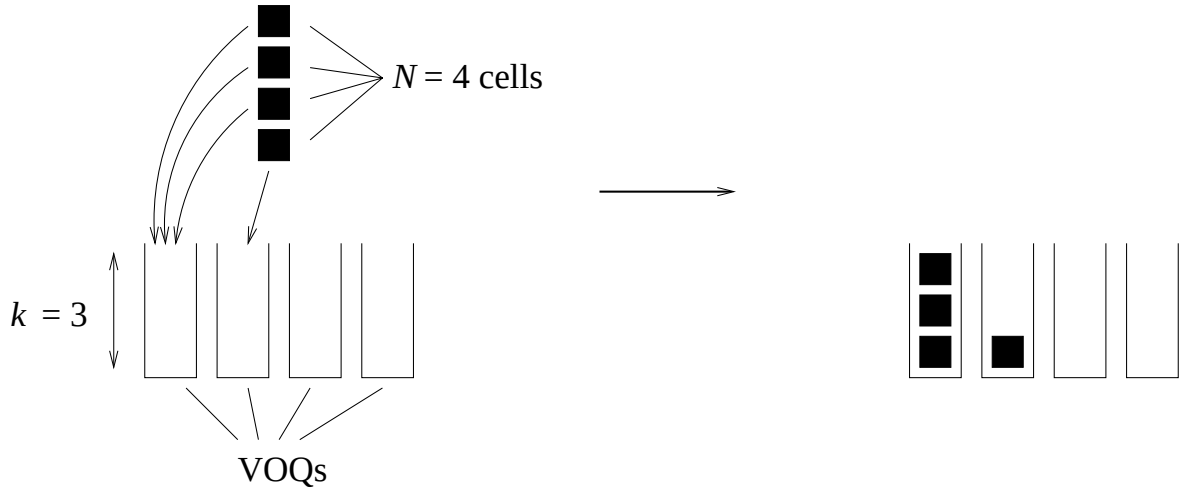

Figure 5-26: Each VOQ can receive up to 3 out of 4 cells arriving in an interconnection cycle.

transferred to the output. Therefore, the delay experienced by a cell at a VOQ in an intermediate link is upper bounded by kN time slots, and this bound is tight.

Reordering Queues and End to End Delay

The delay experienced by different cells may be different even if they arrive at the switch close to each other since they may be routed over different paths. They must be reordered at the output of the switch for packet reassembly. We just showed that the end to end delay is upper bounded by $3kN-1$. It may be the case that a cell with I-O pair (i,j) may experience the worst case delay and a number of others with the same pair go through without any delay. The latter group of cells must be enqueued at the output of the switch for as many as $3kN-1$ time slots. In this period of time, as many as $\max\left\{\left(R_{ij}^{(1)}-s\right),0\right\}\times(3kN-1)$ cell arrivals with the pair (i,j) may occur and all of them must be kept until the cell with the largest delay arrives. The term, $R_{ij}^{(1)}-s$ represents the total rate of the cells with I-O pair (i,j) that follow a different path from the cell with maximum delay. Hence the buffer at output j must be able to store

$$\begin{aligned}(3kN-1)\sum_{i=1}^{N}\max\left\{\left(R_{ij}^{(1)}-s\right),0\right\} &\leqslant 3kN-1(1-s)\\ &\approx 3kN-1 \qquad (5.62)\end{aligned}$$

cells.

To summarize, the maximum end to end cell delay with partial division algorithm over the two

stage cyclic shift architecture is $3kN - 1$ time slots and the buffer sizes at the input of the first interconnection, at the input of the second interconnection and at the output of the switch are $2kN - 1$, kN and $3kN - 1$ cells respectively. Note that the worst case delay is not affected by the possible wait it the reordering queues since, if a cell is queued there, then there is at least one cell with the same I-O pair that is ahead of that cell in arrival order which experienced more delay. That is, if a cell experiences the maximum delay of $3kN - 1$ time slots in the switch, it avoids output queueing.

5.7 Summary and Conclusions

In this chapter, we studied a number of multistage packet switch architectures, and presented different ways of providing service guarantees over them. We gave some fundamental properties of interconnections connected in cascade and developed an algebra which helped us understand how infinitely divisible fluid flows behave as they are routed inside the switch. Then, we studied relations between flow switches and packet switches, and showed that a certain set of rates cannot be supported by a packet switch unless it can be supported by the corresponding flow switch.

Motivated by this, we developed routing algorithms for flows over a number of architectures, including the two stage cyclic shift architecture and the cascaded Banyans. We applied the same ideas to packet switches and calculated bounds on the service lag and cell delay for two different cell scheduling schemes, WF^2Q and FIFO. With WF^2Q, the service lag is constant, while with FIFO, the delay is constant over all I-O pairs. The bounds on the maximum service lag and cell delay are lower by a factor $O(N)$, compared to those with the single stage Birkhoff switch. Besides, unlike single stage crossbar, the architectures we proposed are capable of supporting multicast rates without a significant modification. We worked on a number of different extensions including a routing and scheduling scheme by which the cells with an I-O pair follow only a subset of the paths between the pair and showed that the delay bounds are improved further by another ON.

A summary of the performance results is given in the following table, where r is the rate between the I-O pair under consideration, b is the size of the crossbars in the Banyan network and $k = \frac{1}{sN}$ where s is the quantization parameter.

	Queue Size (per link)	**Maximum Delay**
2 St. Cyc. - Full Div. (WF^2Q)	$6N^2$	$4N/r$
2 St. Cyc. - Full Div. (FIFO)	$6N^2$	$2N^2$
Cascaded Banyan (WF^2Q)	$6N^2b/(b-1)$	$4Nb/r(b-1)$
2 St. Cyc. - Par. Div.	$6kN$	$3kN$

In this section we considered architectures composed of interconnections put in cascade. As an extension, one can consider switch architectures composed of parallel interconnections. Indeed, some preliminary results we have illustrate that, with full division, the delay performance of certain switches with parallel fabrics are superior to those over cascaded fabrics with only a minimal increase in complexity.

As another extension, we can consider a slightly different version of the two stage cyclic shift switch. Suppose the interconnections are capable of switching their configurations only once every γ time slots (time it takes to transfer γ cells in a typical link), i.e., the link rate to configuration switch rate ratio is γ. Note, however, that they are slower only in switching configurations, but still capable of transferring a cell every time slot. Our preliminary results show that, the service lag and the cell delay scale up with γ as expected.

Finally, note that the two stage cyclic shift architecture with FIFO scheduling at the VOQs can be used along with connectionless service agreements. Other than the rate controllers, this scheme does not use the rates. Hence, without the rate controllers, this scheme works under best effort traffic and can be compared to the other algorithms designed for best effort traffic (e.g., [5]-[8]), all of which are based on the input queued single crossbar architecture. Ours is superior to them in the sense that, it, not only provides "100% throughput" like them, but also supports multicast traffic. However, the problem with not using rate controllers is that the presence of bursty and non-responsive flows degrades the performance of the switch. Thus, some sort of traffic policing is necessary for this idea to be practical.

Chapter 6

Providing Service Guarantees over Optical Switches

6.1 Introduction

Even though the optical layer has the potential to provide very high bandwidths, it is not as "agile" as the electronic layer. That is, resource assignments/reservations can be made for longer durations of time and bigger chunks of data, and it is not possible to make updates as frequently as in electronic networks. Given this limitation, an important challenge is to have the optical network provide the services desired by the users sharing the resources, and at the same time harness the bandwidth in an efficient manner. How quickly these services can be provided, and efficiency in supporting these services are challenges that are not just technological. They also involve consideration of network architecture and algorithm design issues.

We will study wavelength switches and provision of QoS over wavelength switches. We will mainly focus on two of the architectures given in Chapter 1. We review these architectures in the following section.

6.1.1 Wavelength Switches

A wavelength switch provides the possibility of routing individual channels coming from any of its input port to any output port according to the channel wavelengths. There are several architectures, depending on whether the switch is rigid, rearrangable, or strictly non-blocking. In this chapter, we will focus on two wavelength switch architectures.

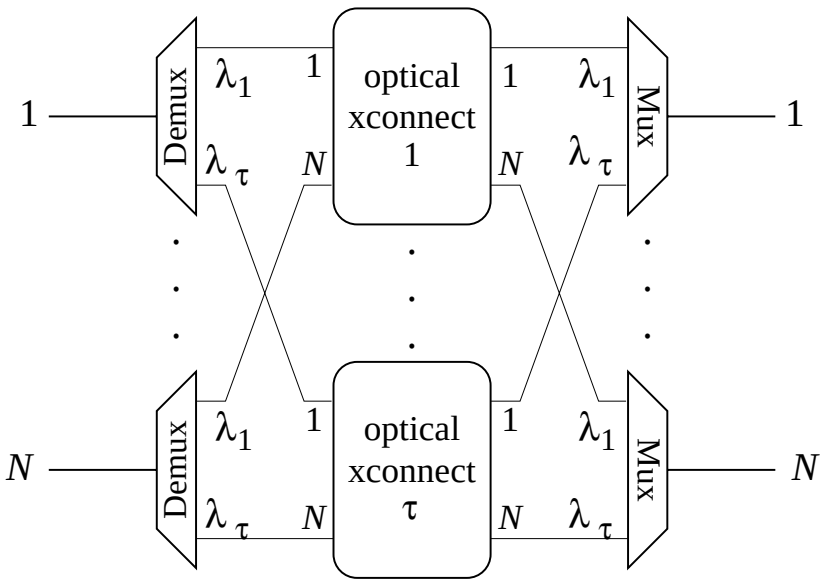

Figure 6-1: The rearrangable wavelength switch architecture.

A rearrangable wavelength switch is depicted in Fig. 6-1, where a space division switching function has been introduced using optical crossconnects (OXCs). An OXC is functionally identical to a crossbar. The only difference is that it is mainly based on opto-mechanical, acousto-optic, thermo-optic, or micro-electro-mechanical (MEMs) technologies, which are currently too slow for efficient packet switching. In this architecture (Fig. 6-1), a separate OXC is used for each wavelength. Each wavelength of any input fiber can be routed to any output fiber not already using that wavelength through one of the OXCs. On the other hand, if the wavelength is already being used at the output fiber, the input cannot use it to set up a new connection. Thus, if an input and output do not have a common unused wavelength, even if they are not fully utilized (i.e., if they are not carrying M wavelengths), a connection request cannot be met without rearranging the current wavelength assignments and crossconnect configurations. If they can be rearranged, it can be shown that the switch given in Fig. 6-1 can meet any connection request between an input and an output that are not fully utilized. This architecture is thus called rearrangably non-blocking. To make it strictly non-blocking, we can simply increase the number of OXCs inside, i.e., expand the number of wavelengths used in the system just like a Clos network.

The wavelength switch given in Fig. 6-1 involves τ OXCs each of which has a size of $N \times N$. To avoid large OXCs which may be impractical, we can introduce blocking to a certain degree. An example is illustrated in Fig. 6-2 for four nodes and 3×3 OXCs. The number of OXCs in this scheme is larger than the number of wavelengths generated at each link. On the other hand, the

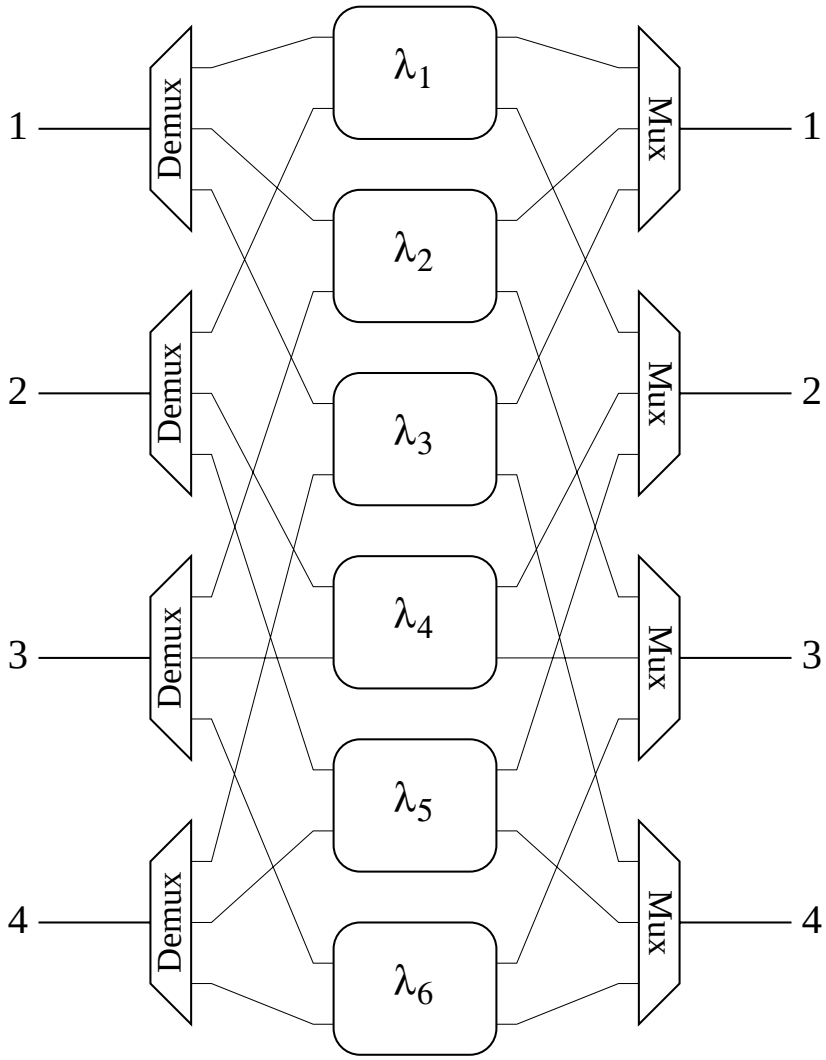

Figure 6-2: A blocking wavelength switch architecture. The size of the OXCs are smaller compared to the rearrangable version.

complexity of each OXC is cut significantly, and so is the overall crosspoint complexity. Crosspoint complexity is an important metric since the price of an OXC is roughly proportional to this metric.

6.1.2 Problem Statement

In this chapter, we consider two problems:

1. The first issue is how to support quality of service over optical networks. In particular, we will extend our previous results on rate reservation based scheduling algorithms for the optical wavelength switches. We will take the limitations of the OXCs and unavailability of optical

memory technology into account, and modify the procedure we developed for crossbar switches to make it suitable for optical wavelength switches. First, we consider a rearrangably non-blocking wavelength switch architecture. We evaluate the number of OXCs and wavelengths necessary and show how to configure these OXCs to accommodate certain traffic requirements. Next, we show that with an expansion in the number of wavelengths (and hence the number of OXCs), the architecture given in Fig. 6-1 can be made strictly non-blocking. That is, any change in the connection requirement between input output (I-O) pairs can be accommodated without a need for the rearrangement of the existing connections.

We use these insights and consider the following scenario. Suppose the rate of traffic generated by some set of end users approach optical wavelengths. We discuss how *efficiently* these end users can bypass routers as illustrated in Fig. 6-3, after deriving a relation between the the following parameters: the amount of traffic injected by end users, the number of routers connected to an optical switch and *efficiency* which we define in this chapter.

2. We consider the blocking architecture given in Fig. 6-2. We study the rates supportable by this architecture and illustrate certain trade-offs involved. In particular, we derive a relation between the number of wavelengths, number of OXCs and the *region of rates* (between I-O pairs) that are supportable over this architecture. We illustrate a system, at the center of which is the blocking wavelength switch, and find the region of rates that this system can support as a function of the number of OXCs in the switch.

6.2 Rate Guarantees over Optical Switches - An Example

First we start with an example in which we consider a packet switch. Then we focus on non-blocking and blocking switches separately.

The crosspoint complexity of an $N\tau \times N\tau$ crossbar fabric, which is the most popular choice for packet switches, is $\mathrm{O}\big((N\tau)^2\big)$. This makes the crossbar very hard to manufacture for larger switches. Instead, we can use smaller crossbars and achieve the same functionality of a crossbar over multiple stages. An example is the three stage Clos network, given in Fig. 6-4. Recall that in an $N\tau \times N\tau$ Clos network, input and output links are divided into groups of τ and each group is connected to a separate crossbar. Connections between these crossbars are made through the middle stage crossbars. Thus, the parameters are the number of links, τ per input/output crossbar,

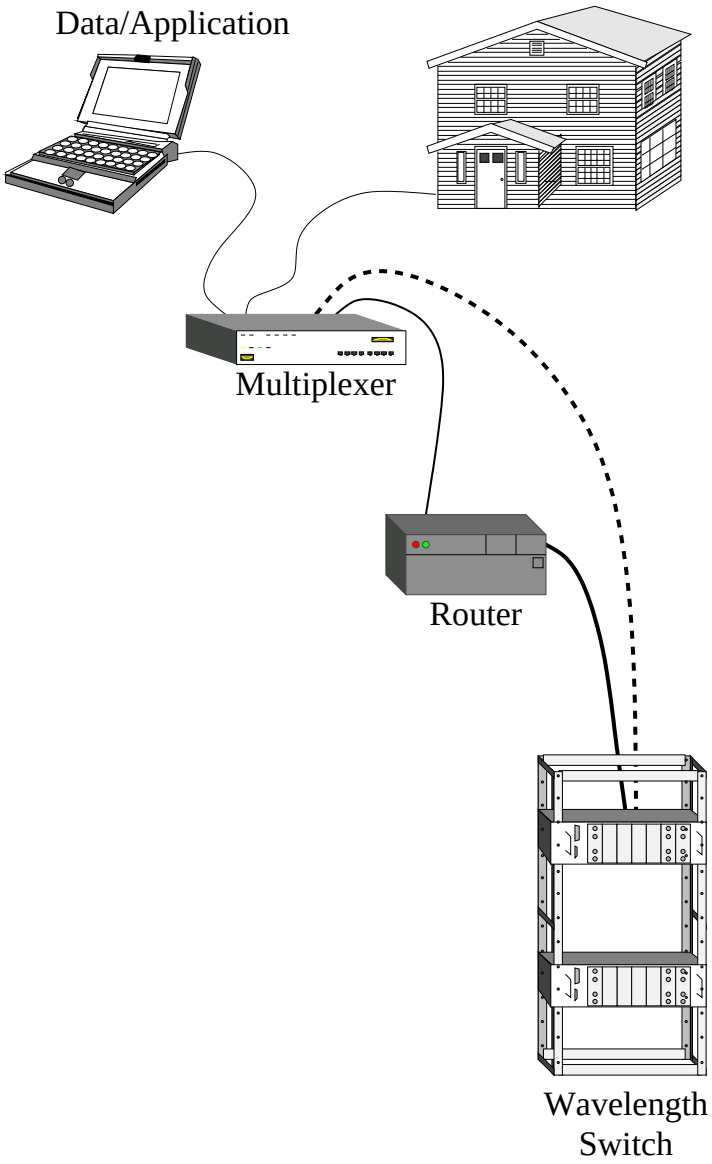

Figure 6-3: With increasing application rates, it becomes more and more feasible to move fibers closer to the end users. If the amount of traffic at some point in the edge grows to optical channel rates, routers can be bypassed.

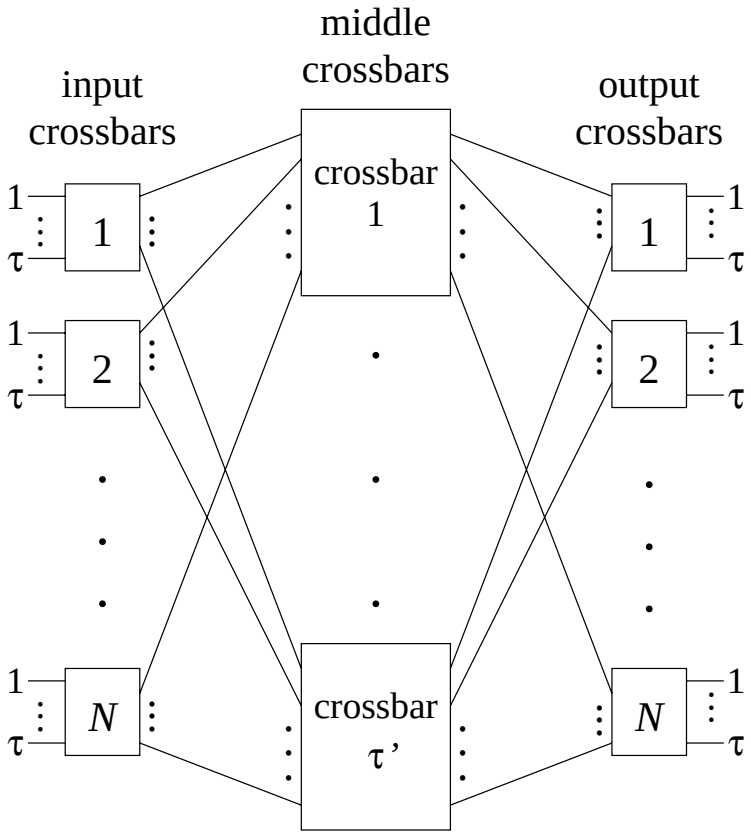

Figure 6-4: A typical three stage Clos network. The parameters are the number of links, τ per input/output crossbar, number, N, of input/output crossbars and the number, τ', of middle stage crossbars. We denote such a network by $C(\tau, N, \tau')$.

number, N, of input/output crossbars and the number, τ', of middle stage crossbars. We denote such a network by $C(\tau, N, \tau')$. As shown in Chapter 3, $\mathrm{C}(\tau, N, \tau)$ is non-blocking. The crosspoint complexity of this network is $\mathrm{O}\big(N\tau^2 + \tau N^2\big)$.

Consider an infinite duration contract with an $N\tau \times N\tau$ doubly stochastic rate matrix, R. Suppose we want to find a sequence of configurations that each crossbar in an $N\tau \times N\tau$ Clos network has to go through to support R. In particular, we will focus on the configurations of the middle stage crossbars. Let us group the entries of R as shown in Fig. 6-5, so that each group consists of τ^2 entries each of which share the same input crossbar and the same output crossbar. For instance, the $\tau \times \tau$ *sub-matrix* labeled as $(1, N)$ consists of the entries at input links connected to the first input crossbar and the output links of the Nth output crossbar. The total rates between each input output crossbar pair can be found by adding up the entries of the corresponding group of entries. Let the matrix of this aggregate rates be R_S. It is an $N \times N$ doubly stochastic matrix multiplied by a factor τ, since every row and column sums to τ. Let us apply the rate quantization

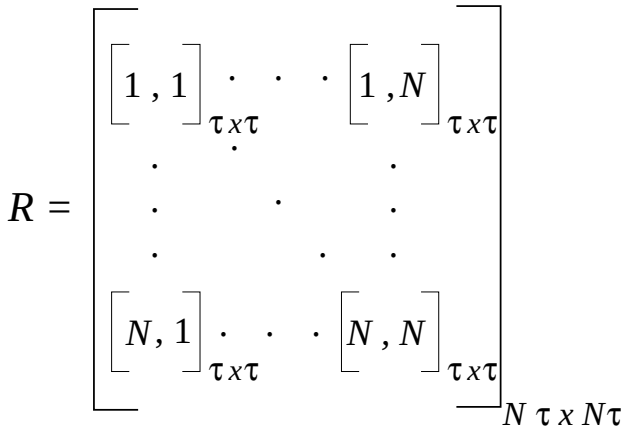

Figure 6-5: Each group of entries represents rates desired between an input and an output crossbar.

algorithm studied in the second chapter to matrix R_S with the quantization parameter $s = 1$. The algorithm will generate the matrix,

$$Q_S = R'_S + U_S \tag{6.1}$$

where R'_S is a doubly stochastic matrix scaled with a factor τ. All the entries of R'_S are integers. Matrix U_S is the constant matrix whose entries are all 1s. Thus, all the rows and columns of Q_S sum to $\tau'' = \tau + N$. One can write Q_S as a sum of τ'' permutation matrices:

$$Q_S = \sum_{i=1}^{\tau''} P_i \tag{6.2}$$

where possibly $P_l = P_m$ for some $l \neq m$. Equation (6.2) implies that, to support R, it is sufficient that the middle crossbars provide the service equivalent to the sum of $N + \tau$ permutation matrices on the right side of the equation. Thus, if we pick the number of the middle stage crossbars to be $\tau' = \tau''$, and configure the ith crossbar to P_i as illustrated in Fig. 6-6, this would be sufficient to support any doubly stochastic matix, R. Note that the middle crossbars do not need to change their configurations, or time share among a number of configurations to support a given R.

We owe this special form of decomposition to rate quantization. Indeed, without rate quantization, the number of permutation matrices required in general is O(N^2) and the coefficient of each of them is not necessarily an integer. Thus, the described one to one correspondence between permutations and crossbars would not be possible.

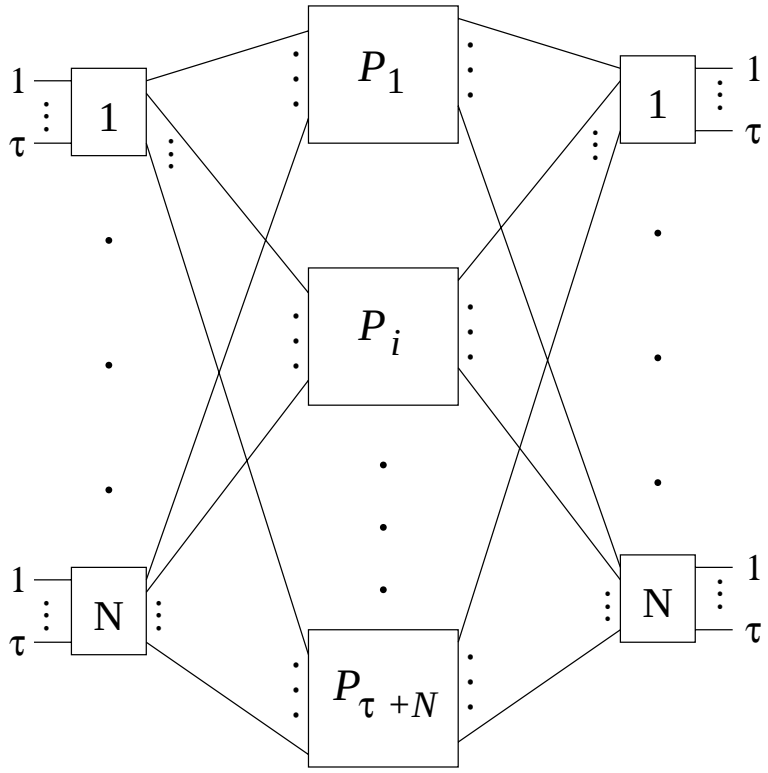

Figure 6-6: With rate quantization, the number of middle crossbars necessary to support any doubly stochastic rate matrix is $\tau + N$, and the ith one is configured to P_i of the decomposition given in Eq. 6.2.

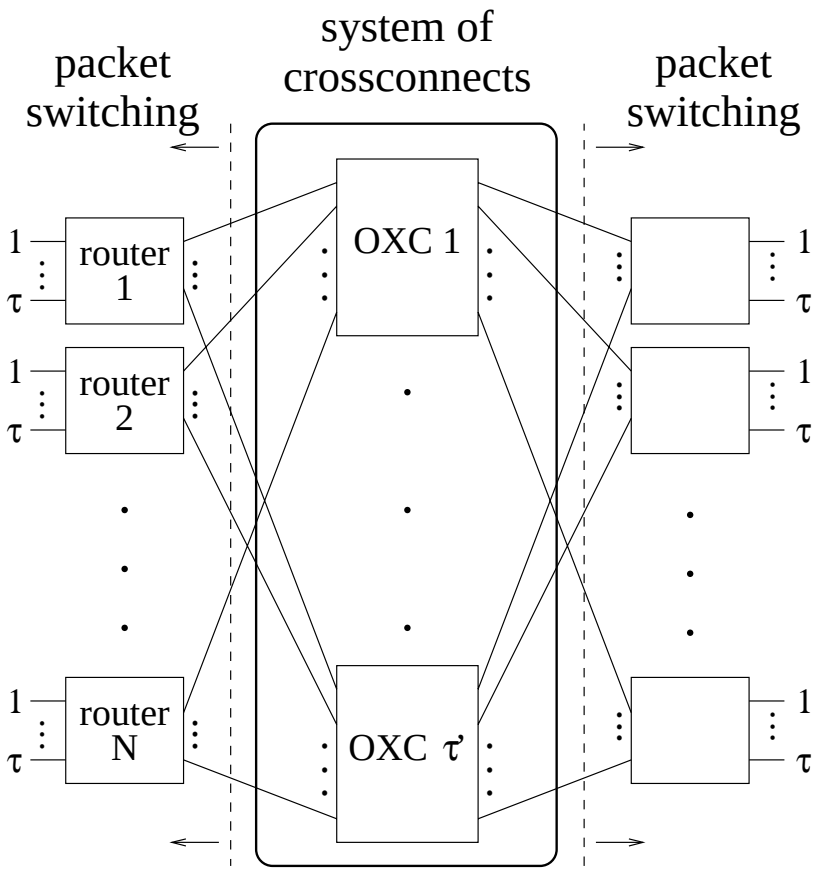

Figure 6-7: Suppose the middle stage is composed of OXCs and the surrounding (input and output stage) crossbars represent the routers we talked about in our main network architecture (Fig. 1-1).

6.3 Rate Guarantees over Non-blocking Wavelength Switches

We have just illustrated a three stage switch where end to end rate guarantees are provided over a fixed configuration middle stage. The insight we gained from this example leads to an analogy between the Clos switch and the system illustrated in Fig. 6-7. In this system, the middle stage crossbars of the Clos architecture are replaced with OXCs, and the input and the output crossbars are replaced with routers. Packet switches use the system of OXCs for transport between other routers. In a sense, this was our ultimate purpose: providing end to end rate guarantees at the router level through a backbone composed of crossconnects, whose configurations cannot be switched in short time scales (e.g., packet transmission time).

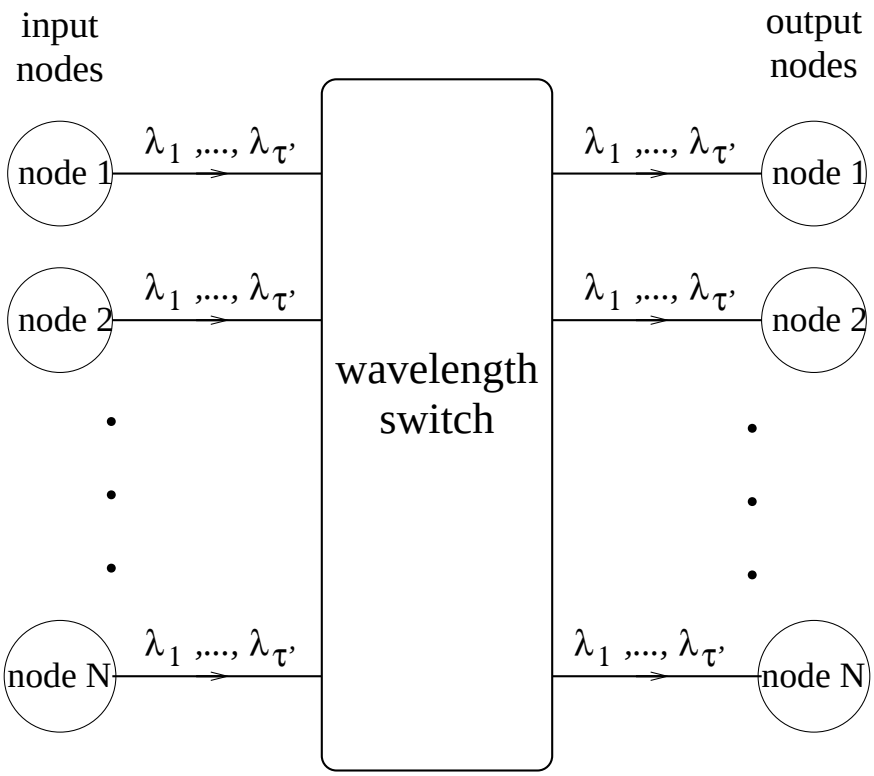

Figure 6-8: Each one of the N nodes generate τ wavelength channels worth of traffic. After rate quantization, the number of wavelengths at the input of each fiber is τ'.

6.3.1 Wavelength Assignment and Efficiency

One can realize that the structure of the non-blocking wavelength switch given in Fig. 6-1 is identical to that of the Clos network. With this in mind, we redraw the system given in Fig. 6-7 as shown in Fig. 6-8. We replace the middle stage with a single wavelength switch and the routers are shown as *nodes* that generate τ wavelength channels worth of traffic which is multiplexed[1] onto the same fiber. Note that a node can also be viewed as another wavelength switch. The output links of the wavelength switch can be connected back to these nodes, but for the analogy to be observed better, we illustrated *output nodes* separate from *input nodes*. Each input node generates τ wavelength channels of traffic. Let the rates desired between the input and output nodes be given as the entries of the matrix R. We assume the presence of a centralized admission controller to make sure that no (output) node is oversubscribed.

Applying the rate quantization algorithm exactly the same way as we did for the Clos network

[1] In fact, if we assume each link to carry one wavelength channel of traffic, τ links of one or more routers are multiplexed.

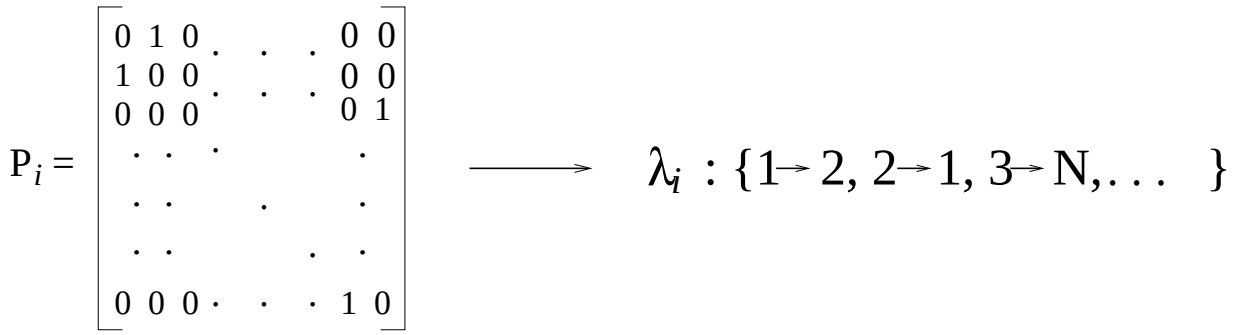

Figure 6-9: If $P_{i,lm} = 1$, then the ith OXC is configured to connect lth input node to the mth output node (through the ith wavelength).

example, we end up with the decomposition given in (6.2). Therefore, we use the non-blocking wavelength switch given in Fig. 6-1, with τ' OXCs. The wavelengths are assigned and the OXC configurations are set according to this decomposition $Q_S = \sum_i P_i$ as follows. Matrix P_i denotes the input/output permutation matrix for frequency λ_i. Thus, $P_{i,lm} = 1$ means that lth input node is connected to mth output node through wavelength i as illustrated in Fig. 6-9. Hence, the lth input of the ith OXC is connected to its mth output.

At this point let us define wavelength efficiency. Wavelength efficiency is the ratio of the amount of traffic generated in a node to the number of wavelengths (OXCs) that the switch uses. The wavelength efficiency of our system is:

$$\text{efficiency} = \frac{\tau}{\tau''} = \frac{\tau}{\tau + N}$$

The number of wavelengths is identical to the number of transceivers[2] at each node. As the amount of traffic injected by a node is increased relative to the number of nodes, our system becomes more and more efficient. We discuss efficiency further in the rest of this section.

6.3.2 Strictly Non-blocking Wavelength Assignment

We gave the conditions for strictly and rearrangably non-blocking Clos networks in Chapter 3. The rearrangably non-blocking Clos network $C(\tau, N, \tau)$ can also support any doubly stochastic rate matrix. However, to support a rate matrix, all of the crossbars that network $C(\tau, N, \tau)$ is composed of must switch their configurations from one time slot to another. This is undesirable

[2] A transceiver is a transmitter, receiver pair. For a detailed treatment of optical transmitters and receivers, see [41].

in our system since the OXCs in the wavelength switch are not capable of switching at such high rates. We showed in the previous section that rate quantization with parameter $s = 1$ eliminates the necessity for the OXCs to switch their configurations every time slot. To achieve this, $\tau' = \tau''$ wavelengths is needed.

Recall that we defined a contract as a doubly stochastic rate matrix and a duration that represents the "lifetime" of the contract. At the end of the lifetime of a contract, another one with a new duration and set of rates is negotiated. A wavelength switch with $\tau' = \tau + N$ wavelengths is rearrangably non-blocking, i.e., the switch has to find a new set of configurations, and wavelength assignments are rearranged at each input node after each rate update. In practice, the desired rates between different input and output nodes may change independently of each other. With each such change, the rate matrix changes, thus, the wavelength assignments and OXC configurations change completely. Suppose each I-O node pair updates its rate every T time slots on the average. This corresponds to an expected change of rate matrix per T/N^2 time slots. This has a significant impact on the implementation complexity of the rearrangably non-blocking system since a centralized controller is necessary. It is highly desirable that each node runs wavelength assignment algorithms independently of each other, and thus the changes in the traffic in some input nodes imply new wavelength assignment only for that set of nodes.

The quantized matrix Q_S has a row and column sum of $\tau'' = \tau + N$. As shown in Chapter 3, $C(\tau, N, 2\tau - 1)$ is strictly non-blocking. Thus, our wavelength switch becomes strictly non-blocking with $\tau' \geqslant 2\tau'' - 1$ OXCs. That is, if some of the entries of the rate matrix change, the new set of rates can be accommodated without changing the middle crossbar connections for the input and the output crossbars that keep their link rates unchanged. Thus, $2(\tau + N) - 1$ wavelengths is sufficient for *strictly non-blocking wavelength assignment*. With strictly non-blocking wavelength assignment, a change in rate requirements can be accommodated without a need for any rearrangement of the assignment for the other nodes. Hence the wavelength assignment need not be implemented centrally.

So far, we focused on a single wavelength switch and showed how to use rate quantization to provide rate guarantees over a single switch. As we mentioned before, a node connected to a switch is not necessarily a router. It can also be another wavelength switch. Hence, the algorithm can be generalized from a single wavelength switch to a network of wavelength switches. We will not talk about the details of this generalization in this paper, but make the following remarks about the

network scenario.

- We showed that the expansion in the number of wavelengths simplifies the wavelength assignment algorithms a great deal. After rate quantization, any sub-wavelength traffic initiated at a node is quantized to a full wavelength. Thus, a wavelength originated at a node does not need to be dropped at another node before its destination for further aggregation. We believe that this reduces the need for OADMs significantly. However, we may need wavelength changers in most of the nodes as shall be discussed briefly at the end of this chapter. One can notice that routing is a major issue in a system of multiple switches. There is a rich literature where the two issues, routing and wavelength assignment are considered jointly.

- For high wavelength efficiency, $\frac{\tau}{\tau+N}$ must be large. If this is not the case, we can improve efficiency by grouping nodes together and treating each group of nodes as one "supernode." Then the wavelength assignment can be implemented at the supernode level. The efficiency can be significantly increased with this slight change. For instance, if n nodes are grouped as a supernode, then

$$\text{efficiency} = \frac{n\tau}{n\tau + N/n}$$

since the number of wavelengths initiated at each supernode is n times as many as that in an ordinary node, and the number of supernodes is $1/n$ of the number of nodes.

6.3.3 Moving Optical Switching Closer to the End Users

In the network picture we presented in the first chapter, most of the end users are connected to optical switches through routers in the core. As the rates that applications desire increase and rate of traffic generated by end users approach optical wavelengths, the function of routers as traffic multiplexers start to diminish. At these rates, it starts to make more sense that end users bypass routers with a fiber link that connects them directly to the optical switches. However, it is essential that the most of the functionality of routers be transferred to the optical switches for the success of this task. In the light of the non-blocking wavelength switch example, we will discuss when this move makes sense in terms of efficiency.

Consider the system illustrated in Fig. 6-10. Suppose a set of M end users share the same optical switch through a number of routers. Let the rate of traffic generated by the ith user be

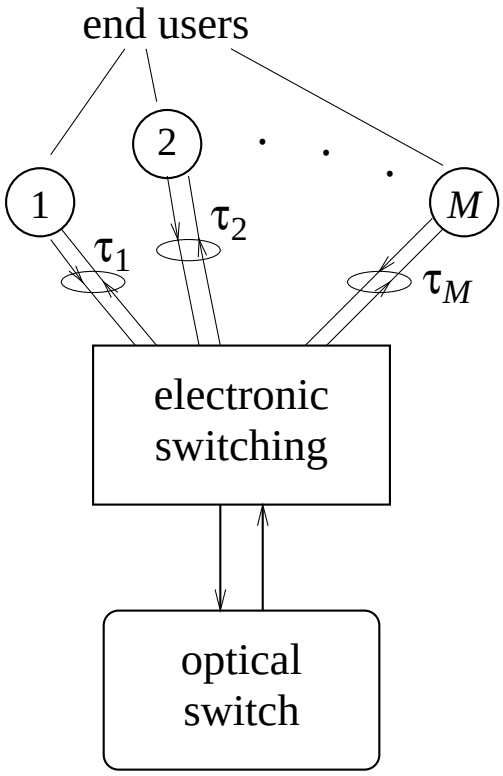

Figure 6-10: A simplistic illustration of the network architecture given in Fig. 1-1. There are M nodes connected to possibly different routers ultimately share the same optical switch.

τ_i. Now, suppose we group these end users in one optical node, and aggregate their traffic into multiple optical channels. Let the total rate of end user traffic be,

$$\tau = \sum_{i=1}^{M} \tau_i$$

Suppose we multiplex the end user traffic onto a fiber and feed it directly into an optical switch as shown in Fig. 6-11. The set of M end users forms an optical node, and the wavelength efficiency at this new node is $\frac{\tau}{\tau+N}$. If this quantity is close to 1, we say that the new node can bypass electronic switching efficiently. Note that the only parameter we considered here is the wavelength efficiency. There are many functions of the electronic layer other than multiplexing, and hence one must also think about how they can be replicated in optics.

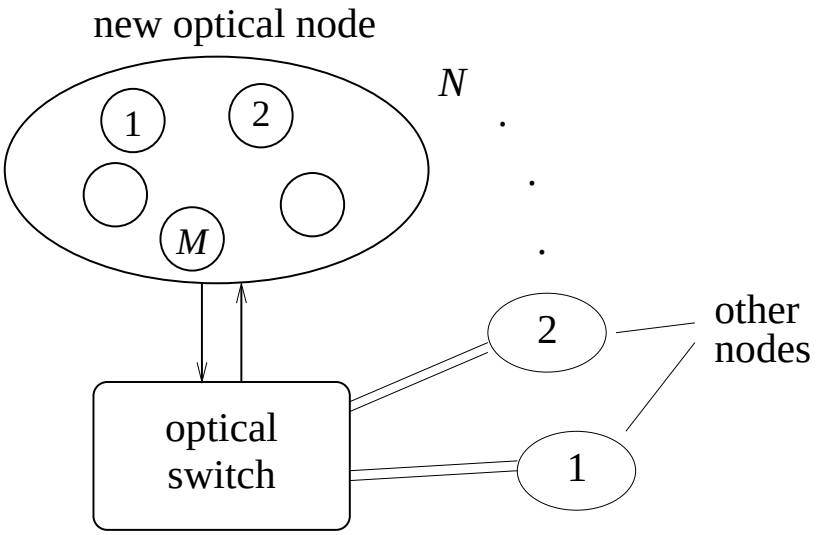

Figure 6-11: If $\tau/N \gg 1$, then the set of nodes can bypass electronic switching and get connected to the optical switch in an efficient manner.

6.4 Rate Guarantees over Blocking Wavelength Switches

The non-blocking wavelength switch given in Fig. 6-8 consists of τ' OXCs, each with a size $N \times N$. Each node has τ' transceivers, tuned to different wavelengths. This switch is non-blocking, since each node generates τ wavelengths of traffic and it can divide this traffic arbitrarily among other nodes. Namely, it can send as much traffic to another node as it desires, given that the total amount of traffic generated and terminated at each node does not exceed τ wavelengths.

In this section, we study service guarantees over blocking wavelength switch architectures such as the one given in Fig. 6-2. In this architecture, the size of an OXC is $x \times x$, $x < N$. Thus, each input and output node can be connected to only a subset of the OXCs and vice versa. There are a number of motivations for studying this blocking architecture.

1. Crosspoint complexity is an important metric since the price of an OXC is roughly proportional to this metric. As the number of nodes increases, the non-blocking switch architecture becomes more and more complex: Each $N \times N$ OXC must be non-blocking, i.e., a connection can be made between any idle input and idle output. A non-blocking OXC can be built using smaller non-blocking OXCs. For instance, if the Clos architecture is used to build $N \times N$ OXCs, the crosspoint complexity of each $N \times N$ OXC is no less than $\mathrm{O}\left(N^{3/2}\right)$ (see [37]).

2. Non-blocking behavior gives tremendous flexibility to a node in dividing the available rate among the other nodes. Indeed, as mentioned, a node can divide a total of τ wavelengths

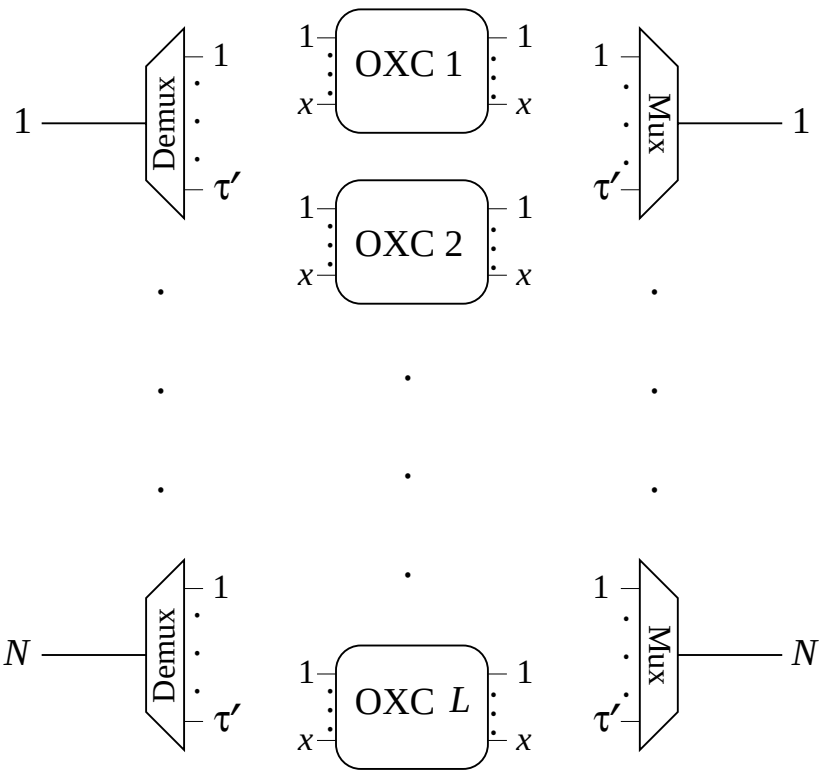

Figure 6-12: The blocking switch architecture in consideration. Each node is connected to τ' of L middle OXCs. These connections will be shown later in this section.

arbitrarily, given the admission control inequalities are not violated. However, this is not always necessarily desirable. It many optical networks, the amount of traffic between any given two nodes is no more than a few wavelengths, and it hardly comes close to the maximum capacity of τ wavelengths. A non-blocking switch may not be necessary in such scenarios.

The blocking architecture we consider has four parameters: the number of nodes, N, the number of wavelengths carried at each link, τ', the number of OXCs, L, and the size of each OXC, x. As mentioned, $L > x$, thus each input and output node can be connected to only a subset of the OXCs and vice versa. These parameters are illustrated in Fig. 6-12. The connections between nodes and the OXCs will be studied in what follows.

In this section, we analyze the rates that can be supported between an input node and the output nodes through this blocking architecture. We assume that rate quantization is applied to the set of input rates, i.e., initially τ wavelengths of traffic is generated at each node and after quantization a total of $\tau'' = \tau + N$ have to be used as shown earlier. We choose $\tau' = \tau''$ as the number of wavelengths per link like the rearrangably non-blocking switch for fair comparisons

between the blocking and non-blocking switches.

First, let us focus on the crosspoint complexity of the blocking architecture. Each input node is connected to τ' OXCs. Each OXC has a size $\frac{N\tau'}{L} \times \frac{N\tau'}{L}$. For example if L is $k\tau'$ for some $k \in \mathbb{Z}^+$, then each OXC has a size $\frac{N}{k} \times \frac{N}{k}$. Hence the size of each OXC is $1/k$ the size of an OXC used in an $N \times N$ non-blocking wavelength switch. On the other hand, there is a need for k times as many OXCs. If the Clos architecture is used to build these $N \times N$ OXCs using smaller OXCs, the overall complexity of the switch is cut by $k^{3/2}$ compared to the corresponding non-blocking switch.

Next, we specify the region of supportable rates between I-O pairs for a set of blocking switches as a function of the three parameters of the switch. From this analysis how to set the connections between nodes and the OXCs "optimally" will be apparent. We define a rate for a node as the vector whose i entry is the number of wavelengths that the node uses to send data to node i. Since there are a total of τ' wavelengths available at each node, entries of the rate vector, $\vec{r}_n$ of a given node n satisfies,

$$\sum_{i \leqslant N} r_{n,i} = \tau'$$

for all $n \leqslant N$. In this section our main tool will be the theory of majorization.

First let us describe how a region can be specified using majorization. Basic definitions and some properties of majorization are given in Appendix A. Now, we state a theorem which was developed by Schur ([25]):

Theorem 6.1 (Schur (1923)) *Suppose $\vec{z}, \vec{y} \in \mathbb{R}^N$, for some $N \in \mathbb{Z}^+$. Then, $\vec{z} \prec \vec{y}$ if and only if there exists a doubly stochastic matrix, R such that*

$$\vec{z} = \vec{y}R$$

Since any doubly stochastic matrix can be written as a convex combination of permutation matrices,

$$\vec{z} = \vec{y}\left(\sum \pi_i P_i\right) \tag{6.3}$$

$$= \sum \pi_i \left(\vec{y}P_i\right) \tag{6.4}$$

where P_i is a permutation matrix, and the summation in Eq. 6.3 is a convex combination. This

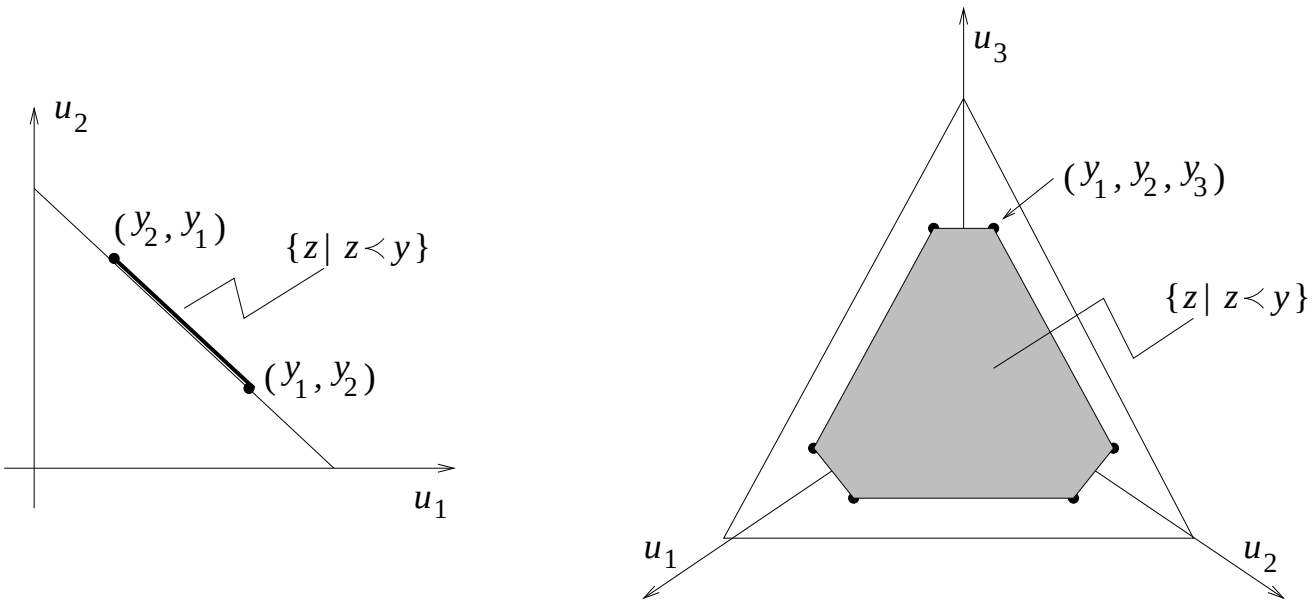

Figure 6-13: If $\vec{z} \prec \vec{y}$, then $\vec{z}$ lies in the region whose extreme points are the vectors that are formed by permuting the entries of $\vec{y}$.

means, all vectors $\{\vec{z}|\vec{z} \prec \vec{y}\}$ lie in the convex hull of $\vec{y}$ and $N! - 1$ other vectors that are formed by permuting the entries of $\vec{y}$ as illustrated in Fig. 6-13.

We design the switch in such a way that the rates supportable by a node are independent of the node number, i.e., if a given rate vector, $\vec{r}$, is supportable at a node, then $\vec{r}P$ is also supportable, for any permutation matrix, P. Hence, we only need to specify a maximal vector, $\vec{w}$, i.e., if $\vec{r}$ is a point in the set of rates supportable by a node, then $\vec{r} \prec \vec{w}$. Conversely, any rate vector $\vec{r}$ is supportable by a node, if $\vec{r} \prec \vec{w}$. To illustrate how we choose the triplet, (N, τ', L) and arrange connections between the nodes and the OXCs, we give the following theorem.

Theorem 6.2 *Let N, L and x be integers such that $L = \begin{pmatrix} N \\ x \end{pmatrix}$. There exists N subsets of the set $\{1, \ldots, L\}$, such that they all have a cardinality of $\begin{pmatrix} N-1 \\ x-1 \end{pmatrix}$, and the intersection of any m, $2 \leqslant m \leqslant x$ of these sets have a cardinality of $\begin{pmatrix} N-m \\ x-m \end{pmatrix}$.*

For a proof see [55]. Following is an example. Suppose $N = 4$, $x = 2$ and $L = \begin{pmatrix} N \\ x \end{pmatrix} = 6$. The following table illustrates how to construct these 4 subsets of $\{1, 2, \ldots, 6\}$. If there is a cross in the

(i, j) entry of this table, then the ith subset consists of a j from the set.

	1	2	3	4	5	6
1	×	×	×			
2	×			×	×	
3		×		×		×
4			×		×	×

Each element of $\{1, \ldots, 6\}$ is assigned to every possible 2-combination ($x = 2$) of $N = 4$ sets. Each pair ($m = 2$) of sets has $(\begin{pmatrix} 4 \\ 0 \end{pmatrix}) = 1$ element in common. By construction, every pair of sets has a distinct common element in $\{1, \ldots, 6\}$. The intersection of every triple is an empty set.

Now, suppose we have a system of $N = 4$ nodes each with $\tau' = 3$ transceivers tuned to three of $L = 6$ wavelengths as shown in the above table. Namely, let the numbers in the first row represent the wavelengths, the numbers in the first column represent the nodes and the crosses represent dropped wavelengths at each node. This blocking switch is, in fact, the one illustrated in Fig. 6-2. Since each pair of nodes share exactly one wavelength, each node can send up to one wavelength of traffic to any other given node. Any point in the region of rates that are supportable by node n satisfies the following:

$$\vec{r}_n \prec [1\ 1\ 1\ 0]$$

Let us go over this construction for OXCs of size $x \times x$. Since x nodes can be connected to each OXC, each column of the table consists of x crosses. Every x combination of the nodes will be matched with exactly one of the L OXCs and one of the L wavelengths is assigned to that OXC. Hence, for the wavelengths dropped at a given node, all the $x - 1$ combinations of the other $N - 1$ nodes are covered, i.e., there are $\begin{pmatrix} N-1 \\ x-1 \end{pmatrix}$ wavelengths dropped at each node (thus, each node is connected to $\begin{pmatrix} N-1 \\ x-1 \end{pmatrix}$ OXCs). Similarly, to find the number of wavelengths shared by any given two nodes, we count the number of columns where both of these nodes have a cross. For such columns, all the $x - 2$ combinations of the other $N - 2$ nodes are covered, i.e., any given two nodes have $\begin{pmatrix} N-2 \\ x-2 \end{pmatrix}$ common wavelengths. Similarly, any m, $2 \leqslant m \leqslant x$ nodes have $\begin{pmatrix} N-m \\ x-m \end{pmatrix}$ common wavelengths.

Next, we find the rate region supported by each node over the switch built using the construction

we just described. Let $\{i_1, i_2, \ldots i_m\}$ be a set of distinct indices in $[1, N]$. We define $\mathcal{L}(n; n_{i_j})$ as the set of wavelengths that are common to nodes n and n_{i_j}. Let us define

$$\mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_m}) = \bigcup_{j=1}^{m} \mathcal{L}(n; n_{i_j}) \tag{6.5}$$

Thus, from basic set theory

$$|\mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_m})| = \sum_{j=1}^{m} \left|\mathcal{L}(n; n_{i_j})\right| - \sum_{j=1}^{m}\sum_{k=1}^{j} \left|\mathcal{L}(n; n_{i_j}, n_{i_k})\right|$$

$$+ \sum_{j=1}^{m}\sum_{k=1}^{j}\sum_{l=1}^{k} \left|\mathcal{L}(n; n_{i_j}, n_{i_k}, n_{i_l})\right| - \ \cdots$$

For the architecture under consideration, for $m < x - 1$,

$$|\mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_m})| = \begin{pmatrix} m \\ 1 \end{pmatrix} \begin{pmatrix} N-2 \\ x-2 \end{pmatrix} - \begin{pmatrix} m \\ 2 \end{pmatrix} \begin{pmatrix} N-3 \\ x-3 \end{pmatrix} + \begin{pmatrix} m \\ 3 \end{pmatrix} \begin{pmatrix} N-4 \\ x-4 \end{pmatrix}$$

$$- \ \cdots \begin{pmatrix} m \\ m \end{pmatrix} \begin{pmatrix} N-m-1 \\ x-m-1 \end{pmatrix} \tag{6.6}$$

and for $m \geqslant x - 1$,

$$|\mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_m})| = \begin{pmatrix} m \\ 1 \end{pmatrix} \begin{pmatrix} N-2 \\ x-2 \end{pmatrix} - \begin{pmatrix} m \\ 2 \end{pmatrix} \begin{pmatrix} N-3 \\ x-3 \end{pmatrix}$$

$$+ \ \cdots \begin{pmatrix} m \\ x-1 \end{pmatrix} \begin{pmatrix} N-x \\ 0 \end{pmatrix} \tag{6.7}$$

Recall that $\vec{r}_{\downarrow} = \left[r_{[1]}, r_{[2]}, \ldots, r_{[N]}\right]$ denotes the decreasing rearrangement of the entries of $\vec{r}$. Since, $\left|\mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_{m-1}})\right|$ is the maximum number of full wavelength connections that could be set up between m nodes (independent of which nodes), we can write the following set of inequalities

for any given node n:

$$\begin{aligned} r_{[1]} &\leqslant |\mathcal{L}(n; n_{i_1})| \\ r_{[1]} + r_{[2]} &\leqslant |\mathcal{L}(n; n_{i_1}, n_{i_2})| \\ &\vdots \\ \sum_{j=1}^{N} r_j &\leqslant |\mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_N})| \end{aligned}$$

We have just found upper bounds on the set of rates that a node supports. Next, we show that these set of N relations can be satisfied with equality. For a node, n, let us define $\vec{v}_n$ as a wavelength assignment. The ith entry, $\vec{v}_{n,i}$, represents the set of wavelengths through which node n is connected to node i.

Nodes n and n_{i_1} can be connected by no more than $\begin{pmatrix} N-2 \\ x-2 \end{pmatrix}$ wavelengths, which are the elements of $\mathcal{L}(n; n_{i_1})$. Let these two nodes use all these wavelengths to set up $\begin{pmatrix} N-2 \\ x-2 \end{pmatrix}$ connections, i.e., $v_{i_1} = \mathcal{L}(n; n_{i_1})$ and thus the first relation is satisfied with equality. Next, choose

$$\begin{aligned} v_{i_m} &= \mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_m}) \backslash \bigcup_{j=1}^{m-1} v_{i_j} \\ &= \mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_m}) \backslash \mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_{m-1}}) \end{aligned} \tag{6.8}$$

for $m \leqslant N$, starting with $m = 2$. This is a valid assignment since

1. The same wavelength is not used for two distinct nodes, i.e., for $j \neq k$,

$$v_{i_j} \cap v_{i_k} = \emptyset$$

2. The link connected to input node n is not oversubscribed, i.e.,

$$\sum \left| n_{i_j} \right| = \tau' = \begin{pmatrix} N-1 \\ x-1 \end{pmatrix}$$

Thus, all the relations can be satisfied with equality. The maximal rate vector for node n, $n \leqslant N$

can be found as,

$$\vec{w}_n = [|v_{i_1}| \ |v_{i_2}| \ \cdots \ |v_{i_N}|] \tag{6.9}$$

Since this vector is independent of n, we use $\vec{w}$ instead of $\vec{w}_n$. The set of rates, $\vec{r}_n$ supported by node n is majorized by $\vec{w}$. Since $\mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_{m-1}}) \subset \mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_m})$,

$$|v_{i_m}| = |\mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_m})| - \left|\mathcal{L}(n; n_{i_1}, n_{i_2}, \ldots, n_{i_{m-1}})\right| \tag{6.10}$$

The values of the two parameters on the right side of Eq. (6.10) can be found using Eq. (6.6) and (6.7) as follows:

$$\begin{aligned}
w_1 &= \begin{pmatrix} N-2 \\ x-2 \end{pmatrix} \\
w_2 &= \begin{pmatrix} N-2 \\ x-2 \end{pmatrix} - \begin{pmatrix} N-3 \\ x-3 \end{pmatrix} \\
w_3 &= \begin{pmatrix} N-2 \\ x-2 \end{pmatrix} - 2\begin{pmatrix} N-3 \\ x-3 \end{pmatrix} + \begin{pmatrix} N-4 \\ x-4 \end{pmatrix} \\
w_4 &= \begin{pmatrix} N-2 \\ x-2 \end{pmatrix} - 3\begin{pmatrix} N-3 \\ x-3 \end{pmatrix} + 3\begin{pmatrix} N-4 \\ x-4 \end{pmatrix} - \begin{pmatrix} N-5 \\ x-5 \end{pmatrix}
\end{aligned}$$

and so on[3].

We just found the region of rates that are supportable by any given node each of which originates $\tau' = \begin{pmatrix} N-1 \\ x-1 \end{pmatrix}$ wavelengths of traffic to be distributed by the blocking switch given in Fig. 6-12. The set of rates, $\vec{r}_n$ supported by any given node $n, n \leqslant N$ satisfies $\vec{r}_n \prec \vec{w}$ where $\vec{w}$ is given in Eq. 6.9. Note that $\vec{w}$ majorizes the vector of rates not only from an input node but also to an output node. Hence if we denote the number of wavelength connections between input node i and output node j as the (i, j) entry of a rate matrix R, then each row and each column of R is majorized by $\vec{w}$. Moreover, any rate matrix whose rows and columns are majorized by $\vec{w}$ can be supported by the blocking switch.

In this section, we presented a method to construct a blocking $N \times N$ wavelength switch using

[3]The coeficients of the above combinatorial terms follow a pattern which is called the Pascal triangle. The coefficients of the terms of w_k are identical to the coefficients of the polynomial $(x-1)^{k-1}$.

$x \times x$ OXCs. While the crosspoint complexity of this switch is $\sim \left(\frac{x}{N}\right)^2$ of a non-blocking switch, the maximum number of connections that could be made between any two nodes is reduced by a factor $\frac{x-1}{N-1}$. One can introduce a *speedup* of $\frac{N-1}{x-1}$ by expanding the number of transceivers at each node so that the system supports the set of rates that a non-blocking switch supports. Note that the crosspoint complexity of the blocking switch with this speedup will still be smaller than that of the non-blocking switch. The ideas we developed this section can be used for other architectures as well. An example is given in the final section of this chapter.

6.5 Summary and Future Extensions

In this chapter we illustrated how to use the rate quantization procedure and provide rate guarantees over wavelength switches. The procedure is an extension of the rate quantization algorithm we used for electronic switches. Thus, we showed that rate quantization can be used in multiple layers, and we can integrate the optical and the electronic layers in an interoperable and compatible manner. Rate quantization also cuts the need for OADMs since it eliminates the need for subwavelength processing in an optical switch.

First, we illustrated how to provide service guarantees over non-blocking optical switches. Then, we focused on a certain blocking switch, and studied the supportable rate region for this switch. It is useful in understanding the relation between degree of blocking and the region of rates that could be supported by a certain set of blocking switches. Then, we discussed how "efficient" it is to move an initially electronically switched node to the input of an optical switch.

We considered the wavelength assignment problem in an optical switch independent of those of other optical switches. In a network of multiple switches, if a new wavelength needs to be assigned to a lightpath, it needs to be done in all the switches on the path. Thus, unless a wavelength changer is used, the wavelength assignment could not be done independently in different switches. We need to consider the problem over the entire network. This problem is regarded as the *routing and wavelength assignment* problem, which could be a very important future extension for this work.

We can extend the result we have on blocking wavelength switches for other architectures. One example is the ring, which is one of the most popular architectures for optical networks. Today, most of the physical layer infrastructure is built around rings. If a single wavelength is used in

a unidirectional[4] ring, only one node can achieve full duplex communication with another node at a time while multiple wavelengths enable many nodes to communicate simultaneously. Using wavelength multiplexers, multiple rings can be supported over the same infrastructure. For any two users to communicate, they must be able to add and drop a common wavelength. Hence, they both need an ADM tuned to the same wavelength.

Suppose we have a constraint that each node can have no more than L ADMs. We believe that an important problem is how to tune these L ADMs at each node to achieve certain traffic requirements. For instance, if a node can send up to L wavelengths of traffic to another node at a time, then the ADMs at each node must be tuned to same set of L wavelengths. In that case only a total of L wavelengths of traffic can be supported by the ring network. However, if there is a bound on the amount of traffic between each pair of nodes, we may use the following modification. Let us use a total of M, $M > L$ wavelengths, and tune each one of L ADMs in such a way that traffic requirements can be met. This division can be made similar to that described in Section 6.4 for blocking wavelength switches. By doing this modification, we increased the total amount of traffic (in the number of wavelengths) supportable by the ring from L to M, without changing the cost of the network (no increase in number of ADMs or number of transceivers used at each node). For instance in a 3-node ring network, if $L = 2$ and if at most 1 wavelength of traffic is required between each pair of nodes, we can tune the ADMs as illustrated in Fig. 6-14 so that a total of 3 wavelengths of traffic can be supported by the ring network. Each node can add and drop a different pair of wavelengths so that each pair of nodes shares a distinct common wavelength.

[4]Data flows in one direction only, i.e., clockwise or counter clockwise.

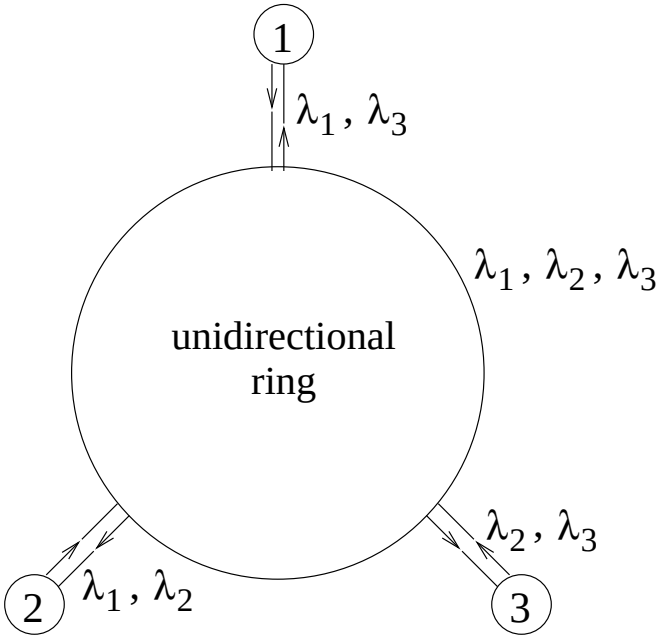

Figure 6-14: A three user unilateral optical ring which supports three wavelengths. Each node can add and drop a different pair of wavelengths so that each pair of nodes has a distinct common wavelength.

Chapter 7

Conclusions

7.1 Summary

In this thesis, we considered several issues related to providing quality of service over electronic and optical switches. In this section, we present a summary of some of the results.

First, let us focus on packet switches. We give a comparison of some of the architectures and algorithms we developed and those given in [15].

1. **Quality of Rate and Delay Guarantees**

   - *Single Stage Crossbar Switch with Birkhoff's Algorithm*: Service lag is upper bounded by N^2. We showed that there exist certain rate matrices for which this is tight for some I-O pairs. We also showed that the cell delay can be as high as $O(N^3)$.
   - *Single Stage Crossbar Switch with Rate Quantization*: Rate quantization improves the service lag to approximately N/speedup with very simple schedulers (e.g., AOS). We showed that this bound can be tight for some I-O pairs. We believe it can be further improved with more sophisticated schedulers.
   - *Two Stage Cyclic Shift Architecture*: Two main methods to implement full division over the two stage cyclic shift architecture are WF^2Q and FIFO queueing.

     With WF^2Q, the service lag is constant over all I-O pairs and it is upper bounded by approximately $4N$ which is a factor $O(N)$ improvement compared to the single stage plain Birkhoff switch. This bound is not necessarily tight, and we argued that it is approximately a factor 2 greater than the actual figure. This implies that the cell delay

is upper bounded by $4N/r$ for an I-O pair with rate r.

With FIFO, the end to end cell delay is constant over all the I-O pairs and it is upper bounded by approximately $2N^2$ time slots. This bound is tight. It is an $O(N)$ improvement over the delay with a plain Birkhoff switch. The corresponding service lag is $2N^2r$ for an I-O pair with rate r.

We also showed that, with the help of rate quantization, the partial division algorithm improves the delay bound to $3kN$ where $k = \frac{1}{sN}$ is the rate quantization parameter.

- *Cascaded Banyans with Internal Buffering*: We used WF^2Q scheduling to implement full division over the cascaded Banyan architecture. The service lag is constant over all I-O pairs and it is upper bounded by $4N\frac{b}{b-1}$ where b is the size of the crossbars each Banyan is composed of. This is a factor $\frac{b}{b-1}$ worse than the service lag with the two stage cyclic shift system.

2. **Algorithm Complexity**[1]

- *Single Stage Crossbar Switch with Birkhoff's Algorithm*: Birkhoff's decomposition algorithm has an $O(N^{4.5})$ computational complexity. The decomposition algorithm cannot be run simultaneously as the crossbar is configured. It needs to be complete before the first configuration can be set.
- *Single Stage Crossbar Switch with Rate Quantization*: Service is periodic. In the first period, computational complexity is $O(N^{2.5})$ per time slot. The same schedule is repeated each period for the contract duration.
- *Two Stage Cyclic Shift Architecture*: For this architecture, switch scheduling is trivial. Both fabrics go through the same set of configurations regardless of the set of rates. The complexity associated with the calculation of WF^2Q schedules for full division is $o(N)$ per scheduler (see [23]). There are N schedulers per link, but since they have the same set of weights, no more than one schedule needs to be generated at each link. Hence the complexity is $o(N)$ per link.

  On the other hand, FIFO scheduling is trivial. The two stage cyclic shift system with FIFO schedulers only needs to keep track of which intermediate link each incoming cell is to be forwarded to, and no further calculation is necessary.

[1] **Here, we deal with time complexity. The time complexity of an algorithm is the number of serial steps that the algorithm goes through.**

The complexity associated with partial division for cell scheduling is identical to the scheduling complexity of the single stage crossbar switch with rate quantization, i.e., $O(N^{2.5})$ per time slot in the first schedule cycle which is repeated afterwards.

- *Cascaded Banyans with Internal Buffering*: Switch scheduling is trivial for this architecture too. All $b \times b$ fabrics go through the same set of configurations regardless of the set of rates. There are b WF^2Q schedulers at each link, and the complexity associated with the calculation of WF^2Q schedules varies from one stage to another. At each stage, it is $o(N)$ per link.

3. **Adaptation to Changing Traffic**

   - *Single Stage Crossbar Switch with Birkhoff's Algorithm*: Each time the rate matrix changes, the decomposition must be rerun. There is no saving in complexity even if the change in the rate matrix is small.
   - *Single Stage Crossbar Switch with Rate Quantization*: We showed that with each single change, schedule rearrangements can be made with an $O(N)$ algorithm based on the Slepian Duguid theorem for circuit rearrangements in Clos networks. No speedup is required beyond that necessary for rate quantization. With an extra speedup of 2, we showed that strictly non-blocking switch scheduling is possible. With strictly non-blocking scheduling, updates can be completely independent for all the I-O pairs.
   - *Two Stage Cyclic Shift Architecture and Cascaded Banyans with Internal Buffering*: Since switch schedules are constant; they do not need to be changed with changing rates. With full division, rate updates do not incur an extra cost over $o(N)$.

     With partial division, the same rearrangement algorithm that is used for the single stage crossbar switch with rate quantization can be used, i.e., the complexity associated with schedule updates is $O(N)$.

4. **Multicast Support**

   - *Single Stage Crossbar Switch*: Unlike unicast, multicast rates are not supportable over single crossbar switches without speedup. Using the Clos network analogy we showed that a speedup of approximately $O(\log N)$ is necessary for multicast support.
   - *Two Stage Cyclic Shift Architecture and Cascaded Banyans with Internal Buffering*:

With full division, support of multicast rates is possible without degrading the performance.

Next, let us recapitulate some results regarding optical switches. We illustrated how to use the rate quantization procedure to provide rate guarantees over optical switches. The algorithms were extensions of those we used for electronic switches. Thus, we showed that rate quantization can be used in multiple layers, and the optical and the electronic layers can be integrated in an interoperable and compatible manner: If rate quantization is used in both the electronic and the optical switches, with the appropriate choice of the quantization parameters, end to end rate guarantees can be given over an optical switch with fixed configuration OXCs. For quantization, a factor $1+N/\tau$ expansion in the number of wavelengths is necessary, where N is the number of optical nodes and τ is the rate of traffic generated at each node in number of wavelengths. Then we studied a certain blocking optical switch architecture. We showed that the complexity of a wavelength switch can be reduced by a factor x/N, at the expense of a factor x/N decrease in the number of wavelength connections that could be set up between any given two nodes.

7.2 Further Directions

In most of the previous chapters, we showed several directions for future extensions. We will not repeat them here, but rather discuss some broader issues.

In the context of providing quality of service, two different issues are considered in most of the literature. The first issue deals with traffic variations. A number of users are multiplexed at each input link of a switch which is assumed to act as a constant rate pipe. Two major questions are, what kind of probabilistic or deterministic service guarantees can be given to each user, and how can the available rate be divided among users so as to satisfy certain quality of service requirements for each user?

The second issue deals with how a switch can provide the constant rate service that the group of users desire. Switch constraints, such as the crossbar constraint, are taken into consideration and different algorithms are developed subject to these constraints.

These two problems are considered separately in most of the literature. We have considered the second issue in this thesis, but we believe that we have some results which may be a good starting point for the joint consideration of these two issues. For instance, with the algorithms we presented in Chapter 3, the variations in a traffic source affect the scheduling for other sources

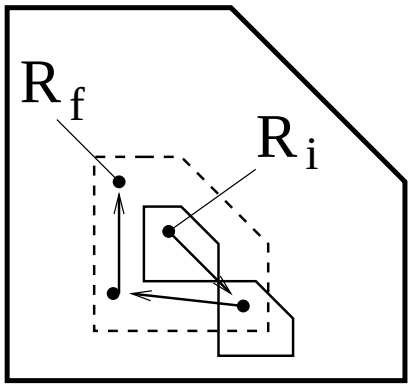

Figure 7-1: With increased quantization parameter, the representation regions whose boundaries are shown by small pentagons expand. The boundary of a new representatation region is illustrated with dashed lines.

only minimally. Still, the two issues can be combined further. We could choose the quantization parameter in such a way that takes the variations of traffic sources into consideration. Following is an example for this.

Suppose, at some point, it is desired that the contract durations be increased. However, the source variations are rapid, i.e., they occur in short time scales and sources need short contracts. In this scenario the contract durations can be increased by increasing the quantization parameter as illustrated in Fig. 7-1. A rate update (and hence a new contract) is needed every time the rate point moves from one representation region into another. By increasing the quantization parameter, the regions are enlarged and the frequency of region changes are reduced as illustrated. As the rate point moves from R_i to R_f, three changes occur in the region with the smaller quantization parameter and no change occurs with the larger parameter. The tradeoff is lower switch throughput (or equivalently a need for higher speedup). Naturally the switch performance is tied to source characteristics.

In most of this thesis, we focused on the problem of providing service guarantees by a single switch. A natural and important extension is providing service guarantees over a network of switches. In the network scenario, resource assignments or updates at different switches must be considered jointly. Also, routing is a major issue. Resource assignments and updates and routing in networks are generally dependent. Thus, consideration of routing and resource assignment jointly lead to better performance. An example is the *routing and wavelength assignment* problem in optical networks (see e.g., [41]). We believe that Chapters 5 provides insight into how certain routing and scheduling algorithms perform in networks of switches.

A final issue that can be considered as an extension is blocking switching fabrics. In general, blocking in switching fabrics is defined as the failure to satisfy a connection requirement because all paths for that connection conflict with paths inside the fabric for some set of connections between different I-O pairs. For instance, a fabric for which I-O pairs (i, j) and (i', j'), $i \neq i', j \neq j'$ cannot be made simultaneously connected is called blocking. An $N \times N$ fabric is said to be blocking if there exists a set of distinct inputs $\{i_1, \ldots, i_N\}$ and a set of distinct outputs $\{j_1, \ldots, j_N\}$ such that connections $(i_1, j_1), \ldots, (i_N, j_N)$ cannot be made simultaneously. There is a one to one correspondence between such sets of N I-O pairs and permutation matrices; indeed, the permutation matrix corresponding to such a set is the one where there is a 1 in locations corresponding to each pair in the set. Hence, a fabric is blocking if the number of distinct *feasible matchings* between I-O pairs is less than $N!$.

Non-blocking fabrics may become increasingly complex as the size of switches grow. For instance, the crosspoint complexity of a crossbar is $O(N^2)$, and it gets harder to manufacture a crossbar for larger switches and high speed applications. There are many architectures such as the ring, the bus or some multistage switches (e.g., Banyan) that are blocking when run without speedup. In many cases some *resource speedup* is introduced to achieve the non-blocking property in these networks. For instance in an optical ring network, more than a single wavelength is used so that multiple nodes can communicate simultaneously. Another example is a wavelength switch which was studied in Chapter 6.

While the problem of providing guaranteed quality of service has been explored for non-blocking fabrics under unicast traffic extensively, it has not been considered for switches with blocking fabrics in general. We can examine the relation between the degree of blocking and the amount of resource speedup necessary for such fabrics to possess the capabilities of non-blocking and multicast enabled fabrics. Namely, we can specify the amount of extra speedup necessary for a blocking fabric to exactly mimic the behavior of a non-blocking fabric. To illustrate what we mean by this, we present the following example. For a 4×4 fabric, assume the configuration corresponding to

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

is not physically possible, but

$$P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, P_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

are. Then, P_2 and P_3 can cover for P_1 if they can be in effect simultaneously (in the same time slot), since I-O pairs $(1, 1)$ and $(2, 2)$ can be connected through P_2, and I-O pairs $(3, 3)$ and $(4, 4)$ through P_3. There is a need for a speedup of 2 so that the absence of P_1 does not affect the non-blocking behavior of the fabric. Conversely, for a given speedup of 2 and given P_2 and P_3 are feasible, P_1 need not be feasible for the fabric to be non-blocking.

Our preliminary results show that the number of configurations necessary for non-blocking networks decays at a very high rate (more than exponentially fast) with increasing speedup.

To our knowledge, there is no previous literature on this problem. We believe there is motivation to work on blocking fabrics for a number of reasons. First of all, as mentioned before, there are many blocking architectures currently used, and non-blocking architectures may be very complex, whereas, the blocking architectures may be simpler and more feasible for certain applications. Besides, speedup may be supported more easily over a blocking fabric than over a non-blocking fabric due to this simplicity.

We also believe that there exist systematic ways to design very simple fabrics (in terms of the number of configurations supported) which, when put in cascade with others, increase the number of possible end to end configurations geometrically with the number of stages. Such an architecture may enable us to build non-blocking fabrics in a distributed manner that have a significant advantage in terms of simplicity over single stage non-blocking fabrics.

Appendix A

Theory of Majorization: Definitions

A.1 Majorization

For any $\vec{x} = (x_1, \ldots, x_N) \in \Re^N$, let

$$x_{[1]} \geqslant \cdots \geqslant x_{[N]}$$

denote the components of $\vec{x}$ in decreasing order, and let

$$\vec{x}_{\downarrow} = \left(x_{[1]}, \ldots, x_{[N]}\right)$$

Definition A.1 *For $\vec{x}, \vec{y} \in \Re^N$, $\vec{x} \prec \vec{y}$ if the following two conditions hold:*

$$\sum_{i=1}^{k} x_{[i]} \leqslant \sum_{i=1}^{k} y_{[i]}, \ k = 1, \ldots, N-1 \tag{A.1}$$

$$\sum_{i=1}^{N} x_{[i]} = \sum_{i=1}^{N} y_{[i]} \tag{A.2}$$

When $\vec{x} \prec \vec{y}$, $\vec{x}$ is said to be *majorized by* $\vec{y}$ (or $\vec{x}$ majorizes $\vec{y}$). This terminology was introduced by Hardy, Littlewood and Polya. The following is a trivial example of majorization.

$$\begin{aligned}\left(\frac{1}{N}, \ldots, \frac{1}{N}\right) &\prec \left(\frac{1}{N-1}, \ldots, \frac{1}{N-1}, 0\right) \prec \cdots \\ &\prec \left(\frac{1}{2}, \frac{1}{2}, 0, \ldots, 0\right) \prec (1, 0, \ldots, 0)\end{aligned}$$

For a complete treatment of the theory of majorization, see [25].

A.2 Order Symmetry

Vectors $\vec{x}$ and $\vec{y}$ are said to be similarly ordered if there is a permutation π such that $x_{[i]} = x_{\pi(i)}$ and $y_{[i]} = y_{\pi(i)},\ i \in \{1, \ldots, N\}$. Equivalently, $\vec{x}$ and $\vec{y}$ are similarly ordered if $(x_i - x_j)(y_i - y_j) \geqslant 0$ for all i, j.

A.3 Alternate Definition for Majorization

For $\vec{x}, \vec{y} \in \Re^N,\ \vec{x} \prec \vec{y}$ if Conditions (A.1) and (A.2) hold simultaneously. Suppose

$$\sum_{i=1}^{N} x_{[i]} = \sum_{i=1}^{N} y_{[i]} = S$$

Subtracting both sides of (A.1), we get

$$\sum_{i=k+1}^{S} x_{[i]} \geqslant \sum_{i=k+1}^{S} y_{[i]},\ k = 1, \ldots, N-1 \tag{A.3}$$

which is equivalent to (A.1). Hence, if $\vec{x}$ and $\vec{y}$ both have non-negative entries, there are at least as many 0s in $\vec{y}$ as in $\vec{x}$.

Appendix B

Theorems on Majorization

B.1 Kemperman's Theorem

Theorem B.1 ([30]) *Let $\vec{x} \in \Re^N$. Suppose that $m \leqslant x_i \leqslant M,\ i \in \{1, \ldots, N\}$. Then there exists a unique $m \leqslant \theta < M$ and a unique integer, $k \in \{1, \ldots, N\}$ such that*

$$\sum x_i = Mk + \theta + m(N - k - 1)$$

With k and θ so determined,

$$\vec{x} \prec [\underbrace{M \cdots M}_{k}\ \theta\ \underbrace{m \cdots m}_{N-k-1}]^T \equiv \vec{x}^{\max}$$

For example, if $0 \leqslant x_i \leqslant s,\ i \in \{1, \ldots, N\}$ and

$$\sum x_i = sk$$

then $\theta = 0$ and

$$\vec{x}^{\max} = [\underbrace{s \cdots s}_{k}\ \underbrace{0 \cdots 0}_{N-k}]^T$$

B.2 Day's Theorem

Lemma B.1 ([25]) *If $\vec{a} \prec \vec{b},\ \vec{u} \prec \vec{v}$ and if $\vec{b}$ and $\vec{v}$ are similarly ordered (see Appendix A.2), then $\vec{a} + \vec{u} \prec \vec{b} + \vec{v}$*

Day's theorem is the generalization of the lemma to the case of sum of m vectors. Let $\{\vec{v}^l\}_{l=1}^m$ be m real N-dimensional vectors and v_i^l be the i^{th} entry of l^{th} vector. Define partial sum sequences:

$$\kappa_i = \sum_{l=1}^{m} v_i^l, \quad \sigma_i = \sum_{l=1}^{m} v_{[i]}^l$$

Theorem B.2 ([29]) *$\vec{\kappa} \prec \vec{\sigma}$, i.e.,*

$$\sum_{l=1}^{m} \vec{v}^l \prec \sum_{l=1}^{m} \vec{v}_{\downarrow}^l$$

Since the set of vectors $\vec{v}_{\downarrow}^l$ are ordered symmetric, Day's theorem is nothing more than a generalization of the above lemma. For a detailed treatment of rearrangements and majorization, see [25].

B.3 Fulkerson and Ryser's Lemma

Theorem B.3 ([31]) *Let $x_1 \geqslant \cdots \geqslant x_N$ and $y_1 \geqslant \cdots \geqslant y_N$ be integers. If $\vec{x} \prec \vec{y}$ and $i \leqslant j$, then*

$$\vec{x} - \vec{u_i} \prec \vec{y} - \vec{u_j}$$

where $\vec{u_i}$ is the unit vector whose ith entry is 1 and all the others are 0.

Appendix C

Rate Quantization Algorithm with Random Processing Order

Algorithm:

Step 1: Given any s, we know from Lemma 2.1 that there exists a σ_{ij}, $0 < \sigma_{ij} \leqslant s$ such that $R_{ij} + \sigma_{ij}$ is an integer multiple of s for all $1 \leqslant i, j \leqslant N$. Let σ be the matrix whose entries are σ_{ij}s. Define $\tilde{R} = R + \sigma$. All rows and columns of $\tilde{R}$ sum to integer multiples of s. By definition, 1 is also an integer multiple of s, and thus, as illustrated in Fig. 2-5, we can represent the sum of the entries of the ith row and the jth column as $1 + k_i s$ and $1 + k'_j s$ respectively where k_i and k'_j are positive integers.

Step 2: In the second step, the algorithm scans $\tilde{R}$ row by row, starting from the first row, and determines whether the entry will remain unchanged or reduced by s before it is copied as the corresponding entry of the output matrix, R'. Each row is scanned starting from the entry with the largest column sum and continuing with entries of decreasing column sums. If both k_i and k'_j are positive for the current (i, j), that entry is reduced by s and otherwise it is copied directly as the corresponding entry of R'. A step by step description of the second part of the algorithm is as follows:

Initial Values: Let $i = 1$ and $R' = \tilde{R}$; let k_m and k'_n be such that, $1 + sk_m$ and $1 + sk'_n$ are the mth row and nth column sum respectively, as illustrated in Fig. (2-5).

Repeat (1)-(2) until $i = N + 1$.

1. Set $E = \{1, \ldots, N\}$. Repeat (a)-(c) until $k_i = 0$.

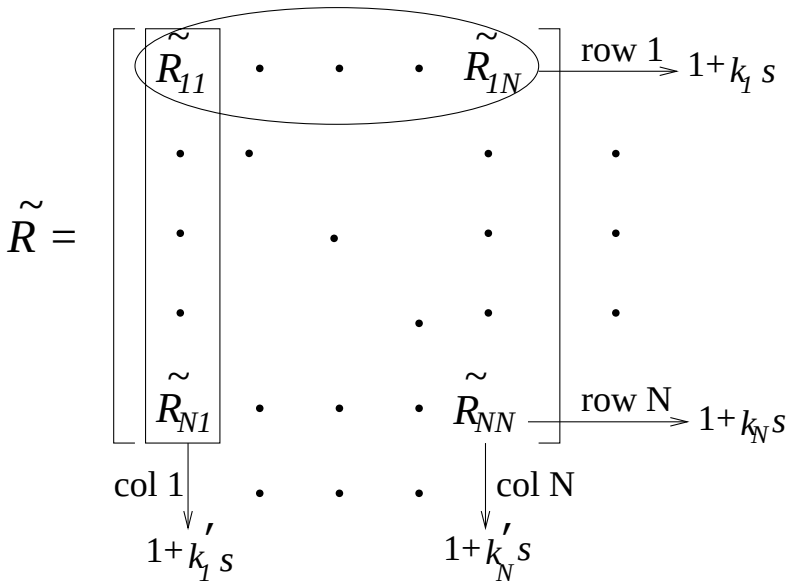

Figure C-1: The $N \times N$ matrix $\tilde{R}$ is illustrated. Each row i and and column j sum to $1 + k_i s$ and $1 + k_j' s$ respectively where k_i snd k_j' are non-negative integers.

(a) $j = \arg\max_{j \in E} k_j'$

(b) $R_{ij}' = \tilde{R}_{ij} - s,\ k_i \to k_i - 1,\ k_j' \to k_j' - 1,\ E \to E - \{j\}.$

2. $i \to i + 1$

The described algorithm reduces the elements of $\tilde{R}$ with lower row indices earlier. One might also randomize the procedure and work on a row randomly picked at every iteration.

Lemma C.1 *The algorithm successfully terminates with a matrix, R' which is doubly stochastic.*

Before we give the proof of the lemma, let us introduce some notation. Let

$$\begin{aligned} k_i &= \frac{1}{s}\left(\sum_{j=1}^{N}(R_{ij}') - 1\right) \\ k_j' &= \frac{1}{s}\left(\sum_{i=1}^{N}(R_{ij}') - 1\right) \end{aligned}$$

In other words, k_i and k_j' are initially reduced as stated in the algorithm and then normalized as R' is produced. Succinctly, we can represent k_i and k_j' as an entry of the vectors $\vec{k}$ and $\vec{k'}$ respectively. Let $n_i',\ i \geqslant 1$ be the number of columns j, for which $k_j' \geqslant i$. For example, if $\vec{k'} = [2\ 1\ 1]^T$, then $\vec{n'} = [3\ 1\ 0]^T$ as illustrated in Fig. C-2.

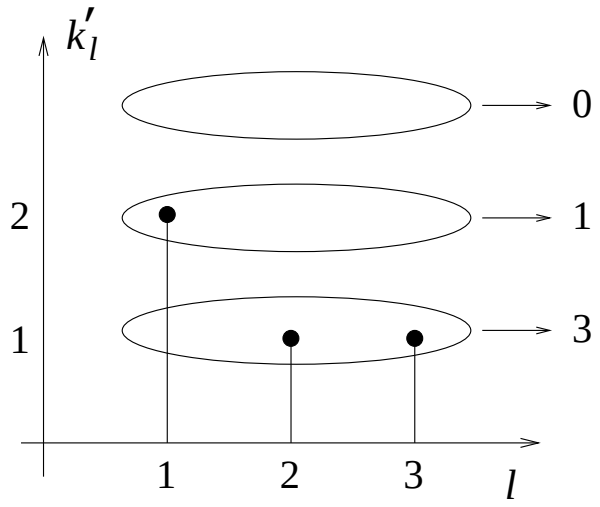

Figure C-2: Vector $\vec{k'} = [2\ 1\ 1]^T$, thus $\vec{n'} = [3\ 1\ 0]^T$.

Proof: By induction. We shall first show that initially

$$n'_1 \geqslant k_i \tag{C.1}$$

for all i, $1 \leqslant i \leqslant N$, Thus, for any $\vec{k}$ and any i selected to be the first to be the first row to be processed, the algorithm will always be able to find sufficient entries to reduce (by s) to make the row sum equal to 1. We will prove a more general version of (C.1):

$$\vec{k} \prec \vec{n'} \tag{C.2}$$

namely, the vector $\vec{k}$ is majorized by the vector $\vec{n'}$. The definition and some examples about majorization were given in Appendix A.1.

First we prove that (C.2) holds at the beginning of the algorithm. Recall that $\sigma = \tilde{R} - R$. Hence,

$$\frac{1}{s}\sigma_{ij} \leqslant 1$$

$\forall i, j$. Let the lth column vector of σ be $\vec{v}_l$ and thus $v_{l,j} = \sigma_{jl}$ and $\langle \vec{v}_l, \vec{e} \rangle = k'_l s$, where $\vec{e} = [1 \cdots 1]^T$. From Kemperman's theorem (Appendix B.1), $\vec{v}_l$ is majorized by any vector for which k'_l entries are

s, and the other $N - k'_l$ entries are 0. Hence,

$$\vec{v}_l \prec [\underbrace{s \cdots s}_{k'_l} \; \underbrace{0 \cdots 0}_{N-k'_l}]^T \equiv \vec{v}_l^{\max} \tag{C.3}$$

Thus, the vector on the right side of (C.3) is the *maximal vector* (in the sense of majorization) of the set of vectors whose entries are between 0 and s and $\langle \vec{v}, \vec{e} \rangle = k'_l s$. Let us denote the maximal vector of the lth column vector by $\vec{v}_l^{\max}$.

Now, let us define a new matrix, $\frac{1}{s}[\vec{v}_1^{\max} \cdots \vec{v}_N^{\max}]$, where each column is the maximal vector of the corresponding column of $\frac{1}{s}\sigma$. Note that the vector of column sums for this new matrix is $\vec{k'}$, and thus the corresponding distribution will be $\vec{n'}$; however, the row sums are not $\vec{k}$. Let the vector of row sums for our matrix be $\vec{k}_{\text{new}}$. Thus, $k_{\text{new},1}$ is the number of columns with $k'_j \geqslant 1$, i.e., n'_1; $k_{\text{new},2}$ is the number of columns with $k'_j \geqslant 2$, i.e., n'_2, and so on. More precisely, $k_{new,i}$ is the number of columns j, for which $k'_j \geqslant i$. Thus,

$$k_{new,i} = n'_i \tag{C.4}$$

But the vectors, $\vec{v}_l^{\max}$, $l \in \{1, \ldots, N\}$ are order symmetric (see Appendix A.2 for the definition). Hence we get the desired result using Day's theorem (Appendix B.2):

$$\begin{aligned} \vec{k}_{new} &= \vec{n}' = \frac{1}{s}\sum_{l=1}^{N} \vec{v}_l^{\max} && \text{(C.5)} \\ &\succ \frac{1}{s}\sum_{l=1}^{N} \vec{v}_l && \text{(C.6)} \\ &= \vec{k} && \text{(C.7)} \end{aligned}$$

We just showed that at the beginning of the algorithm, $\vec{n'} \succ \vec{k}$, and thus, $n_1 \geqslant k_i$, for all $i \leqslant N$. That is, the first step of the algorithm can be executed successfully to make the first row sum to 1. The partial sums[1] of the two sequences are illustrated in Fig. C-3. Such curves are called Lorentz curves and if, for two vectors, $\vec{v}^I \prec \vec{v}^{II}$, then the partial sum curve for $\vec{v}^{II}$ will always be above that of $\vec{v}^I$.

Next, we will prove that a similar majorization relation holds at the beginning of every step of

[1]The mth partial sum of a vector, $\vec{v}$, is defined to be $\sum_{j=1}^{m} v_j$. Recall that $\vec{v}^I \prec \vec{v}^{II}$ if every partial sum of $\vec{v}^{II}$ is at least as great as that of $\vec{v}^I$

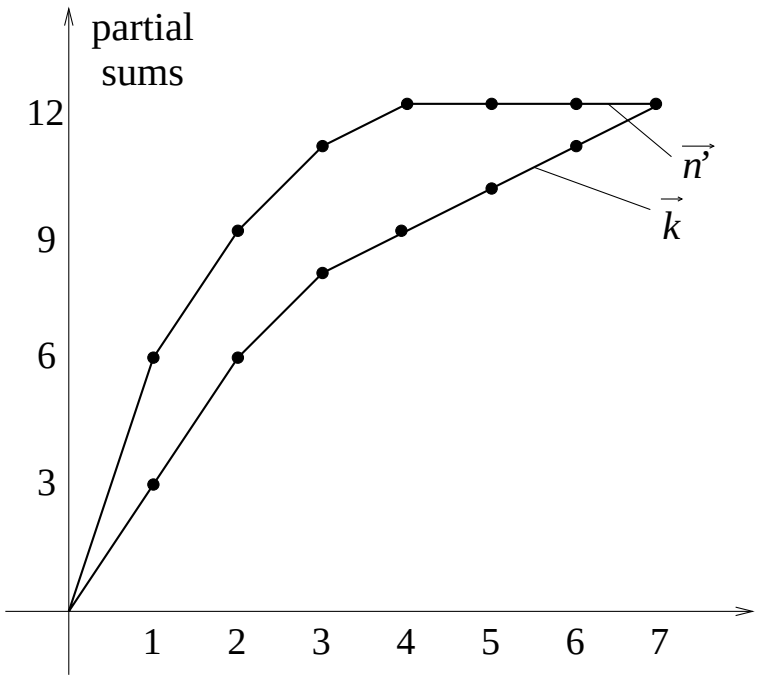

Figure C-3: Sample Lorentz curves for $\vec{n'}$ and $\vec{k}$ are illustrated. Since $\vec{n'} \succ \vec{k}$ initially, the partial sum curve of $\vec{n'}$, is above that of $\vec{k}$.

the algorithm. We will use induction as follows. We have shown that $\vec{k} \prec \vec{n}'$ at the beginning of the first step. We now assume that it holds at the beginning of the ith step, $1 \leqslant i \leqslant N-1$ and show that it still holds at the end of the ith step. As a byproduct, we also show that the algorithm can successfully complete each step.

Suppose, the algorithm successfully constructed the first i rows of R'. We will show that (C.2) still holds at the beginning of the $i+1$st step, and the corresponding row of R' can be formed successfully.

First, let us focus on the two vectors, $\vec{n'}$ and $\vec{k}$ at the beginning of step i. At this point, $k_1, \ldots, k_{i-1} = 0$. The sum of the entries of the row that is currently being processed is k_i. By the induction hypothesis, we assume $\vec{k} \prec \vec{n'}$; therefore, there should be as many 0s in vector $\vec{n'}$ as there are in k (verified in Appendix A.3). Since there are at least $i-1$ 0s in $\vec{k}$, we have $n'_{N-i+2}, \ldots, n'_N = 0$. The entries of $\vec{n'}$ and $\vec{k}_{\downarrow}$ (defined in Appendix A.1 as the decreasing rearrangement of the entries of $\vec{k}$) can be listed as follows:

$$\begin{array}{ccccccc} n'_1 & \cdots & n'_{r-1} & n'_r & n'_{r+1} & \cdots & \overbrace{0 \cdots 0}^{i-1} \\ k_{C(1)} & \cdots & k_{C(r-1)} & k_i & k_{C(r+1)} & \cdots & \underbrace{0 \cdots 0}_{i-1} \end{array}$$

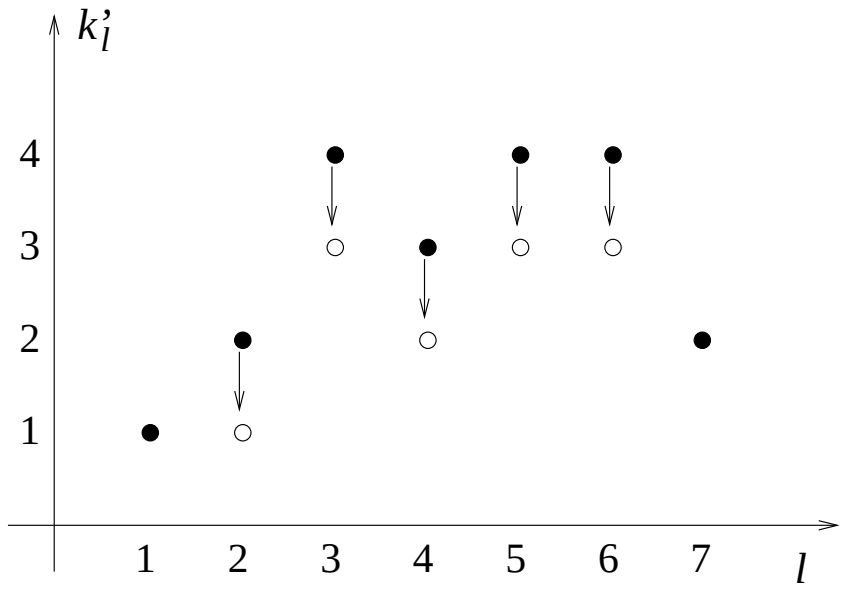

Figure C-4: If $k_i = 5$, the entries of the ith row that are decreased are illustrated above.

where $C(q)$ is the qth entry in decreasing order from the largest in $\vec{k}$. Let the current position of k_i in this ordered list be r as shown above, i.e., $i = C(r)$. Since $\vec{k} \prec \vec{n'}$, there exists at least one entry in $\vec{n'}$ which is greater than or equal to k_i. Let the smallest such entry be $n'_{r-r'}$.

Lemma C.2 *At the end of ith step, the only change in* $\vec{n'}$ *is that the entries* $n'_{r-r'}$ *and* $n'_{r-r'+1}$ *will be replaced with* $\left[n'_{r-r'+1} + \left(n'_{r-r'} - k_i\right)\right]$ *and a 0.*

Proof: These two changes can be explained as follows. The algorithm will look into the current R' for the column with an entry which has not yet been reduced in step i and which has the maximum column sum, and reduce it by s. Suppose this maximum column sum is ms for some $m \in \mathbb{Z}^+$. This operation will reduce the number of columns, i, such that $k_i = m$ by 1. Thus, the only change in $\vec{n'}$ will be in the smallest non-zero entry, n'_m, which will decrease by 1. If that entry is greater than 1, then there were multiple entries with the maximum column sum. The algorithm continues with these other entries. Hence, if the original value of k_i is greater than n'_m, then after processing n'_m entries, n'_m will become 0 and $k_i - n'_m$ entries will be left at the ith row to be decreased. The algorithm will go on with the entries that have not been reduced before and with highest possible column sums. At this stage, the new value, $\hat{n}'_m$, of n'_m is 0 and the new value, $\hat{k}_i$ of k_i is $k_i - n'_m$. Note that n'_m potential entries have already been processed, and if k_i is greater than the second largest entry, n'_{m-1}, of $\vec{n'}$ then n'_{m-1} will be reduced to $\hat{n}'_{m-1} = n'_m$ but no further beyond that, since n'_m potential entries have already been processed. Similarly, each entry of $\vec{n}'$, which is smaller

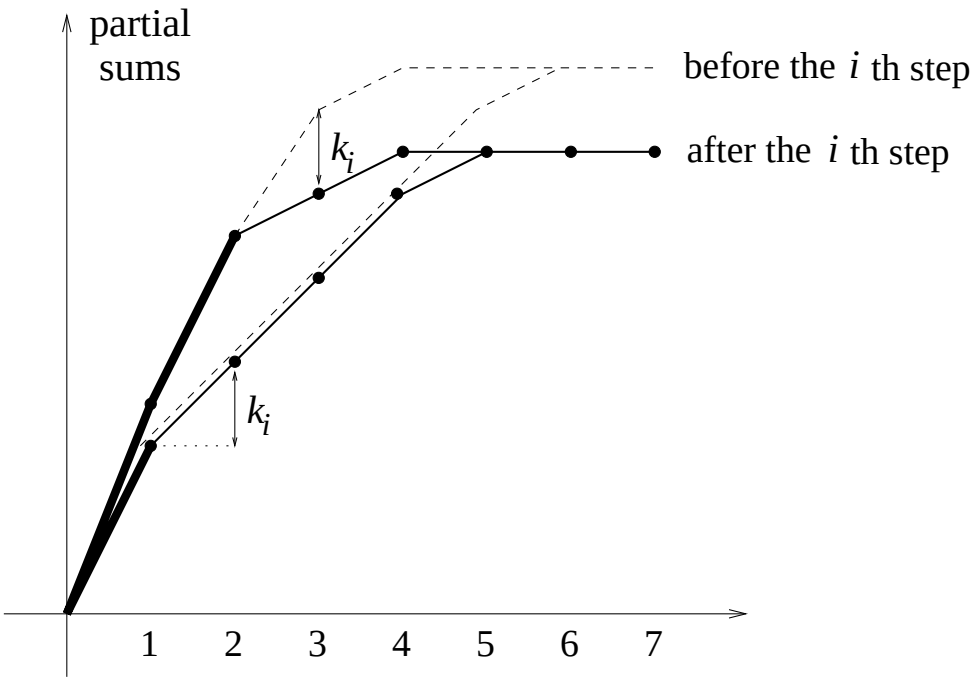

Figure C-5: In the ith step, k_i is removed (replaced with a 0) and the smallest entry of $\vec{n}'$ greater than or equal to k_i is reduced by $k_i - n'_{r-r'+1}$, and the following entry, $n'_{r-r'+1}$ is removed (replaced with a 0). The dashed curve is the initial curve, and the solid one is the one at the end of the ith step. The bold segments are the ones which do not change. The distance between the two curves does not decrease at all.

than k_i will be replaced with the next entry in order. Finally, the first entry, $n'_{r-r'}$, in $\vec{n'}$ that is greater than k_i will be reduced by only $n'_{r-r'} - k_i$. Hence, after the ith row is processed, $\vec{n'}$ will have a 0 replacing $n'_{r-r'}$, and a $\left[n'_{r-r'+1} + \left(n'_{r-r'} - k_i\right)\right]$ replacing $n'_{r-r'+1}$. Note that at the end of the ith step, $\vec{k}_{\downarrow}$ will be the same except k_i will be replaced with a 0.

This process is illustrated in Fig. C-4 assuming $\vec{k'} = [1\ 2\ 4\ 3\ 4\ 4\ 2]$, i.e., $\vec{n'} = [7\ 6\ 4\ 3\ 0\ 0\ 0]$ at the beginning of step i. If $k_i = 5$, then at the end of step i, $\vec{n'} = [7\ 5\ 3\ 0\ 0\ 0\ 0]$. Notice that 6 is the smallest entry in $\vec{n'}$ greater than or equal to $k_i = 5$. Hence, 6 and 4 are changed to $6 + (4 - 5) = 5$ and 0 respectively.

We will consider the two possible scenarios, $r' < 0$ or $r' \geqslant 0$ and show that in either case, $\vec{k} \prec \vec{n}'$ at the end of the ith step of the algorithm. But before that we present a graphical illustration of what follows. Suppose the algorithm has just processed the ith row. The Lorentz curves of $\vec{k}$ and $\vec{n}'$ are illustrated in Figures C-5 and C-6. The entry, k_i will be removed from $\vec{k}$. The new Lorentz curve for $\vec{k}$ can be sketched from the old one by just removing the appropriate segment of the curve and attaching the two separate parts together as illustrated in these figures. The new Lorentz

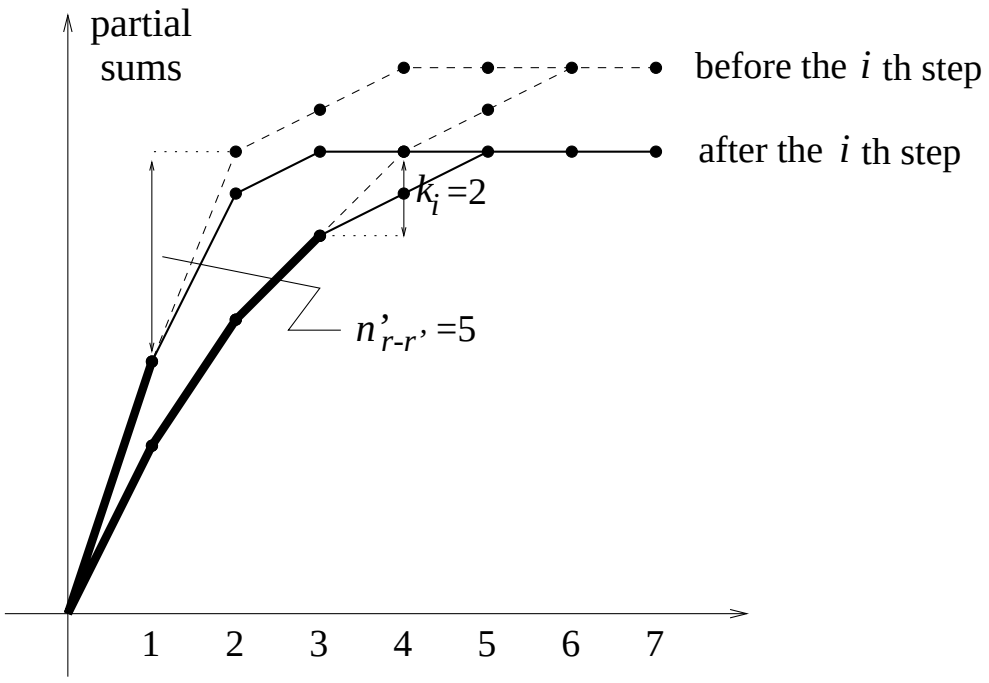

Figure C-6: In the ith step, k_i is removed (replaced with a 0) and the smallest entry of $\vec{n}'$ greater than or equal to k_i is reduced by $k_i - n'_{r-r'+1}$, and the following entry, $n'_{r-r'+1}$ is removed (replaced with a 0).The dashed curve is the initial curve, and the solid one is the one at the end of the ith step. The bold segments are the ones which do not change. The distance between the two curves decreases, but the curve for $\vec{n}'$ is still above that of $\vec{k}$.

curve for $\vec{n}$ can similarly be sketched with some modification to the old one. The algorithm will find the segment with the smallest increment greater than k_i. Then, it will reduce this increment by $k_i - n'_{r-r'}$, remove $n'_{r-r'}$, and attach the two separate parts. There are two alternatives according to the relative places of these changed segments. The erased segment of $\vec{k}$ may be to the left of the modified segments of $\vec{n}'$ (i.e., $r' < 0$) as shown in Fig. C-5, or it may be to the right ($r' \geqslant 0$) as shown in Fig. C-6.

In both scenarios, the two Lorentz curves intersect at 0 and at $\sum_l n'_l = \sum_l k_l$. Initially, these are the only two points they intersect, and the curve for $\vec{n}'$ is always above the curve for $\vec{k}$, otherwise. We need to show that this is the case after the ith step. In the first scenario, this can be easily observed from Fig. C-5. Since the removed segment in $\vec{k}$ is to the left of the reduced segment of $\vec{n}'$, the distance between the two curves will only increase in between these segments, and remain the same outside this region. The second scenario is more complicated. The decrease in the curve of $\vec{n}'$ occurs earlier than that of $\vec{k}$. Thus, the distance between the two curves decrease in between these two segments. This scenario requires a more careful treatment. We focus on these two scenarios in the following two items to prove the desired result.

1. If $r' < 0$, the statement is rather straightforward. It is nothing more than an application of Fulkerson and Ryser's Lemma (Appendix B.3). However, to develop intuition for the second case, we will elaborate a little more on this scenario. There are three regions we need to consider as shown in the following table:

n'_1	$\cdots$	n'_{r-1}	n'_r	n'_{r+1}	$\cdots$	$n'_{r-r'}$	$n'_{r-r'+1}$	$\cdots$	$\overbrace{0\cdots0}^{i-1}$
$k_{C(1)}$	$\cdots$	$k_{C(r-1)}$	k_i	$k_{C(r+1)}$	$\cdots$	$k_{C(r-r')}$	$k_{C(r-r'+1)}$	$\cdots$	$\underbrace{0\cdots0}_{i-1}$
I			**II**				**III**		

   At the end of the ith step, the partial sums of the two sequences are as follows. In region I, no entry is changed in either one of the vectors, hence we are OK. In region II, at the end of the ith step, k_i will be replaced with a 0 and it will no longer be in the second region. All the entries of $\vec{n}'$ will be unchanged up to $n'_{r-r'}$. Thus, the partial sums will change in favor of $\vec{n'}$ by an extra k_i from n'_r all the way down to n'_{r-r}. This entry is replaced with $\left[n'_{r-r'+1} + \left(n'_{r-r'} - k_i\right)\right]$, and the next entry, $n'_{r-r'+1}$, will be replaced with a 0 and removed from the second region. The total decrease in the partial sums of $\vec{n'}$ in this region is k_i. The

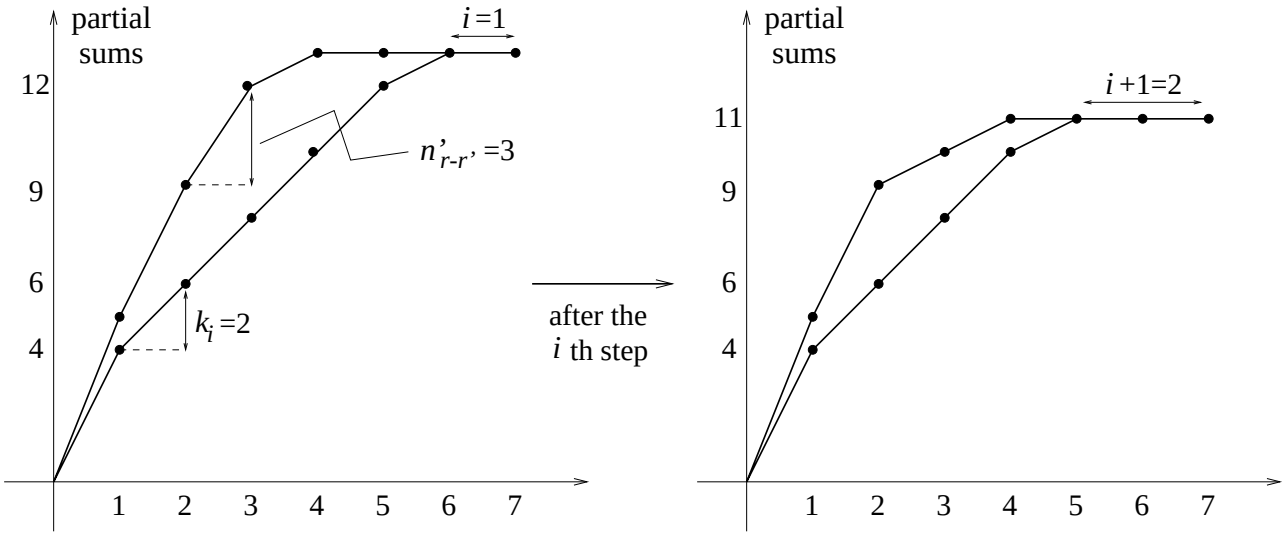

Figure C-7: Sample Lorentz curves for $\vec{n'}$ and $\vec{k}$ before and after the ith step are illustrated for the first case. The smallest entry in $\vec{n'}$ which is greater than or equal to k_i is 3. In this example $r' = -1$, i.e., the entry of $\vec{n'}$ being processed is just on the right of k_i.

extra k_i gained in favor of $\vec{n'}$ earlier by the removal of k_i from vector $\vec{k}$ is good enough to make up for this loss of $\vec{n}'$. The third region of both $\vec{n'}$ and $\vec{k}$ are expanded similarly, with the addition of a 0, which will not affect the partial sums, and hence the majorization is preserved. The initial and final Lorentz curves for this case are illustrated in Fig. C-7.

2. If $r' \geqslant 0$ we have to be more careful. Again, there are still three regions we need to consider:

$$\begin{array}{ccc|cccc|ccc} n'_1 & \cdots & n'_{r-r'-1} & n'_{r-r'} & n'_{r-r'+1} & \cdots & n'_r & n'_{r+1} & \cdots & \overbrace{0\cdots 0}^{i-1} \\ k_{C(1)} & \cdots & k_{C(r-r'-1)} & k_{C(r-r')} & k_{C(r-r'+1)} & \cdots & k_i & k_{C(r+1)} & \cdots & \underbrace{0\cdots 0}_{i-1} \end{array}$$

I **II** **III**

Before the i^{th} step, let

$$\sum_{j=1}^{r-r'-1} \left(n'_j - k_{C(j)}\right) \stackrel{\text{def}}{=} d \geqslant 0$$

Thus,

$$\sum_{j=r-r'}^{N} \left(k_{C(j)} - n'_j\right) = d$$

Since, $\sum_{j=r+1}^{N} k_{C(j)} \geqslant \sum_{j=r+1}^{N} n'_j$,

$$\sum_{j=r-r'}^{r} \left(k_{C(j)} - n'_j\right) \leqslant d \tag{C.8}$$

At the end of the ith step, the partial sums are unchanged in the first region. The first element, $n'_{r-r'}$, of $\vec{n}'$ in the second region will be replaced with $n'_{r-r'} - k_i + n'_{r-r'+1}$. Since $n'_{r-r'}$ is the smallest entry of $\vec{n'}$ which is greater than or equal to k_i,

$$k_{C(r-r')} \geqslant k_{C(r-r'+1)} \geqslant \cdots \geqslant k_i \geqslant n'_{r-r'+1} \geqslant \cdots \geqslant n'_r \tag{C.9}$$

Combining (C.9) with (C.8), we get the following set of inequalities. First, (C.8) can be rewritten as:

$$\left(n'_{r-r'} - k_i + n'_{r-r'+1}\right) + n'_{r-r'+2} + \cdots + n'_r + d \geqslant k_{C(r-r')} + \cdots + k_{C(r-1)} \tag{C.10}$$

Combining (C.10) with (C.9),

$$\begin{aligned}
\left(n'_{r-r'} - k_i + n'_{r-r'+1}\right) + n'_{r-r'+2} + \cdots + n'_{r-1} + d &\geqslant k_{C(r-r')} + \cdots + k_{C(r-2)} && \text{(C.11)}\\
&\vdots \\
n'_{r-r'} - k_i + n'_{r-r'+1} + n'_{r-r'+2} + d &\geqslant k_{C(r-r')} + k_{C(r-r'+1)} && \text{(C.12)}\\
n'_{r-r'} - k_i + n'_{r-r'+1} + d &\geqslant k_{C(r-r')} && \text{(C.13)}
\end{aligned}$$

Replacing $d = \sum_{j=1}^{r-r'-1} \left(n'_j - k_{C(j)}\right)$, we get the desired partial sum inequalities for region II. The third region of both $\vec{n'}$ and $\vec{k}$ are expanded similarly, with the addition of a 0, which will not affect the partial sums, and hence the majorization. The initial and final Lorentz curves for this case are illustrated in Fig. C-8.

Thus, we proved that at the beginning of each step, (C.2) holds and $n'_1 \geqslant k_i$, for all $i \leqslant N$. Therefore, no matter which row is being processed, the algorithm will always be able to find the desired number of entries to reduce, and at the end of the algorithm, $k_i = 0$, for all $i \leqslant N$. But,

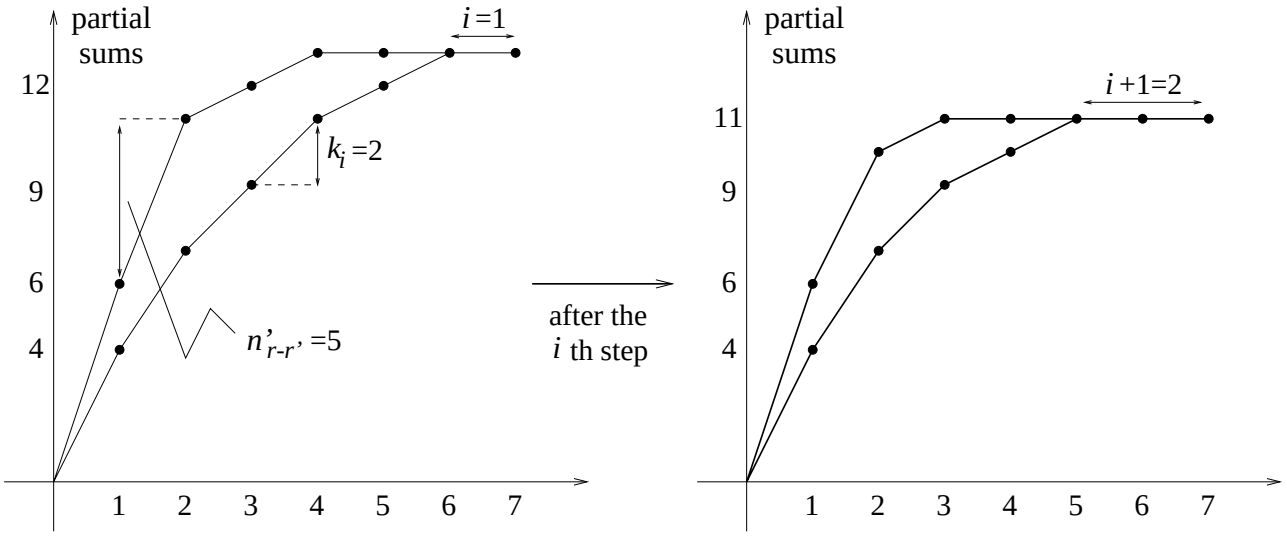

Figure C-8: Sample Lorentz curves for $\vec{n'}$ and $\vec{k}$ before and after the ith step are illustrated for the second case. The smallest entry in $\vec{n'}$ which is greater than or equal to k_i is 5. In this example $r' = 2$, i.e., the entry of $\vec{n'}$ being processed is two entries to the left of k_i.

since

$$0 = \sum_{i=1}^{N} k_i = \sum_{j=1}^{N} k'_j \tag{C.14}$$

and $k'_j \geqslant 0$, for all $j \leqslant N$, it is also true that $k'_j = 0$, for all $j \leqslant N$ completing the proof.

Bibliography

[1] DWDM, SONET, and Photonics: "The Emerging All-Optical Network 2001-2006," in *Insight Research*, 2001/05/02.

[2] Keshav S. and Sharma R., "Issues and Trends in Router Design," in *IEEE Communications Magazine*, vol. 36, No. 5, pp. 144-151, May 1998.

[3] Awdeh R. Y. and Mouftah H. T., "Survey of ATM Switch Architectures," in *Computer Networks and ISDN Systems*, vol. 27, pp. 1567-1613, 1995.

[4] Turner J. and Yamanaka N., "Architectural Choices in Large Scale ATM Switches," in *Technical Report, Department of Computer Science, Washington University*, WUCS 97-21, 1997.

[5] Kam A. and Siu K. Y., "Linear Complexity Algorithms for QoS Support in Input-Queued Switches with No Speedup", in *IEEE Journal on Selected Areas of Communications*, vol. 17, No. 6, p. 1040-1056, June 1999.

[6] McKeown N., Anantharam V. and Walrand J., "Achieving 100% Throughput in an Input Queued Switch", in *Proceedings of INFOCOM 96*, p. 296-302, 1996.

[7] McKeown N., Mekittikul A., Anantharam V. and Walrand J., "Achieving 100% Throughput in an Input Queued Switch", in *IEEE Transactions on Communications*, vol. 47, No. 8, pp. 1260-1267, August 1999.

[8] McKeown N., "The iSLIP Scheduling Algorithm for Input Queued Switches", in *IEEE/ACM Transactions on Networking*, vol. 7, No. 2, p. 188-201, April 1999.

[9] Partridge C., et.al., "A 50 Gb/s Router", in *IEEE/ACM Transactions on Networking*, vol. 6, No. 3, p. 237-247, June 1998.

[10] Karol M., Hluchyi M. and Morgan S., "Input Versus Output Queueing on a Space Division Switch", in *IEEE Transactions on Communications*, vol. 35, 1987.

[11] Oie Y., Murata M., Kubota K. and Miyahara H., "Effect of Speedup in Non-Blocking Packet Switch", in *Proc. ICC '89, Boston, MA*, p. 410-414.

[12] Anderson T. E., et.al., "High Speed Switch Scheduling for Local Area Networks", in *ACM Transactions on Computer Systems*, vol. 11, No. 4, pp. 319-352, November 1993.

[13] Stiliadis D. and Varma A., "Providing Bandwidth Guarantees in an Input-buffered Crossbar Switch", in *Proceedings of INFOCOM 1995*, vol.3, pp. 960-968.

[14] Weller T. and Hajek B., "Scheduling Nonuniform Traffic in a Packet Switching System with Small Propagation Delay", in *IEEE/ACM Transactions on Networking*, vol. 5, No. 6, pp. 813-823, December 1997.

[15] Chang C. S., Chen W. J. and Huang H. Y., "On Service Guarantees for Input Buffered Crossbar Switches: A Capacity Decomposition Approach by Birkhoff and von Neumann", in *IEEE IWQoS '99*, pp. 79-86, London, U.K., 1999.

[16] Chang C. S., Chen W. J. and Huang H. Y., "Birkhoff-von Neumann Input Buffered Crossbar Switches", in *Proceedings of INFOCOM 2000.*

[17] Hung A., Kesidis G., McKeown N., "ATM Input Buffered Switches with Guaranteed Rate Property", in *Proceedings of IEEE ISCC '98*, pp. 331-335, Athens, 1998.

[18] Chuang S. T., Goel A., McKeown N. and Prabhakar B., "Matching Output Queueing with a Combined Input/Output-Queued Switch", in *IEEE Journal on Selected Areas of Communications*, vol. 17, No. 6, p. 1030-1039, June 1999.

[19] Krishna P., Patel N., Charny A. and Simcoe J., "On the Speedup Required for Work-Conserving Crossbar Switches", in *IEEE Journal on Selected Areas of Communications*, vol. 17, No. 6, p. 1057-1066, June 1999.

[20] Prabhakar B. and McKeown N., "On the Speedup Required for Combined Input and Output Switch", in *Proceedings of INFOCOM 1999.*

[21] Charny A., Krishna P., Patel N. and Simcoe R., "Algorithms for Providing Bandwidth and Delay Guarantees in Input Buffered Crossbars with Speedup", in *Proceedings of INFOCOM 1998.*

[22] Charny A., "Providing QoS Guarantees in Input Buffered Crossbar Switches with Speedup", in *PhD Dissertation*, MIT, Cambridge, MA, 1998.

[23] Bennett J., Zhang H., "WF^2Q: Worst Case Fair Weighted Fair Queueing Algorithms", in *Proceedings of INFOCOM 1996.*

[24] von Neumann J., "A certain zero-sum two-person game equivalent to the optimal assignment problem," in *Contributions to the Theory of Games*, vol. 2, pp.5-12, Princeton University Press, Princeton, New Jersey, 1953.

[25] Marshall A. W. and Olkin I., "Inequalities: Theory of Majorization and Its Applications," Academic Press, New York, NY, 1979.

[26] Bertsekas D. P., "Nonlinear Programming," Athena Scientific, Belmont, MA, 1995.

[27] Parekh A. K. and Gallager R. G., "A Generalized Processor Sharing Approach to Flow Control in Integrated Service Networks: the Single Node Case", in *IEEE/ACM Transactions on Networking*, vol. 1, pp. 344-357, 1993.

[28] Hall M., "Combinatorial Theory," Blaisdell Publishing Company, 1967.

[29] Day P. W., "Rearrangement Inequalities", in *Canad. J. Math.*, vol. 24, pp. 930-943, 1972.

[30] Kemperman J. H. B., "Moment Problems for Sampling Without Replacement, I, II, III", in *Nederl. Akad. Wetensch. Proc. Ser.*, vol. 76, pp. 149-188, 1973.

[31] Fulkerson D. R. and Ryser H. J., "Multiplicities and Minimal Widths for (0-1) Matrices," in *Canad. J. Math.*, vol. 14, pp. 11-17, 1962.

[32] Paxson V., "Growth Trends in wide-area TCP connections," in *IEEE Network*, vol. 8, pp. 8-17, Jul.-Aug. 1994.

[33] Eriksson H., "MBone: The Multicast Backbone," in *Commun. ACM*, vol. 37, pp. 54-60, Aug. 1994.

[34] Prabhakar B., McKeown N. and Ahuja R., "Multicast Scheduling for Input-Queued Switches", in *IEEE Journal on Selected Areas of Communications*, vol. 15, no. 5, June 1997.

[35] Iyer S., Awadallah A. and McKeown N., "Analysis of a Packet Switch with Memories Running Slower Than the Line Rate", in *Proceedings of INFOCOM 2000.*

[36] Stephens J. and Zhang H., "Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture", in *Proceedings of INFOCOM 1998.*

[37] Hui J. Y., "Switching and Traffic Theory for Integrated Broadband Circuits," in *Kluwer Academic Publishers*, Boston, MA, 1990.

[38] Turner J., "An Optimal Non-blocking Multicast Virtual Circuit Switch", in *Technical Report*, Washington University Computer Science Department, WUCS-93-30, 1993.

[39] Pretzel O., "Error Correcting Codes and Finite Fields," in *Clarendon Press*, Oxford, 1992.

[40] Berlekamp E. R., "Algebraic Coding Theory," in *McGraw-Hill*, New York, NY, 1968.

[41] Ramaswami R. and Sivarjan K. N., "Optical Networks," in *Morgan Kaufman*, 1998.

[42] Tse D., Gallager R. and Tsitsiklis J., "Statistical Multiplexing of Multiple Time-Scale Markov Streams", in *IEEE Journal on Selected Areas of Communications*, August 1995.

[43] Feldman A., Gilbert A. C. and Willinger W., "Data Networks as Cascades: Investigating the Multifractal Nature of Internet WAN Traffic", in *Proceedings of the ACM/SIGCOMM 98*, pp. 42-55, 1998.

[44] Qiao C. and Yoo M., "Optical Burst Switching - A New Paradigm for an Optical Internet," in *Journal of High Speed Networks*, Special Issue on Optical Networking, vol. 8, no. 1, pp.69-84, 1999.

[45] Awduche D., et. al., "Multi-Protocol Lambda Switching: Combining MPLS Traffic Engineering Control with Optical Crossconnects," in *IEEE Communication Magazine*, pp. 111-116, March 2001.

[46] Guillemot C., et.al., "Transparent Optical Packet Switching: The European ACTS KEOPS Project Approach," in *IEEE Journal of Lightwave Technology*, vol. 16, no. 12, Dec. 1998.

[47] Kam A., Siu K.Y., Barry R., Swanson E., "A Cell Switching WDM Broadcast LAN with Bandwidth Guarantee and Fair Access," in *IEEE Journal of Lightwave Technology*, vol. 16, p. 2265-2280, 1998.

[48] Chen H., "Fluid Approximations and stability of Multiclass Queueing Networks I," in *Technical Paper, Electrical Engineering Department, Stanford university*, 2000.

[49] Dai J. G., "The Throughput of Data Switches with and without Speedup," in *Annals of Applied Probability*, vol. 5, pp. 637-665, 1995.

[50] Kesidis G, Walrand J. and Chang C. S., "Effective Bandwidths for Multiclass Markov Fluids and Other ATM Sources," in *IEEE Transactions on Networking*,vol. 1, no. 4, pp. 424-428, Aug. 1993.

[51] Gallager R. G., "Discrete Stochastic Processes," in *Kluwer Academic Publishers*, 1995.

[52] Clos C., "A Study of Non-blocking Switching Networks," in *The Bell System Technical Journal*, 32, 406-424, 1953.

[53] Slepian D., "Two Problems on a Particular Crossbar Switching Network," in *Unpublished Manuscript*, 1952.

[54] Paull M. C., "Reswitching of Connection Networks," in *Bell Syst. Tech. J.*, vol. 41, pp. 833-855, 1962.

[55] Yang Y., Masson G. M., "The Necessary Conditions for Clos Type Non-blocking Multicast Networks", in *IEEE Transactions on Computers*, vol. 48, No. 11, p. 1214-1227, November 1999.

[56] Lin G. H., "Nonblocking Routing Properties of Clos Networks," in *Advances in Switching Networks* , Kluwer Academic Publishers, Boston, MA, 2000.

[57] Laywine C. F. and Mullen G. L., "Discrete Mathematics Using Latin Squares," in *Wiley-Interscience Series in Discrete Mathematics and Optimization*, 1998.