PROVIDING PERFORMANCE GUARANTEES WITH CROSSBAR-BASED ROUTERS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Shang-Tse Chuang

December 2004

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as dissertation for the degree of Doctor of Philosophy.

[illegible]

Prof. Nicholas W. McKeown Principal Advisor

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as dissertation for the degree of Doctor of Philosophy.

[illegible]

Prof. Balaji Prabhakar

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as dissertation for the degree of Doctor of Philosophy.

Ashish Goel

Prof. Ashish Goel

Approved for the University Committee on Graduate Studies

Abstract

The backbone of the Internet is comprised of a network of high performance routers. Every router receives packets and forwards them to the correct outgoing port. Routers are designed to achieve a specific capacity, and network operators expect routers to provide predictable performance. This thesis is about how to design practical routers with predictable performance.

Network operators expect certain performance characteristics; for example, if the arrival rate is less than the router's advertised capacity, they can reasonably assume the router can handle the traffic. Somewhat surprisingly, no commercial router can do this today. And some operators would like to provide customers with service guarantees such as guaranteed bandwidth and delay through their network. To provide these guarantees, each individual router within the network must also provide bandwidth and delay guarantees.

Most high performance routers built today use crossbars and a centralized scheduler. Building crossbar-based routers which provide these performance guarantees is difficult using current technology. Scheduling the crossbar is a bottleneck and limits scalability. It also makes it impractical to provide performance guarantees. As line rates and the number of ports increase, the scheduling problem will only become more difficult.

This thesis describes crossbar scheduling algorithms that provide deterministic throughput, bandwidth, and delay guarantees. Each technique is based on an analytical counting method. The first result — for crossbar switches — is primarily a theoretical observation of what is possible, but is too complex to be practical. To make the result more practical, I describe crossbars with a small amount of internal buffering. A buffered crossbar simplifies the scheduling process and provides a practical way to build routers with performance guarantees.

Acknowledgements

I am indebted to many for their advice and assistance throughout my time at Stanford.

First and foremost, I would like to thank my adviser, Nick McKeown, for his continued support, guidance, and encouragement in both academia and industry. Nick always has been able to bring out the best in people and has a great instinct to identify what matters most. I really appreciate all that you have taught me.

I am abundantly grateful to my co-advisor, Balaji Prabhakar, for his encouragement and most of all his advice. Thank you for always being there for me.

I would also like to thank Ashish Goel for the many discussions that led to a number of results in this thesis. I really appreciate the good times we spent together.

Numerous discussions with Sundar Iyer lead to most of the work on the buffered crossbar results. It has been a pleasure working with you.

Many people have greatly helped in the preparation of this thesis. I am especially grateful to Greg Watson and Isaac Keslassy for reviewing and providing feedback on the entire thesis. I would also like to acknowledge the helpful feedback from Midge Eisele, Nandita Dukkipati, Sundar Iyer, and Martin Casado.

My time at Stanford has been split into two fantastic periods. From the earlier period, I would like to thank the following group members for all the good times: Adisak, Jeff, Ken, Lorenz, Pablo, Rolf, Steve, and Youngmi. My thanks also go out to the following group members from the latter period: Gireesh, Greg, Guido, Isaac, Martin, Masayoshi, Matthew, Mingjie, Nandita, Pablo, Rui, Sundar, and Yashar.

I would also like to thank my family: my father Shinn-Fu, my mother Jen-Ian, my brother Ki and my sister Tian for always encouraging me to make the most out of my life.

And most of all, I would like to thank my wife, Ingrid, for her unconditional support, encouragement, and patience. This thesis is dedicated to my loving wife.

Contents

List of Figures

Chapter 1

Introduction

The Internet consists of end-hosts, links, and routers. This thesis is about the design and architecture of high performance Internet routers that make up the backbone of the Internet — in particular, how to design them to perform predictably and yet still be implementable.

A router consists of several processing stages. When designing a router to have predictable performance (we will explain below exactly what we mean by "predictable"), we need to make sure that each processing stage behaves predictably. At the very least, a router has two main processing stages: address lookup and switching. In the address lookup stage, the router examines the packet header to decide where the packet should be sent next. In the switching stage, the router transfers the packet to the correct outgoing port in preparation to depart. This thesis focusses on the switching stage of the router.

Some routers have more processing stages. For example, if a router provides bandwidth or delay guarantees to different packet flows, it needs to be "flow-aware" [1]. Flow-aware routers include firewalls, load-balancers, and routers that provide per-flow delay and bandwidth guarantees. When a packet arrives to a flow-aware router, the router first has to decide which flow the packet belongs to. This is called *classification.* Usually, a packet is then buffered in a queue dedicated to its flow so that the packets within a flow are all treated the same way and are processed in FIFO order.

Figure 1-1 shows how a packet is processed by a flow-aware router from when it arrives on the ingress line until it departs on the egress line. The packet goes through

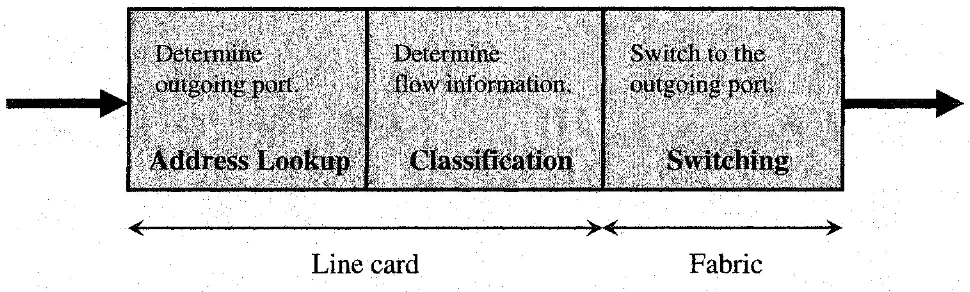

Figure 1-1: Datapath of a packet through a flow-aware router.

the following stages. In the address lookup stage, the router determines the outgoing port of the packet. Then, the router determines which flow the packet belongs to in the classification stage. In the switching stage, packets are segmented into cells, traverse the switch fabric as cells, and reassembled back into packets again before they leave the router.[1]

Network operators need to design and plan their networks. They know the capacity of the links, and they are told the capacity of the router. However, although they can use up to 100% of the links, it is perhaps surprising that they can not use up to 100% of the router capacity. A router's capacity depends on the arriving traffic patterns that expose imperfections in the router, such as blocking or inefficient scheduling algorithms. No commercial router today can guarantee that 100% of its capacity is available to the network operator. This makes it hard to plan a network — not just under normal operating conditions (when utilizations are typically low), but under failure conditions when traffic is rerouted and the utilization of links and routers grows. Put another way, when a network operator spends millions of dollars on a router for its backbone network, it seems reasonable to expect that one can use all of the router capacity, regardless of the arrival process of packets. In other words, as long as the router receives packets at a rate within its capacity, it should be able to fully utilize the links and, therefore, guarantee 100% throughput.

Furthermore, network operators want their networks to be designed to provide their customers with a number of service guarantees. These services (which are the

[1]Although packets arriving to the router may have variable length, we will assume that they are treated internally as fixed-length *cells*. This is common in high performance routers [2, 3].

basis of Service Level Agreements (SLAs)) provide guaranteed bandwidth and delay through their networks. For example, large corporations might want to guarantee a fixed bandwidth between their company sites. Similarly, if a network operator can guarantee a maximum delay through their networks, they can also sell services to customers who run real-time applications such as video and voice. In order for a network to provide bandwidth and delay guarantees, each individual router within the network must also provide bandwidth and delay guarantees.

In summary, network operators want routers which provide throughput, bandwidth, and delay guarantees. However, most high performance routers built today use architectures based on crossbars; and building crossbar-based routers which provide these performance guarantees is difficult using current technology. The bottleneck that prevents routers from providing these guarantees is scheduling in the switching stage. A crossbar-based router requires a centralized scheduler to determine when cells are to traverse the switch fabric. As line rates and number of ports increase, due to growth of Internet traffic, the scheduling problem will only become more difficult. In this thesis, I will introduce a set of new scheduling algorithms that provide performance guarantees and simplify the scheduling process.

The remainder of the introductory chapter describes in detail the desired performance guarantees and gives an overview of switch architectures and previous work. Finally, the chapter describes the motivation of the thesis and outlines new algorithms and architectures presented later in this thesis.

1.1 Performance Guarantees

Network operators would like to build their networks using routers that give performance guarantees. They want routers that provide throughput, bandwidth and delay guarantees. These guarantees are the same guarantees an output queued (OQ) switch (shown in Figure 1-2) can provide.

When a packet arrives at an OQ switch, it is immediately placed in a memory dedicated to its outgoing line, where it waits until it leaves the switch. The output chooses which packet to send next from among all the packets waiting to leave from

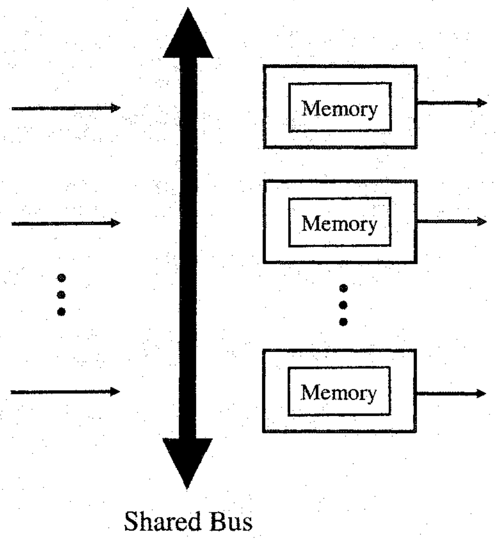

Figure 1-2: Output Queued (OQ) Switch

the output. Because nothing prevents an output from keeping the output line busy whenever it has a packet, we say the switch is "work conserving". A work conserving switch has the highest throughput of any switch because the output line cannot carry any higher workload. It also has the lowest expected packet delay because — on average — packets leave earlier in a work-conserving switch than in any other switch.[2]

It is also well known that an OQ switch can allocate bandwidth to different flows of packets using techniques such as weighted fair queueing (WFQ) [4, 5] and deficit round robin (DRR) [6]. Furthermore, with an appropriate scheduling algorithm and sufficient internal buffers, an OQ switch can control the delay of individual packets through the switch [4, 5].

[2]If packets are all the same length, then a work-conserving switch with any service policy — such as FCFS — minimizes expected packet delay. If packets are of unequal length, then a specific work-conserving policy — such as shortest remaining processing time — minimizes expected packet delay.

1.1.1 Throughput Guarantees

A generic switch with N inputs and N outputs is shown in Figure 1-3. Assume time is slotted into cell times. $A_{i,j}(n)$ is the cumulative number of arrivals to input i of cells destined to output j at time n while $A_i(n)$ is the aggregate number of arrivals to input i. During each cell time, at most one cell can arrive at each input. $\lambda_{i,j}$ is the arrival rate of $A_{i,j}(n)$. $D_{i,j}(n)$ is the cumulative number of departures from output j of cells that arrived from input i while $D_j(n)$ is the aggregate number of departures from output j. Similarly, during each cell time, at most one cell can depart from each output. $X_{i,j}(n)$ is the total number of cells from input i to output j still in the system at time n. The evolution of cells from input i to output j can be represented as

$$X_{i,j}(n+1) = X_{i,j}(n) + A_{i,j}(n) - D_{i,j}(n).$$

Let $A(n)$ denote the vector of all arrivals $\{A_{i,j}(n)\}$, $D(n)$ denote the vector of all departures $\{D_{i,j}(n)\}$, and $X(n)$ denote the vector of the number of cells still in the system. With this notation, the evolution of the system can be described as

$$X(n+1) = X(n) + A(n) - D(n).$$

Definition 1 *An arrival process is said to be admissible when no input or output is oversubscribed, i.e., when* $\sum_i \lambda_{i,j} < 1$, $\sum_j \lambda_{i,j} < 1$, $\lambda_{i,j} \geq 0$.

Definition 2 *Traffic is called independent and identically distributed (iid) if and only if:*

1. Every arrival is independent of all other arrivals both at the same input and at different inputs.

2. All arrivals at each input are identically distributed.

Definition 3 *A switch is said to achieve 100% throughput if under any admissible iid traffic, for every* $\epsilon > 0$, *there exists* $B > 0$ *such that*

$$\lim_{n\to\infty} Pr\{\sum_{i,j} X_{i,j}(n) > B\} < \epsilon$$

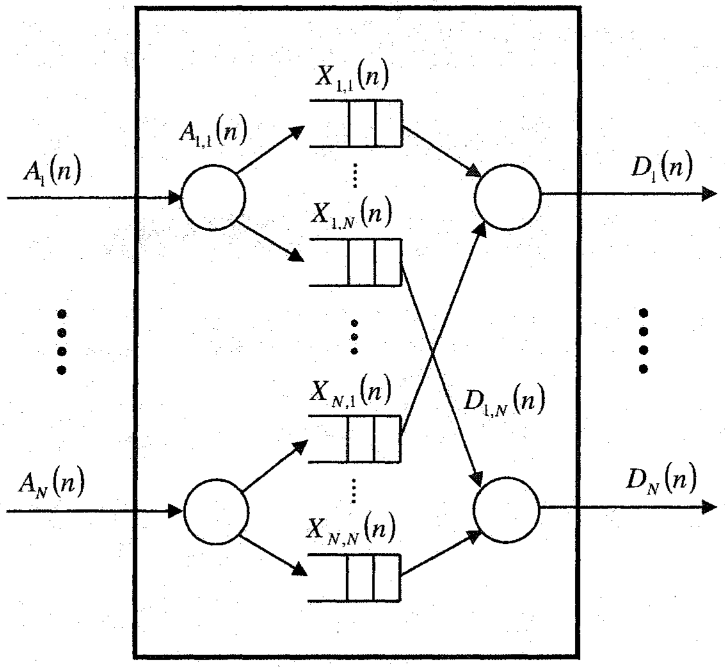

Figure 1-3: A simple model of a generic switch with N inputs and N outputs.

Definition 4 *A switch is said to be work-conserving if whenever there is a cell in the system for an output, then that output must be busy.*

In an OQ switch, an output can select any cell destined to that output to be placed on the outgoing line. Therefore, if an output selects a cell when any of the queues associated with that output is nonempty, an OQ switch is work-conserving. Note that when a switch is work-conserving, outputs are idle only if there are no cells in the system for that output. This is the busiest any system can be and therefore maximizes throughput and minimizes the number of cells stored in the switch. So a switch that is work-conserving also provides 100% throughput.

1.1.2 Bandwidth and Delay Guarantees

Bandwidth and delay guarantees can be provided by implementing a weighted fair queuing (WFQ) scheduler in an OQ switch. Arriving packets are classified into flows and stored in per-flow queues. When a packet arrives, its finishing time is

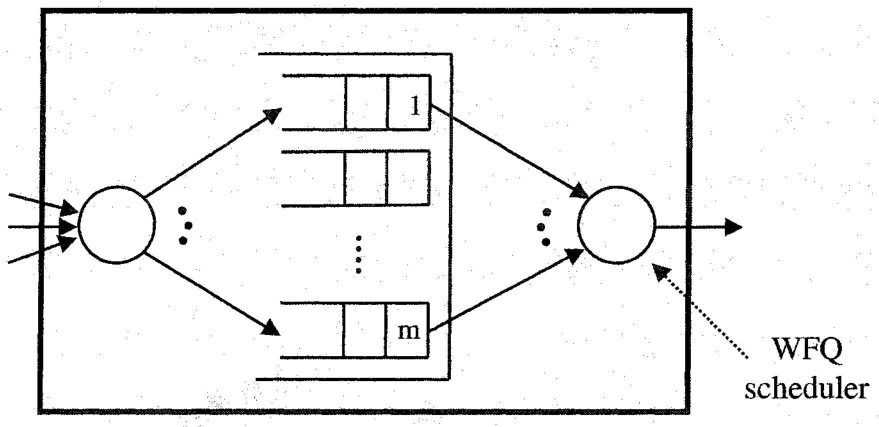

Figure 1-4: A diagram of one output of an OQ switch using a weighted fair queueing (WFQ) scheduler.

calculated. Packets are then serviced by the weighted fair queueing scheduler in the order of their finishing times. This has been proved to guarantee that each packet will depart the switch by its assigned finishing time and therefore makes it possible to bound the delay of the packet through the switch. More information can be found in the following references [4, 5]. Figure 1-4 shows the usual way to represent an OQ switch that implements WFQ. Each output consists of many logical First-In-First-Out (FIFO) queues, one for each flow. The WFQ scheduler decides which packet to transmit next by considering the head-of-line packet in each FIFO, and choosing the one with the earliest finishing time.

For our purposes here, we are going to use a different way to represent an OQ switch that implements WFQ. Instead of one queue for each flow, we are going to represent the output using a single queue that we call a Push-In-First-Out (PIFO) queue. As we will see later, this turns out to be a useful and equivalent abstraction of WFQ, that is easier to analyze. We will use the PIFO queue in several of our results in this thesis.

A PIFO is a single queue of all packets waiting to depart from an output. When a new packet arrives for the output, it is "pushed-in" to some location in the queue. Once in the queue, the cell's relative ordering with packets already in the queue does not change; packets can not switch places. Of course, new packets can arrive

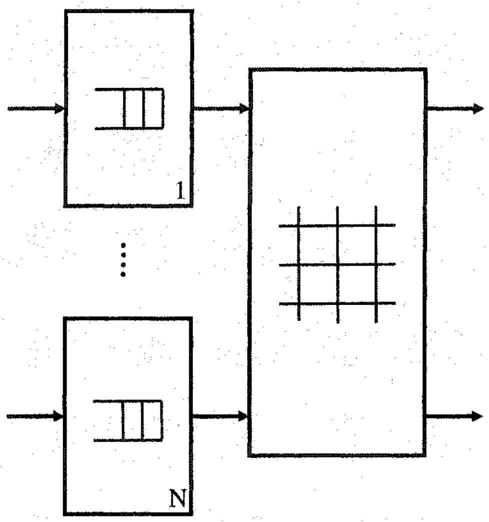

Figure 1-5: An Input Queued (IQ) switch.

later and get pushed-in ahead of (or behind) the packet. Packets can only depart from the head of line. Essentially, when a packet arrives, the WFQ scheduler picks its departure order relative to other packets already in the output. In this way, a WFQ scheduler with a PIFO queue is exactly equivalent to a WFQ scheduler with per-flow FIFO queues. By selecting the finishing times of packets as before, an OQ switch using a PIFO queueing policy per output can provide bandwidth and delay guarantees. Furthermore, the PIFO model includes several other queueing policies such as weighted round-robin and strict priorities. For an OQ switch, each output maintains a queue for the cells waiting to depart the switch. A single PIFO queue can be used to model this by arranging the cells in the queue based on the departure order of the cells.

1.2 Switch Architectures

An OQ switch can behave as well as an ideal router and provides the best performance guarantees; however, an OQ switch is impractical for high performance routers since the memory bandwidth of a $N \times N$ OQ switch must run N times as fast as the line rate. Unfortunately, with high line rates, memories with sufficient bandwidth are

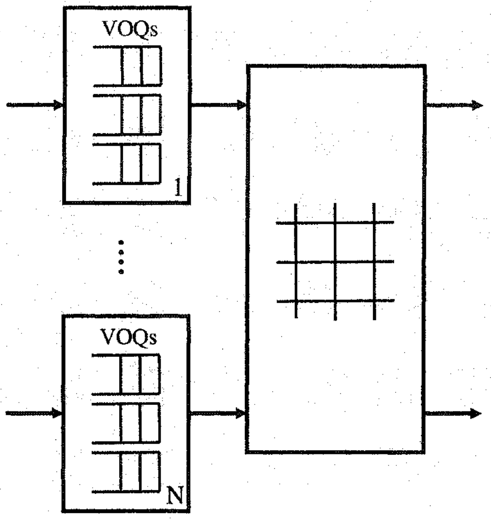

Figure 1-6: An Input Queued (IQ) switch with Virtual Output Queues (VOQs).

simply not available.

On the other hand, an input queued (IQ) switch shown in Figure 1-5 needs memory to run only as fast as the line rate. This makes input queueing very appealing for switches with fast line rates or with a large number of ports. For this reason, the highest performance switches and routers use input-queued (IQ) switches [2, 3]. However, IQ switches which use a single FIFO queue can suffer from head-of-line (HOL) blocking which occurs if the head-of-line cell blocks cells destined for a different output to be scheduled. This phenomenon can have a severe effect on throughput. It is well known that if each input maintains a single FIFO, then HOL blocking can limit the throughput to just 58.6% [7].

Fortunately, HOL blocking can be removed by using a simple buffering strategy at each input. Rather than maintain a single FIFO queue for all cells, each input maintains a separate FIFO queue for each output [8, 9, 10] as shown in Figure 1-6. This queueing discipline is often referred to as virtual output queueing (VOQ). HOL blocking does not occur because a cell cannot be blocked by a cell queued ahead of it that is destined for a different output. Furthermore, no additional speedup is required because at most one cell can arrive and depart from each input in a cell time. It has been shown that using a maximum weight matching algorithms such as MWM [9],

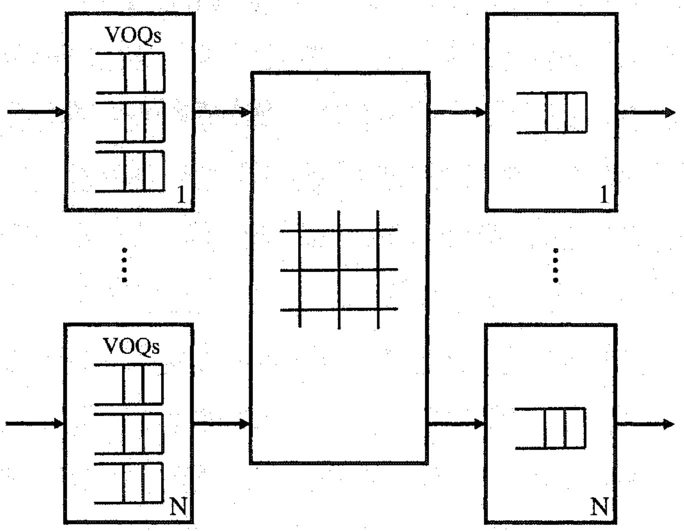

Figure 1-7: A Combined Input Output Queued (CIOQ) switch with Virtual Output Queues (VOQs).

OCF [11], or LPF [12], a switch using VOQs can achieve 100% throughput. However, these algorithms are too complex and are hard to implement in practice.

One method that has been proposed to increase performance is to increase the "speedup" of a switch. A switch with a speedup of S can remove up to S packets from each input and deliver up to S packets to each output within a time slot, where a time slot is the time between packet arrivals at input ports. Hence, an OQ switch must have a speedup of N whereas an IQ switch may have a speedup of one. For values of S between 1 and N, packets need to be buffered at the inputs before switching as well as at the outputs after switching. We call this architecture a combined input and output queued (CIOQ) switch shown in Figure 1-7.

Both analytical and simulation studies of a CIOQ switch which maintains a single FIFO at each input have been conducted for various values of speedup [13, 14, 15, 16]. A common conclusion of these studies is that with a speedup of four to five one can achieve about 99% throughput when arrivals are independent and identically distributed at each input and when the distribution of packet destinations is uniform across the outputs. However, not only do these studies consider only average delay (and simplistic input traffic patterns), they also make no guarantees about the delay of individual packets. This is particularly important if a router is to offer QoS guarantees.

1.3 Motivation of Thesis

We believe that a well-designed network switch should perform predictably in the face of all types of arrival process and allow the delay of individual packets to be controlled. So instead of finding values of speedup that work well on average or with simplistic and unrealistic traffic models, we want to figure out how a CIOQ switch mimics an OQ switch for all types of traffic. (Here, "mimics" means that when the same inputs are applied to both the OQ switch and our crossbar-based switch, the corresponding output processes from the two switches are completely indistinguishable.) This thesis solves problems associated with designing a CIOQ switch that mimics an OQ switch.

1.3.1 Minimum Speedup to Emulate an OQ Switch

The approach of mimicking an OQ switch was first formulated in [17]. They showed that a CIOQ switch with a speedup of four can mimic a FIFO OQ switch for arbitrary input traffic patterns and switch sizes using an algorithm called Most Urgent Cell First Algorithm (MUCFA). Is there a speedup smaller than four that also provides the same behavior? Our objective is to find the minimum speedup required for a CIOQ switch to mimic an OQ switch.

1.3.2 Emulating a PIFO OQ Switch

In [17], it was shown that mimicking a FIFO OQ switch was achievable. A FIFO OQ switch is work-conserving, but does not provide delay guarantees. An OQ switch that employs a PIFO model such as WFQ can provide delay guarantees. Therefore, another problem is to find an algorithm that allows a CIOQ switch to mimic a PIFO OQ switch.

1.3.3 Providing Emulation in a Practical Way

The MUCFA algorithm requires a speedup of four and is too complex to implement in practice. A centralized location was required to keep track of the exact departure time of each cell in the system. Without finding a practical algorithm, mimicking an OQ switch is only theoretically interesting. Another problem we solve in this thesis is to find practical methods to mimic an OQ switch.

1.4 Outline of Thesis

This thesis explains how to design a router to achieve the same throughput, bandwidth, and delay guarantees as an ideal OQ router. In Chapter 2, we present an analytical counting method that can be used to show how a CIOQ switch using a traditional crossbar can mimic an OQ switch. Running at a speedup of two, we show how cell scheduling on a traditional crossbar can theoretically mimic an OQ switch with delay guarantees. We will then discuss the communication complexity and describe how it is impractical for implementation.

In Chapter 3, we introduce a crossbar with a small amount of internal buffering. We will show how the counting method can also be applied on a CIOQ switch using a buffered crossbar to achieve the same behavior while simplifying the scheduling process. The key to the buffered crossbar is to allow inputs and outputs to schedule independently and in parallel. We will also show how a buffered crossbar running at a speedup of two can achieve 100% throughput and mimic an OQ switch with a distributed algorithm. We will also introduce a novel mechanism called header scheduling which allows a buffered crossbar running at a speedup of two to mimic an OQ switch with delay guarantees.

In the concluding chapter, we will suggest how to implement a router which achieves the same throughput, bandwidth, and delay guarantees as an ideal router. The proposed architecture incorporates the practical benefits of a buffered crossbar and provides a simple path to scale crossbar-based routers.

Chapter 2

Mimicking an OQ Switch Using a Traditional Crossbar

In this chapter, we show that a CIOQ switch with a speedup of two can behave identically to an OQ switch. The result holds for switches with an arbitrary number of ports and for any traffic arrival pattern. This result is also found to be true for a broad class of widely used output link scheduling algorithms such as weighted fair queueing, strict priorities, and FIFO. We introduce some specific scheduling algorithms that achieve this result. We also show more generally that a speedup of $2 - 1/N$ is both necessary and sufficient for a CIOQ switch to mimic a FIFO OQ switch.

It is worth briefly considering the implications of this result. It demonstrates that it is possible to mimic an $N \times N$ OQ switch using buffer memory operating at only twice the speed of the external line. Previously, an OQ switch could only be implemented with memories operating at N times the speed of the external line. However, the advantages do not come for free. In essence, the memory bandwidth is reduced at the expense of a fast cell scheduling algorithm that is required to configure the crossbar. As we shall see, the scheduling algorithms are complex and not yet practicable to mimic fast OQ switches with a large number of ports. While we propose some strategies in this chapter, the later chapters provide a more practical solution.

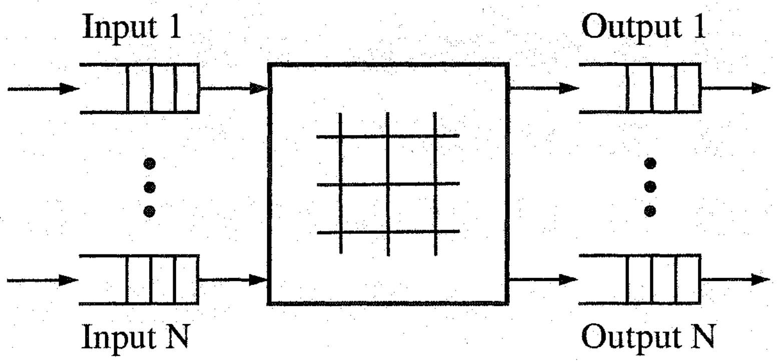

Figure 2-1: A general Combined Input and Output Queued (CIOQ) switch

2.1 Background

Consider the single stage, $N \times N$ switch shown in Figure 2-1. Throughout this thesis we assume that the switch is empty before time $t = 1$. The switch is said to have a speedup of S if it can remove up to S cells from each input and transfer at most S cells to each output in a time slot. A speedup of S requires the switch fabric to run S times as fast as the input or output line rate. For $1 < S < N$ buffering is required both at the inputs and at the outputs and leads to a combined input and output queued (CIOQ) architecture. We wish to solve the following problem:

The speedup problem: Determine the smallest value of S and an appropriate cell scheduling algorithm π that

1. allows a CIOQ switch to exactly mimic the performance of an output-queued switch (in a sense that will be made precise),

2. achieves this for any arbitrary input traffic pattern, and

3. is independent of the switch size.

We will require that any solution of the speedup problem provides the following feature: a CIOQ switch must behave identically to an OQ switch in the following sense:

Definition 5 : Identical Behavior – *A CIOQ switch is said to mimic an OQ switch if, under identical inputs, the departure time of every cell from both switches is identical.*

As a benchmark with which to compare our CIOQ switch, we will assume that there exists a shadow $N \times N$ OQ switch that is fed the same input traffic pattern as the CIOQ switch. Our goal is to arrange for each cell to depart from the CIOQ switch at exactly the same time as its counterpart cell departs from the OQ switch. In the CIOQ switch, the sequence in which cells are transferred from their input queues to the output queue is determined by a scheduling algorithm. In each time slot, the scheduling algorithm matches each non-empty input with at most one output and, conversely, each output is matched with at most one input. The matching is used to configure the traditional crossbar fabric before cells are transferred from the input side to the output side. A CIOQ switch with a speedup of S is able to make S such transfers during each time slot.

Selecting the appropriate scheduling algorithm is the key to making a CIOQ switch mimic its shadow OQ switch. In order to provide identical behavior, a cell must be transferred to the output of the CIOQ switch before the departure time of the counterpart cell in the OQ switch. If the cell is prevented from reaching its output in time, the departures from both switches are not identical, and we will fail to mimic the shadow OQ switch.

2.2 General Structure of Scheduling Algorithms

For most of this thesis, we break each time slot of a CIOQ switch with a speedup of S into the following three phases:

- **The Arrival Phase** – All arrivals of new cells to the input ports take place during this phase.
- **The Scheduling Phases** – The scheduling algorithm selects cells to transfer from inputs to outputs and then transfers them across the crossbar. With a speedup of S, the scheduling algorithm makes S crossbar configurations.
- **The Departure Phase** – All departures of cells from the output ports take place during this phase.

The order in which the three phases occur is not crucial to our algorithms. However, we shall use the above ordering as it makes our proofs simpler.

A *matching* of input ports to output ports is a set of cells waiting on the input side such that all these cells can be sent across the crossbar in a single transfer. During each scheduling phase the scheduler finds a *stable* matching between the input ports and the output ports.

In order to find a stable matching, it is assumed that each input of the CIOQ switch maintains a priority list, which can be thought of as an ordered set of cells waiting at the input port. A cell can be prevented from reaching its output on time by other cells at its input with a higher priority. The more cells ahead of it in the priority list, the longer it might take to be transferred to the output. Many orderings of the cells are possible — each ordering leading to a different scheduling algorithm, as we shall see.

Each output maintains a queue for the cells that have been transferred from the inputs and that are waiting to depart from the switch. In addition, each output also maintains an output priority list, an ordered list of cells at the inputs waiting to be transferred to this particular output. The output priority list is constructed based on the order in which the cells would depart from the OQ switch we wish to mimic, i.e., the shadow OQ switch. This priority list will depend on the queueing policy such as WFQ, strict priorities, FIFO, etc., followed by the OQ switch.

Definition 6 : Stable Matching – *A matching of input ports to output ports is said to be stable if for each cell c waiting in an input queue, one of the following holds:*

1. Cell c is part of the matching; i.e., c will be transferred from the input side to the output side during this scheduling phase.

2. A cell that is ahead of c in its input priority list is part of the matching.

3. A cell that is ahead of c in its output priority list is part of the matching.

Notice that conditions 2 and 3 above may be simultaneously satisfied, but condition 1 excludes the other two. The conditions for a stable matching can be achieved using the so-called *stable marriage problem.* Solutions to the stable marriage problem are called stable matchings and were first studied by Gale and Shapley [18]; they gave an algorithm that finds a stable matching in at most M iterations, where M is the sum of the lengths of all the input priority lists.

Our specification of the scheduling algorithm for a CIOQ switch is almost complete: the only thing that remains is to specify how the input queues are maintained. Different ways of maintaining the input queues result in different scheduling algorithms. In fact, the various scheduling algorithms presented later differ only in the ordering of the cells in their input queues. For reasons that will become apparent, we restrict ourselves to a particular class of orderings, which is defined as follows:

Definition 7 : Input Priority List Ordering – *When a cell arrives, it is given a priority number that dictates its position in the list; i.e., a cell with priority number X is placed at location $(X+1)^{th}$ from the head of the list. A cell is placed in an input priority list according to the following rules:*

1. Arriving cells are placed at an arbitrary location in the list,

2. The relative ordering of cells in the list does not change once cells are in the list, i.e., cells in the list cannot switch places, and

3. Cells may be selected to depart from the list from any location.

Thus, to complete our description of the scheduling algorithms, we only need to specify an insertion policy which determines where an arriving cell gets placed in its input priority list.

On the output side, the CIOQ switch keeps track of the departure order of each waiting cell. During each time slot the cell that departs from an output and is placed onto the outgoing line is the one with the earliest departure order. For the CIOQ switch to successfully mimic the shadow OQ switch, we must ensure that each cell crosses over to the output side before it is time for the cell to leave.

Even before we finish defining the algorithm, we can already see that it must maintain a large amount of state. More importantly, the algorithm must keep track of a large amount of *global* state, taking into account information about the queues at all the inputs and all the outputs. We will discuss in Section 2.6 the information complexity of these algorithms, and the difficulty of implementing them at high speed.

2.3 Definitions

Based on the input and output priority lists, we can describe the following definitions that are crucial to the rest of the thesis.

Definition 8 : Output Cushion — *At any time, the output cushion of a cell c, $OC(c)$, is the number of cells waiting in the output buffer at cell c's output port with an earlier departure order than cell c.*

Notice that if a cell is still on the input side and has a small (or zero) output cushion, the scheduling algorithm must urgently deliver the cell to its output so that it may depart on time. Since the switch is work-conserving, a cell's output cushion decreases by one during every time slot, and can only be increased by newly arriving cells that are destined to the same output with an earlier departure order.

Definition 9 : Input Thread — *At any time, the input thread of cell c, $IT(c)$, is the number of cells ahead of cell c in its input priority list.*

In other words, $IT(c)$ represents the number of cells currently at the input that need to be transferred to their outputs more urgently than cell c. A cell's input thread is decremented only when a cell ahead of it is transferred from the input and is possibly incremented by newly arriving cells. Notice that it would be undesirable for a cell to simultaneously have a large input thread and a small output cushion — the cells ahead of it at the input may prevent it from reaching its output before its departure time. This motivates our definition of *slackness*.

Definition 10 : Slackness – *At any time, the slackness of cell c, $L(c)$, equals its output cushion minus its input thread; i.e., $L(c) = OC(c) - IT(c)$.*

Slackness is a measure of how large a cell's output cushion is with respect to its input thread. If a cell's slackness is small, then it urgently needs to be transferred to its output. Conversely, if a cell has a large slackness, then it may languish at the input without fear of missing its departure time.[1] Our approach will be to find

[1]Note that a cell's input thread and slackness are only defined when the cell is waiting at the input side of the switch.

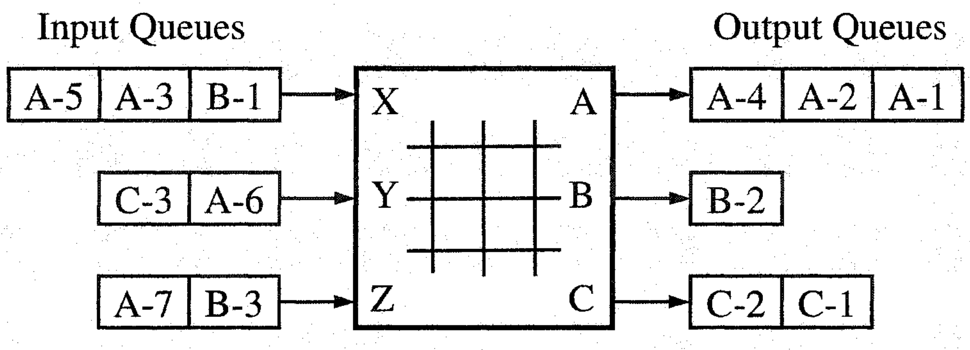

Figure 2-2: A snapshot of a CIOQ switch

scheduling algorithms for which the slackness is always non-negative. Although not strictly necessary, this will ensure that when a cell is transferred to the output its output cushion is non-negative. The idea is that when a cell's output cushion reaches zero, the cell's input thread must also equal zero. This means either: (1) that the cell is already at its output, and will depart the output on time, or (2) that the cell is at the head of its input priority list (because its input thread is zero), and will be transferred to the output immediately, which ensures that the cell will depart the output on time.

Figure 2-2 shows a snapshot of a CIOQ switch with a number of cells waiting at its inputs and outputs. For convenience we assume the time the snapshot was taken to be 1. Let P-t denote a cell that, in the shadow switch, will depart from output port P at time t. Consider, for example, the cell c denoted in the figure by A-3. For the CIOQ switch to mimic the shadow OQ switch, the cell must depart from port A at time 3. Its input thread is $IT(c) = 1$, since B-1 is the only cell ahead of c in the input priority list. Its output cushion is $OC(c) = 2$, since out of the three cells queued at A's output buffer, only two cells A-1 and A-2 will depart before it. Further, the slackness of cell c is given by $L(c) = OC(c) - IT(c) = 1$.

2.4 Necessity and Sufficiency

Having defined the general structure of the scheduling algorithms, we now address the next natural question: what is the minimum required speedup, S, for a CIOQ switch to mimic an OQ switch. The following theorem answers this question.

Theorem 1 *(Necessity). An $N \times N$ CIOQ switch needs a speedup of at least $2 - 1/N$ to mimic an $N \times N$ FIFO OQ switch.*

Proof: The proof is by counter-example and is presented in Appendix A.1. ■

Remark: Since FIFO is a special case of a variety of output queueing disciplines, e.g., weighted fair queueing, strict priorities, etc., the lower bound applies to these queueing disciplines as well.

Theorem 2 *(Sufficiency). An $N \times N$ CIOQ switch with a speedup of $2 - 1/N$ can mimic an $N \times N$ FIFO OQ switch.*

Proof: The proof is based on an insertion policy that we call Last In Highest Priority (LIHP) and is presented in Appendix A.2. ■

2.5 A Simple Input Queue Insertion Policy for a Speedup of 2

The proof of Theorem 2 is based on the LIHP input queue insertion policy and is, unfortunately, complex and somewhat counterintuitive. Further, LIHP is complex to implement, making it of little practical value. So in an attempt to provide a more intuitive understanding of the speedup problem, we present a simple and slightly more practical insertion policy that, with a speedup of two, mimics an OQ switch with a PIFO queueing discipline. We call this insertion policy Critical Cells First (CCF). As shown in the introduction, the broad class of PIFO queueing policies includes widely-used queueing policies such as WFQ and strict priority queueing. Notice that with an arbitrary PIFO policy, the departure time of a cell never decreases, but may increase because of the arrival of higher priority cells.

Recall that to specify a scheduling algorithm for a CIOQ switch, we simply need to give an insertion policy for the input queues. Critical Cells First (CCF) inserts an arriving cell as far from the head of its input queue as possible, such that the input thread of the cell is not larger than its output cushion. More formally:

The CCF Insertion Policy: Suppose cell c arrives at input port P. Let X be the output cushion of c. Insert cell c into the $X + 1^{th}$ position from the front of the input priority list at P. Hence, upon arrival cell c has a slackness of zero. If the size of this list is less than X cells, then place c at the end of the input priority list at P. Therefore, in this case, c has a positive slackness.

A consequence of CCF is that the slackness of a cell is always non-negative upon arrival. The intuition behind this insertion policy is that a cell with a small output cushion needs to leave soon (i.e., it is more critical) and therefore needs to be delivered to its output sooner than a cell with a larger output cushion. In other words, a cell with a large output cushion can safely reside further from the head of its input queue.

We now prove that CCF with a speedup of two mimics an OQ switch. Informally, the proof proceeds as follows. We first show a property of the CCF algorithm: that a cell never has a negative slackness, i.e., the input thread of a cell never exceeds its output cushion. We then proceed to show how this ensures that a cell always reaches the output side in time to leave.

Lemma 3 *The slackness $L(c)$ of a cell c waiting on the input side is non-decreasing from time slot to time slot.*

Proof: Let the slackness of a cell c be $L(c)$ at the beginning of a time slot. During the arrival phase, the input thread of c can increase by at most one because an arriving cell might be inserted ahead of c in its input priority list. During the departure phase, the output cushion of c decreases by one. Now consider what happens in a scheduling phase. If c is scheduled in a scheduling phase, then it is delivered to its output and we no longer need to concern ourselves with c. If c is not scheduled, either a cell ahead of c in its input priority list or a cell ahead of c in its output priority list is scheduled (by the property of stable matchings, see Definition 6). Therefore, during a scheduling phase, either the input thread of c decreases by one, or the output cushion of c increases by one. The slackness of c, therefore, increases by at least one during each scheduling phase. Counting the changes in each of the three phases (arrival, departure, and two scheduling phases), we conclude that the slackness of cell c cannot decrease from time slot to time slot. ■

Corollary 4 *The slackness $L(c)$ of a cell c is always non-negative.*

Proof: Since the slackness of an arriving cell is non-negative due to the CCF insertion policy, it follows from Lemma 3 that the slackness of a cell is always non-negative. ■

Theorem 5 *Regardless of the incoming traffic pattern, a CIOQ switch that uses CCF with a speedup of 2 mimics a PIFO OQ switch.*

Proof: Suppose that the CIOQ switch has successfully mimicked the OQ switch up until time slot $t-1$, and consider the beginning (first phase) of time slot t. We must show that any cell reaching its departure time is either: (1) already at the output side of the switch or (2) will be transferred to the output during time slot t. From Corollary 4, we know that a cell always has a non-negative slackness. Therefore, when a cell reaches its time to leave (i.e., its output cushion has reached zero), the cell's input thread must also equal zero. This means either: (1) that the cell is already at its output and may depart on time or (2) that the cell is simultaneously at the head of its input priority list (because its input thread is zero) and at the head of its output priority list (because it has reached its departure time). In this case, the stable matching algorithm is guaranteed to transfer it to its output during the time slot, and therefore the cell departs on time. ■

2.6 Towards Making CCF Practical

CCF as presented above suffers from two main disadvantages. First, the stable matching that we need to find in each scheduling phase can take as many as M iterations, where M is the sum of the lengths of all the input priority lists. Second, the algorithm has a high information complexity — CCF needs to know both the output cushion and departure order of each cell at the inputs, information that is not locally available at each input, but depends on the state of all the switch outputs.

We address these disadvantages in this section. The Delay Till Critical (DTC) strategy reduces the number of iterations needed to compute a stable matching to N; and the Group By Virtual Output Queue (GBVOQ) algorithm can mimic FIFO OQ switches without using global information. Unfortunately, combining the solutions does not help, i.e., we can reduce either the number of iterations to N or reduce the information complexity, but not both at the same time using these algorithms.

2.6.1 The Delay Till Critical (DTC) Strategy

The Delay Till Critical strategy is as follows: *During each scheduling phase, mark as "active" all cells with a slackness of zero and mark all other cells as inactive.* The stable matching algorithm now considers only active cells. Intuitively, cells with zero slackness are the most critical and should be considered for immediate transfer across the fabric. Since the slackness of a cell can never become negative,[2] CCF combined with DTC strategy can mimic any OQ switch that follows a PIFO queueing policy.

We will show that this simple strategy reduces the number of iterations required to compute a stable matching to N. Before we prove this fact, let us examine the problem that we are trying to remove. At any time instant, we define the dependency graph G to be a directed graph with a vertex corresponding to each active cell that is waiting on the input side of the CIOQ switch. Let a and b be two cells waiting at the input side. There is a directed edge from b to a if and only if cell a is ahead of b either in an input priority list or in an output priority list. Clearly two cells have to share either the same input port or the same output port if there is to be an edge between them. If we use CCF as defined in Section 2.5, there may be cycles in this dependency graph. These cycles are the main cause of inefficiency in finding stable matchings, and the DTC strategy is designed to remove these cycles.

Lemma 6 *If DTC is used in conjunction with CCF then, during any scheduling phase, the dependency graph is acyclic.*

[2]As soon as the slackness becomes zero, the cell would be marked active and the slackness would increase by one during the current scheduling phase (see Lemma 3). To prevent the slackness of a cell from decreasing by two before the next scheduling phase, we assume that the departure phase occurs before the second scheduling phase.

We defer the proof of Lemma 6 to Appendix B and instead focus on its implications, and how a match can be constructed in N iterations. First, let us consider how the first cell in the match is found. Since there are no cycles in G, there has to be at least one "sink" (i.e., a cell with no outgoing edges). Let cell c be the sink. Since there are no active cells ahead of c in either its input priority list or its output priority list, cell c has to be part of any stable matching of active cells. Hence c is guaranteed to be transferred to the output side, and therefore we can remove from the graph all cells which have the same input or output port as c; they clearly cannot be part of the match. The resulting graph is again acyclic, and we can repeat the above procedure $N - 1$ more times to obtain a stable matching. Notice that each iteration of the above N iteration algorithm is quite straightforward.

We now address the second disadvantage of CCF, i.e., that of high information complexity.

2.6.2 The Group By Virtual Output Queue (GBVOQ) Algorithm

With CCF, the stable matching algorithm needs to calculate both the departure order and output cushion of each cell in the input queues. These quantities require centralized information about the state of all the queues in the system, making CCF (as described) unsuitable for a distributed implementation. However, for mimicking a FIFO OQ switch, we can group incoming cells into Virtual Output Queues and obtain an upper bound of N on the number of cells that need to be considered from each input. GBVOQ, the algorithm which achieves this bound, is described below.

At each input, GBVOQ maintains a single priority list as before, as well as a VOQ for each output port. All cells belong to a VOQ and to the single input priority list. When a new cell arrives, it is always placed at the tail of the corresponding VOQ. If the VOQ is empty, the new cell is placed at the head of the input priority list. If, on the other hand, the VOQ is non-empty, the new cell is inserted in the input priority list just behind the last cell belonging to the same VOQ; i.e., all cells that are in the same VOQ occupy contiguous positions in the input priority list. Therefore, to make a scheduling decision, it is sufficient to only keep track of the relative priority ordering of VOQs. Since there are at most N VOQs at an input port in a FIFO switch, the size of the input priority list is bounded. The following lemma, which is proved in Appendix C, shows that GBVOQ assigns a non-negative slackness to an incoming cell.

Lemma 7 *The slackness $L(c)$ of a newly arriving cell c is non-negative using the GBVOQ algorithm.*

From Lemma 7, a CIOQ switch that uses GBVOQ with a speedup of two successfully mimics a FIFO OQ switch. At first glance, this insertion policy seems unfair. However, when a cell arrives to an empty VOQ, it is possible that there are no other cells in the system destined to that output. Therefore, the cell must immediately be transferred to the output in order to keep that output busy. The above insertion policy is designed to take care of this case.

Apart from small priority lists, GBVOQ has other desirable properties. First, the decision of where an incoming cell needs to be inserted is much simpler for GBVOQ than CCF — each input port can maintain its local priority queue without any access to global information. Second, during the stable matching phase, to determine which of two cells has a higher output priority, we only need to compare the arrival timestamps of the two cells. The cell which arrived earlier will have a smaller departure order (and hence a higher output priority) because of the FIFO property.

However, GBVOQ requires up to N^2 iterations. The reason is that if two cells at the same input port are destined to the same output port, the one with the earlier departure order occurs ahead of the other in both the input and output priority list. Therefore, only the cell at the head of a VOQ needs to be considered. The DTC strategy cannot be applied to GBVOQ since to determine if a cell needs to be marked active, we again need access to global state, namely the output cushion of each cell at the head of a VOQ. Finding a solution which simultaneously has low information complexity and low number of iterations is a problem solved in the next chapter.

2.7 Conclusions

We have seen that a CIOQ switch with a speedup of just two can behave identically to an OQ switch which employs a wide variety of packet scheduling algorithms such as WFQ, strict priorities, etc. Perhaps more importantly, we show this to be true for any traffic arrival pattern and for arbitrary switch sizes.

However, while this result makes possible a significant reduction in memory bandwidth, it comes at the expense of a scheduling algorithm. The scheduling algorithm is required to configure the crossbar, operating at least twice as fast as cells can arrive. While the algorithms that we describe here are quite simple, they require a stable matching of up to at least N iterations making them unsuitable for fast switches with a large number or ports. The result does not, however, preclude algorithms that are more readily implemented at higher speed. In the next chapter, we describe how a buffered crossbar, i.e., a crossbar with a small amount of internal buffering, can reduce this scheduling complexity.

Chapter 3

Buffered Crossbars

In the previous chapter, it was shown that a CIOQ switch using a traditional unbuffered crossbar with a speedup of two can mimic any PIFO OQ switch. However, the scheduling algorithms require high communication overhead and a large number of iterations making them impractical for fast OQ switches with a large number of ports.

In this chapter, we show how a buffered crossbar, i.e., a crossbar with a small amount of internal buffering, running at a speedup of two can also behave identically to an OQ switch. The scheduler for a buffered crossbar allows inputs and outputs to schedule independently and in parallel making it much simpler and, therefore, more practical than for a traditional unbuffered crossbar. We introduce a number of scheduling algorithms which provide throughput, rate, and delay guarantees. As we shall see, buffered crossbars removes the need for high information complexity and reduces scheduling to a single iteration.

3.1 Background

Figure 3-1 shows a 3×3 buffered crossbar, with line-rate R. To prevent head-of-line blocking, the inputs maintain virtual output queues (VOQs). Fixed length packets wait in the VOQs to be transferred across the switch. Each crosspoint contains a buffer that can hold one cell. The buffer between input i and output j is denoted as B_{ij}; when the buffer holds a cell, $B_{ij} = 1$, else $B_{ij} = 0$.

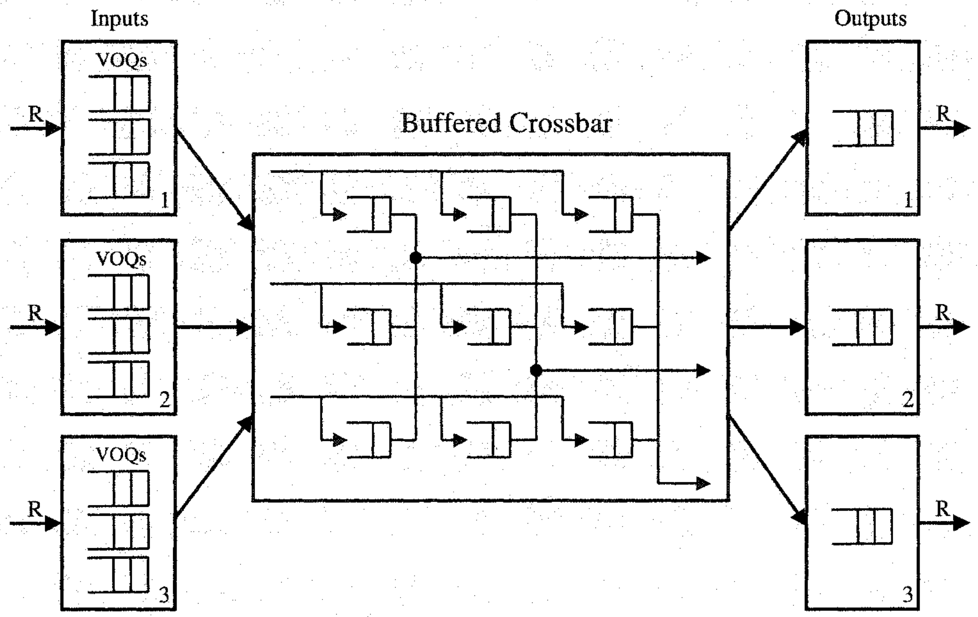

Figure 3-1: The architecture of the buffered crossbar with three ports.

Because the packets are all the same length, time is slotted, with a time slot equal to the time it takes for a cell to arrive on the external line. Internally, the switch runs faster than the external line, and the ratio between the two is the *speedup*. If the switch can remove S cells from each input and transfer S cells to each output in a time slot, then it has a speedup of S. Throughout most of this chapter we will assume that $S = 2$, and so the switch has output queues.

3.1.1 Why use Buffered Crossbars?

Buffered crossbars are interesting because they have simpler scheduling algorithms than an unbuffered crossbar. In an unbuffered crossbar, the scheduler must find a matching between inputs and outputs that does not oversubscribe either. The appeal of a buffered crossbar switch is that its scheduler is much simpler. The scheduler operates in two stages. First, each input (independently and in parallel) picks a cell to place into a crosspoint buffer. Then in the second stage each output (independently and in parallel) picks a crosspoint buffer to take a cell from. The processing can be distributed to run on each input and output, and so no longer requires a single centralized scheduler. It can be pipelined to run at high speed, making buffered crossbars appealing for high performance switches and routers. At first glance, the architecture seems similar to the DSM and SMS architectures [26, 27] where memory is sandwiched between two switching stages. However, the buffered crossbar architecture is different; it still follows the CIOQ architecture with the caveat that there is a small amount of internal buffering in the crossbar.

Researchers first noticed via simulation that buffered crossbars provide good throughput for admissible uniform traffic with simple algorithms [19, 20, 21, 28]. Simulations also indicated that, with modest speedup, a buffered crossbar can closely approximate fair queueing [22]. In [25], the authors described a mechanism to provide fair allocation and confirmed through simulations that a buffered crossbar can allocate service in a weighted max-min fair manner. Until recently, there were no analytical results on guaranteed throughput to explain or confirm the observations made by simulations.

The first analytical results came in 2001, when Javidi *et al.* proved that, with uniform traffic, a buffered crossbar can achieve 100% throughput [23]. More recently, Magill *et al.* proved that a buffered crossbar with a speedup of two can mimic a first-in first-out output queued (FIFO OQ) switch with any arrival traffic pattern [24]. Magill *et al.* also showed that a buffered crossbar with k cells per crosspoint can mimic a FIFO OQ switch with k strict priorities.

In this chapter, we describe a series of algorithms with a broad class of performance guarantees over and above FIFO and strict priority FIFO emulation. We prove that these algorithms can achieve 100% throughput, can mimic an OQ switch using a weighted round robin scheduler (which gives rate guarantees), and can also achieve delay guarantees. The main benefit of these algorithms is that each input and output makes simple scheduling decisions *independently* and *in parallel*, eliminating the need for a centralized scheduler. Our results show buffered crossbars can greatly simplify the scheduling process.

Of course, simplifying the scheduler comes at the expense of a more complicated crossbar; it now has to hold and maintain N^2 packet buffers. In the past, this would

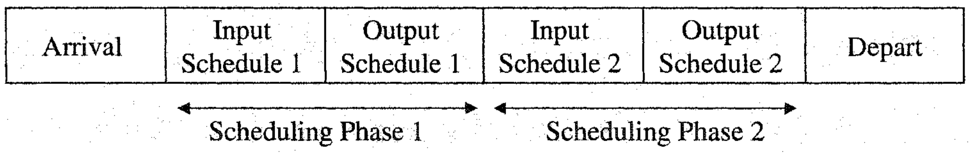

Figure 3-2: The scheduling phases for the buffered crossbar. The exact order of the phases does not matter, but we will use this order to simplify proofs.

have been prohibitively complex; the number of ports and capacity of a crossbar switch used to be limited by the N^2 crosspoints that dominated the chip area (hence the development of multi-stage switch fabrics, such as Clos, Banyan and Omega switches based on smaller crossbar elements). But nowadays, crossbar switches are limited by the number of pins required to get data on and off the chip [29]. Improvements in process technology and reductions in geometries mean that the logic required for N^2 crosspoints is small compared to the size of chip needed for N inputs and N outputs. The chips are pad-limited, with an underutilized die. A buffered crossbar can use the unused die for packet buffers. For example, we believe that in current technology, the 128×128 unbuffered crossbar switch reported in [29] could hold 128^2 cell buffers.

3.2 Achieving 100% throughput with an arbitrary scheduling algorithm

Figure 3-2 shows the scheduling phases in a buffered crossbar with a speedup of two. The two scheduling phases each consists of two parts: input scheduling, and output scheduling. In the input scheduling phase, each input (independently and in parallel) picks a cell to place into an empty crosspoint buffer. In the output scheduling phase, each output (independently and in parallel) picks a cell from a non-empty crosspoint buffer to take from. The key to creating a scheduling algorithm is determining the input and output scheduling policy which decides how inputs and outputs pick cells in the scheduling phases. We will see a number of different policies each of which provides a different scheduling algorithm. The first algorithm we will consider is the most general. In each scheduling phase, the input picks any non-empty VOQ, and the output picks any non-empty crosspoint.

We will adopt the following notation and definitions. The switch has N ports, and VOQ_{ij} holds cells at input i destined for output j. X_{ij} is the occupancy of VOQ_{ij},[1] and $Z_{ij} = X_{ij} + B_{ij}$ is the sum of the number of cells in the VOQ and the corresponding crosspoint. We will assume that all arrivals to input $i \in 1, 2, 3, ..., N$ are Bernoulli i.i.d. with rate λ_i, and are destined to each output $j \in 1, 2, 3, ...N$ with probability λ_{ij}. We will denote the arrival matrix as $A \equiv [\lambda_{ij}]$, where for all i, j,

$$\lambda_i = \sum_{j=1}^{N} \lambda_{ij}, \lambda_j = \sum_{i=1}^{N} \lambda_{ij}, \lambda_{ij} \geq 0$$

In what follows, we will show that the buffered crossbar can give 100% throughput. The result is quite strong in the sense that it holds for any arbitrary work-conserving input and output scheduling policy with a speedup of two. In other words, each input i can choose to serve any non-empty VOQ for which $B_{ij} = 0$, and each output j can choose to serve any crosspoint for which $B_{ij} = 1$.

Theorem 8 *(Sufficiency) A buffered crossbar can achieve 100% throughput with speedup two for any Bernoulli i.i.d. admissible traffic.*

Proof: We describe an intuition of the proof. The main proof appears in Appendix D.

For each VOQ_{ij}, let C_{ij} denote the sum of the cells waiting at input i and the cells waiting at all inputs destined to output j (including cells in the crosspoint for output j),

$$C_{ij} = \sum_k X_{ik} + \sum_k (X_{kj} + B_{kj}) \tag{3.1}$$

It is easy to see that when VOQ_{ij} is non-empty (i.e., $X_{ij} > 0$), then C_{ij} decreases in every scheduling phase. There are two cases:

[1]We will see later that other queueing structures are useful and that it is not necessary to place cells in VOQs.

- *Case 1:* $B_{ij} = 1$. Output j will receive one cell from the buffers destined to it and $\sum_k (X_{kj} + B_{kj})$ will decrease by one.
- *Case 2:* $B_{ij} = 0$. Input i will send one cell from its VOQs to a crosspoint, and $\sum_k X_{ik}$ will decrease by one.[2]

With $S = 2$, C_{ij} will decrease by two per time slot. When the inputs and outputs are not oversubscribed, the expected increase in C_{ij} is strictly less than two per time slot. So the expected change in C_{ij} is negative over the time slot, and this means that the expected value of C_{ij} is bounded. This in turn implies that the expected value of X_{ij} is bounded and the buffered crossbar has 100% throughput. ■

3.3 Counting Method with a Buffered Crossbar

We will now show how the buffered crossbar can mimic an OQ switch in a simple distributed manner where each input and output makes decisions independently and in parallel.

In chapter 2, a counting method is introduced to show that a CIOQ switch using a traditional unbuffered crossbar with a speedup of two can mimic a PIFO OQ switch. As a reminder, the counting method requires that in each scheduling phase, at least one of the following conditions for each cell c is satisfied: (1) cell c is transferred from the input side, (2) a cell that is ahead of cell c in its input priority list is transferred from the input side, or (3) a cell that is ahead of cell c in its output priority list is transferred to the output side.

It is proved that meeting the conditions of the counting method ensured that the slackness of a cell increased by at least one in each scheduling phase, which is essential in proving that the slackness of any cell is always non-negative. However, the scheduling algorithms require a *stable marriage algorithm* to meet the conditions of the counting method. The solution requires up to at least N iterations of the

[2]If a cell from VOQ_{ij} is sent to crosspoint B_{ij}, then $\sum_k (X_{kj} + B_{kj})$ stays the same at the end of the input scheduling phase since X_{ij} decreases by one and B_{ij} increases by one. In the output schedule, *Case 1* applies and C_{ij} will further decrease by one. As a result, if a cell from VOQ_{ij} is sent to crosspoint B_{ij}, then C_{ij} decreases by two in that scheduling phase.

stable marriage algorithm for a switch with N ports, making them too complex to implement for fast switches with large number of ports.

In order to ensure that the slackness of a cell increases by at least one in each scheduling phase for a buffered crossbar, the input and output scheduling policies must carefully be selected to guarantee that the conditions of the counting method are met. The input scheduling policy gives preference to cells based on the input priority list. Similarly, the output scheduling policy gives preference to cells based on the output priority list. Since the output priority list is ordered based on departure order, preference is given to cells with an earlier departure order.

However, the buffered crossbar has an additional requirement to meet the conditions of the counting method. The input priority list must also be arranged so cells destined to the same output are ordered based on departure order. Specifically, cells to the same output with an earlier departure order must have a higher priority. Cells to different outputs can still be ordered in any way. This requirement is necessary to ensure that, in the output scheduling phase, the cell selected has the earliest departure order of the cells stored in the input queues corresponding to the non-empty crosspoint, as can be seen in the following example.

Let cells a, b, and c all be destined to output j. Cell a is stored in input queue i_1, cell b is stored in input queue i_2, cell c is stored in crosspoint $B_{i_1 j}$, and no other cells are destined to output j at time t. The departure order is $t_a < t_b < t_c$ for cells a, b, and c respectively. In the input scheduling phase, input i_1 does not select cell a since cell c is already in the crosspoint $B_{i_1 j}$, and input i_2 selects cell b. In the output scheduling phase, cell b is selected since it has an earlier departure order than cell c. As a result, the conditions of the counting method is not met for cell a since cell b, which has a later departure order, does not have a higher priority than cell a in the output priority list. This occurred because at some point in time cell c was incorrectly given a higher priority than cell a in the input priority list. This motivates the following "Group By Virtual Output Queue" insertion policy previously described in Chapter 2.

GBVOQ Insertion Policy:

1. When a cell arrives to a non-empty VOQ, the cell is inserted in the input priority

list just behind the last cell belonging to the same VOQ. This ensures that cells destined to the same output are ordered based on departure order.

2. When a cell arrives to an empty VOQ, the cell is inserted at the head of the input priority list.

We will now prove in the following lemma that the buffered crossbar can satisfy the conditions of the counting method.

Lemma 9 *The slackness $L(c)$ of a cell c decreases by at least one in each scheduling phase.*

Proof: Let's assume that cell c belongs to VOQ_{ij}. There are two cases to consider in a scheduling phase.

- *Case 1:* If $B_{ij} = 0$, then we know that in the input scheduling phase, a cell will be transferred from input i to one of the buffers B_{i*}. If cell c is transferred to B_{ij}, then we no longer need to consider it.[3] If a different cell is transferred to its crosspoint buffer, the cell would belong to c's input thread, and $IT(c)$ will decrease by one.
- *Case 2:* If $B_{ij} = 1$, then a cell will be transferred from one of the crosspoints B_{*j} to output j in the output scheduling phase. By definition of the GBVOQ insertion policy the cell in crosspoint B_{ij} has an earlier departure order than cell c. Since the output scheduling policy selects the non-empty crosspoint that contains the cell with the earliest departure order, $OC(c)$ increases by one.

Therefore, $L(c)$ increases by at least one per scheduling phase. ■

The counting method using the GBVOQ insertion policy can be applied trivially to show that a buffered crossbar can mimic a restricted PIFO OQ switch, i.e., a PIFO OQ switch with the restriction that cells from an input/output pair depart the switch in the order they arrive. This restricted policy includes output link schedulers which are fair across all inputs, i.e., provide rate guarantees between each input/output pair.

[3]If a cell is transferred to the crosspoint then it is available for selection in the output scheduling phase and can be placed on the output line whenever necessary.

Theorem 10 *(Sufficiency) A buffered crossbar with a speedup of two can mimic the restricted PIFO OQ switch, regardless of the incoming traffic pattern.*

Proof: See Appendix E.1. ■

3.4 Rate Guarantees with a Buffered Crosbar

Our goal is to find a way for a buffered crossbar to provide a pre-determined and guaranteed rate for each flow passing through the switch. In an OQ switch, this is straightforward to do with, e.g., a weighted round-robin (WRR) scheduler. A WRR scheduler serves each flow queue in turn in round-robin order, giving service to each queue in proportion to the weight assigned to it. If a queue is empty, it is skipped and not served. It is well known that — when packets are all of equal length — WRR gives each flow a rate in proportion to its weight, and hence can give a minimum rate guarantee to each flow. Furthermore, if the arrival processes are suitably constrained (e.g., by leaky buckets), then the delay of each packet through the switch can be bounded [5].

One approach to proving that a buffered crossbar can provide rate guarantees would be to show that the buffered crossbar can mimic a PIFO OQ switch. Then, because WRR is a special case of PIFO, we can conclude that the buffered crossbar can support WRR and provide rate guarantees just as an OQ switch can. But as we will see in the next section, it is harder for a buffered crossbar to mimic a PIFO OQ switch.

So instead, we are going to start by solving an easier problem by considering a restricted PIFO OQ switch. In what follows, a flow is defined to be those cells between a specific input/output pair. We will consider a WRR OQ switch that serves these flows in WRR order and try to mimic the same behavior with a buffered crossbar.

In the restricted WRR OQ switch, each arriving cell is assigned a virtual finishing time by the WRR scheduler. Reviewing the way a WRR scheduler works, let us consider the cells at just one output. The k^{th} cell from input i is assigned virtual finishing time F_i^k, where

$$F_i^k = max\{F_i^{k-1}, V(a_i^k)\} + \frac{1}{\phi_i}.$$

a_i^k is the arrival time of the k^{th} cell from input i to that output, $V(t)$ is the virtual time (round number) at time t, and ϕ_i is the weight assigned to input i. When the output line is free, the WRR scheduler serves the cell with the smallest virtual finish time.

Similarly, in the WRR buffered crossbar, a virtual finishing time needs to be assigned to each cell so as to determine the correct departure order. The problem is that cells are buffered at both inputs and outputs (and in the crossbar). Calculating the virtual finishing time when the cell arrives would require the input to have information about the cell's output and all the cells at other inputs destined to it. This is impractical. Fortunately with our restricted definition of flows, cells are held in the input priority list in their departure order and — as we will show below — it is sufficient for the output to assign a virtual finish time only when cells reach the crosspoint buffer.

The output needs to know upon arrival of the k^{th} cell to the switch whether the $k-1^{th}$ cell (from this input to the given output) has departed. If it has departed, then the k^{th} cell is transferred to the crosspoint buffer immediately and is assigned the virtual finish time based only on the current virtual time. If it has not departed, then the k^{th} cell is not be transferred immediately or the $k-1^{th}$ cell must be in the output queue. Therefore, the k^{th} cell is assigned the virtual finish time based on the virtual finish time of the $k-1^{th}$ cell. We will formalize these cases in the proof.

Theorem 11 *(Sufficiency) A buffered crossbar can mimic an OQ switch using a weighted round-robin policy with speedup two, regardless of the incoming traffic pattern.*

Proof: See Appendix E.2. ■

A consequence of the proof is that the scheduling of cells to mimic WRR service is practical, and can be done independently and in parallel by each input and output.

An input independently generates its input priority list with only local knowledge. It picks a cell to place in the crosspoint knowing only which crosspoints are empty. An output calculates the virtual finish time of a cell when the cell arrives to the crosspoint buffer. The output just needs to know if the cell was transferred to the crosspoint immediately upon arrival which can be carried in one bit of the cell. Each output picks the cell in the crosspoint with the smallest virtual finish time.

3.5 Delay Guarantees in a Buffered Crossbar

In practice, an input/output pair of a switch carries many flows, not only one. For example, it carries TCP flows between source/destination pairs, and we might want to give each flow a different rate or delay guarantee. In order to do this, we need to relax our constraint on the definition of the flow, and determine how to assign a different rate to each flow. This is what we will do next, and we will see that it increases the complexity of the buffered crossbar and requires more speedup.

In a PIFO OQ switch, an arriving cell can be pushed into any location in the queue. It could, for example, be scheduled to depart ahead of all currently queued cells between the same input/output pair. In order to meet the conditions of the counting method, the cell in the crosspoint must have the earliest departure order of all cells stored in the input queue belonging to its input/output pair. This causes problems for the buffered crossbar switch. Imagine the situation in which a crosspoint has a cell in it, and an arriving cell has an *earlier* departure order than the cell in the crosspoint buffer. This causes what we call *"crosspoint blocking"* since the arriving cell cannot overtake the cell in the crosspoint buffer.

If each crosspoint had a cell buffer for each flow, crosspoint blocking could be avoided. However, this does not scale for a large number of flows. We now show how a buffered crossbar can overcome crosspoint blocking in a manner which is independent of the number of flows between an input/output pair.

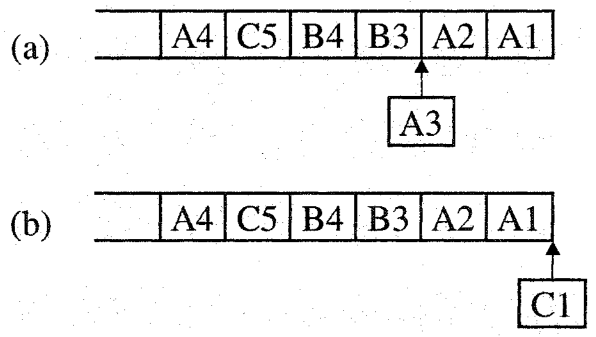

Figure 3-3: The insertion policy for achieving delay guarantees. The figure shows the priority list for a given input. The letter denotes the output destination, and the number denotes the cell's departure order for a given output. (a) Arriving cell $A3$ is inserted immediately after cell $A2$. (b) Arriving cell $C1$ is inserted at the head of the priority list since no other cell has a departure order less than cell $C1$ destined to output C.

3.5.1 Delay guarantees with speedup three

When a cell arrives to an input with an earlier departure order than the cell in the crosspoint buffer, we will swap the cell in the crosspoint with the newly arriving cell. Logically, the cell that was previously in the crosspoint is recalled to the input where it is treated like a newly arriving cell. By modifying the arrival phase to include swapping, crosspoint blocking can be avoided. This is at the expense of additional speedup to perform the swap.

The modified arrival phase requires a new insertion policy. This policy needs to meet two requirements: (1) To prevent crosspoint blocking, cells from an input/output pair must be inserted based on their departure order, and (2) The slackness of a cell must be non-negative when inserted.

The Insertion Policy: As a consequence of the first requirement, an arriving cell c destined to output port j is inserted behind all cells destined to output j with a departure order less than cell c. To satisfy the second requirement, cell c is inserted *immediately* behind the cell that departs before it (if it exists), destined to the same output. If no such cell exists, cell c is inserted at the head of the priority list. This ensures that the slackness of the cell c is non-negative.

The priority list defined by this insertion policy has the property that cells from input i to output j are ordered based on their PIFO departure order. An example is shown in Figure 3-3.[4]

Theorem 12 *(Sufficiency) A buffered crossbar can mimic a PIFO OQ switch (and hence give delay guarantees) with speedup three, regardless of the incoming traffic pattern.*

Proof: See Appendix E.3. ■

3.5.2 Delay guarantees with speedup two

We overcame crosspoint blocking by swapping the cell in a crosspoint with a newly arriving cell. This was necessary because we allowed cells to be transferred to the buffered crossbar even before they were scheduled to depart. This early transfer was the cause of crosspoint blocking, and thus required swapping. But we could eliminate the need for swapping if we avoided transferring a cell to the crosspoint until it was really ready to be transferred to the output.

For example, we could put the cell header in the crosspoint buffer, and only transfer the cell across the crossbar when chosen by the output. Scheduling would now be done in two distinct phases. First, input and output scheduling would be done based on the cell headers. Second, the cell bodies would be transferred when the output chooses a cell. We call this *header scheduling.*

However, this creates a problem. An input could receive up to N grants (one from each output) in a single output scheduling phase. Fortunately, over p consecutive phases the number of grants received by an input is bounded by $p + N - 1$. This is because an input can communicate at most one header per input scheduling phase, and there are at most N outstanding headers (one for each crosspoint) per input. On the other hand, each output grants at most one header per scheduling phase. So there are at most p grants for an output over any p consecutive scheduling phases.

[4]This insertion policy should be contrasted with the CCF insertion policy, which does not maintain cells from input i to output j in the correct departure order. CCF does not need to maintain this ordering because the stable marriage problem considers all cells queued at the inputs when scheduling.

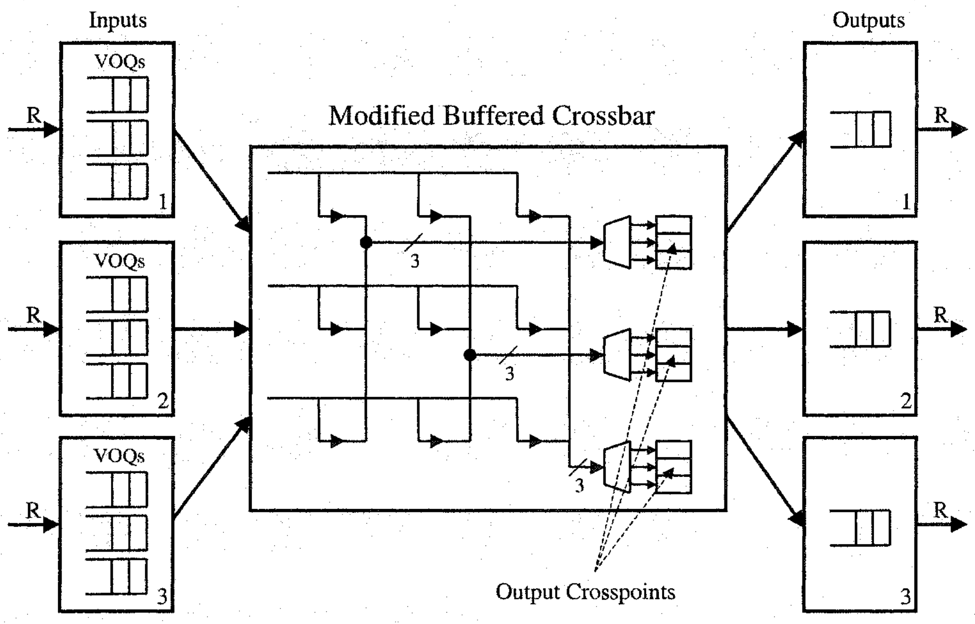

Figure 3-4: The architecture of a modified buffered crossbar with three ports.

Since inputs send at most one cell per scheduling phase and an input can receive up to $p+N-1$ grants in p consecutive scheduling phases, the cell for a header granted in phase x might not be transferred until phase $x+N-1$. With only one cell per crosspoint, a cell can prevent an input from sending another cell to the same output for another $N-1$ scheduling phases. We, therefore, modify the buffered crossbar to have N cells of buffering per output as shown in Figure 3-4. There are still N^2 buffers in the crossbar, but buffers are dedicated to outputs rather than input/output pairs.

Theorem 13 *(Sufficiency) With speedup of two and N cells of buffering per output, a buffered crossbar can mimic a PIFO-OQ switch with a fixed delay of $N/2$ time slots.*

Proof: See Appendix F. ■

The result comes at the expense of a more complicated buffering scheme in the crossbar and requires N cells buffering per output. Since these N cells can arrive in the same scheduling phase, there is an additional implementation complexity. This can be eliminated by modifying the buffered crossbar so it has N cells for each B_{ij} for a total of N^3 cells. In the latter case, no more than one cell can arrive to a crosspoint in each scheduling phase. While requiring more storage, it will also mimic a PIFO

OQ switch with a fixed delay of $N/2$ time slots with speedup of two. This might be practical for small values of N. In both modified buffered crossbar architectures, the number of crosspoints is independent of the number of flows in the switch.

3.6 Conclusions

It is hard to scale crossbar-based routers because the scheduler for a crossbar must resolve the input and output constraints simultaneously. Whereas centralized schedulers get very complicated, the scheduler for a buffered crossbar allows inputs and outputs to make decisions independently and in parallel. With speedup of two and scheduling algorithms which are distributed and easy to implement, buffered crossbars provide throughput, rate, and delay guarantees.

Although the buffered crossbar is more complex than an unbuffered crossbar, the bandwidth and pin count are the same, the CIOQ architecture is maintained, and no memory needs to run faster than twice the line-rate. This provides a simple path to scale crossbar based routers. In the concluding chapter, we will discuss how a buffered crossbar can change the way crossbar-based routers are built.

Chapter 4

Conclusions

This thesis describes a practical architecture for a router that can achieve the same throughput, bandwidth, and delay guarantees as an ideal OQ router. We showed how a CIOQ switch using a traditional crossbar can mimic an OQ switch. However, using a traditional crossbar leads to a design which is impractical for implementation. Fortunately, a router using a buffered crossbar can also provide the same guarantees as an ideal router. A buffered crossbar simplifies the scheduling process and provides a practical way to build routers with performance guarantees. In this chapter, we will suggest a design of a high performance router which incorporates the buffered crossbar. We will then conclude the thesis by discussing future research on buffered crossbars.

4.1 Router Design Using Buffered Crossbars

As described in the Introduction, a typical high performance router usually consists of two elements. These include the linecard and the switch fabric. The linecards are able to send cells to other linecards via the switch fabric. We will describe how a cell traverses through our suggested router in order to explain how to incorporate the buffered crossbar technology.

There are a number of advantages a buffered crossbar can provide. The main benefit is the ability to make input/output scheduling decisions with very little complexity. This simplicity drastically increases the frequency at which decisions can be made.

Another advantage is to remove centralized scheduling and allow the scheduling logic to be distributed to the linecards.

However, distributed scheduling requires that the buffered crossbar convey to each linecard empty/full status for each corresponding crosspoint. As a result, each linecard receives N status bits for input scheduling and N status bits for output scheduling for a total of $2N$ status bits per time slot where N represents the number of ports. Unfortunately, many router vendors require their boxes to be scalable in terms of the number of ports.

Another disadvantage to distributed scheduling is the need for crosspoints with storage for more cells. So far in this thesis, we assumed that the delay to send a cell or receive crosspoint status information between the linecard and the buffered crossbar was negligible. In practice, the round trip time between the linecard and buffered crossbar can correspond to many cells. To account for this delay, each crosspoint requires cell storage equivalent to the round trip time.

The router architecture we are proposing is not distributed and requires a centralized scheduler. On the other hand, the design still takes advantage of how the buffered crossbar simplifies the scheduling process. Figure 4-1 shows an ingress linecard, a centralized scheduler, a buffered crossbar, and an egress linecard. The centralized scheduler performs input and output scheduling and holds the occupancy state of the cells stored in the ingress linecards. We assume that the buffered crossbar can hold one cell per crosspoint. The life of a cell is described below:

1) A cell arrives to the ingress linecard and is destined to a specific outgoing port. The cell is stored in the packet buffer of the ingress linecard.

2) The ingress linecard then makes a request for this cell to traverse the buffered crossbar. The request is sent to the centralized scheduler.

3) The centralized scheduler receives the request and keeps track of the cells currently stored in each ingress linecard. This information is conveyed to the input scheduler block. Sometime later, the input scheduler dedicated for that ingress linecard makes a grant for the cell to be transferred to the buffered crossbar. The grant is conveyed to the output scheduler block. In addition, the grant for that cell is sent to the ingress linecard.

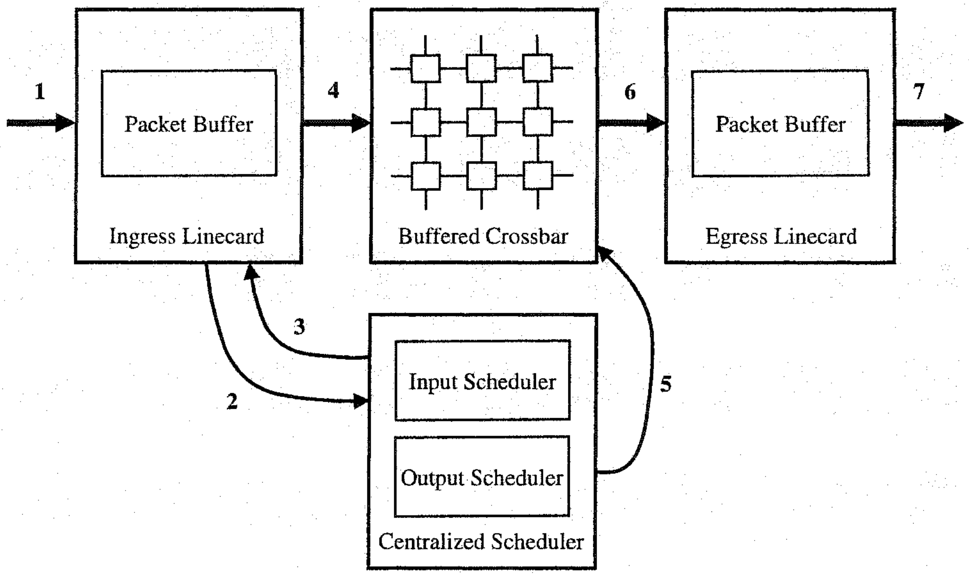

Figure 4-1: Datapath of a packet through a router using a buffered crossbar.

4) The ingress linecard receives the grant and sends the corresponding cell to the buffered crossbar. The cell is stored in the corresponding crosspoint buffer.

5) Sometime later, the output scheduler dedicated to the egress linecard schedules the cell to be transferred to the egress linecard. This decision is conveyed to the buffered crossbar.

6) The buffered crossbar sends the cell to the corresponding egress linecard. The cell is stored in the packet buffer of the egress linecard.

7) Sometime later, the cell is placed on the outgoing line.

In this suggested design, the communication between a linecard and the switch fabric is limited to one request, one grant, and one sending cell and one receiving cell per scheduling phase. The centralized scheduler stores requests for the cells currently waiting in the ingress linecard. Each input scheduler independently determines which cell to send to the empty crosspoints. Based on the crosspoint status, each output scheduler independently determines which cell to transfer to the egress linecards. Even though the input and output schedulers are located in a centralized location, the decisions are made independently and in parallel.

4.2 Future Research

We noted that technology today is capable of allowing one cell per crosspoint when the number of ports was less than one hundred. As technology progresses, the size of memories that can fit on chip continues to grow. So, one question to ask is what other features are possible if a buffered crossbar can store a number of cells per crosspoint. In the following sections, we conjecture that there are a number of other possible guarantees a buffered crossbar with larger crosspoints can provide.

4.2.1 Larger Buffers per Crosspoint

In Section 3.4, a buffered crossbar with one cell per crosspoint was shown to mimic an OQ switch using a weighted round robin scheduler. A consequence of this result is that the scheduling of cells is practical, and can be done independently and in parallel by each input and output. However, this result made the assumption that cells from an input/output pair depart the switch in the order they arrive. Fortunately, if there are k cells per crosspoint, then the result can be trivially applied to include switches where there are at most k flows between an input/output pair.

These additional flows could be used to provide better quality of service or the support for subports. This feature comes at a cost of statically allocating a number of cells per crosspoint. Another possible direction for future research is to explore how cells could dynamically be allocated across a number of crosspoints. It would be interesting to investigate how the sharing of cells would influence the scheduling policies. We believe that throughput, rate, and delay guarantees can be provided with more efficient use of the crosspoint buffers.

4.2.2 100% Throughput with Larger Buffers

Another possible use of larger buffers could be to reduce the speedup required to provide throughput guarantees. For uniform traffic, Javidi *et al.* proved that a buffered crossbar with one cell per crosspoint can achieve 100% throughput [23]. In Section 3.2, a buffered crossbar with one cell per crosspoint running at a speedup of two was shown to provide 100% throughput. We believe that a buffered crossbar with N cells per crosspoint, where N is the number of ports in the switch, can achieve 100% throughput at a speedup less than two. Future research could determine the speedup and combination of the input and output scheduling policies required to provide throughput guarantees.

4.2.3 Variable Length Packets

This thesis primarily focused on fixed sized cells. However, most packets in the Internet are variable length. If a crosspoint in a buffered crossbar could be the size of a packet, we believe that throughput, rate, and delay guarantees can be provided for packets. Many difficult problems can be avoided if routers only had to process packets.

Segmentation and reassembly has been a required function to convert packets into cells for cell switching. Because cells are a fixed size and packets are segmented to fit these cells, many cells are padded with useless data. This has been referred to as the 65-byte problem where the size of a cell is 64 bytes. The first cell holds 64 bytes of valid data. However, the second cell only holds 1 byte of valid data. To overcome this inefficiency, many router vendors run their linecards at twice the rate.

The algorithms to support variable length packets for the buffered crossbar would be similar. The only difference is that input and output schedulers would make decisions on packets instead of cells. This also allows a system to be designed where linecards can run asynchronously. Future research could confirm that performance guarantees for variable length packets using buffered crossbars is achievable.

Appendix A

Necessity and Sufficiency of a Speedup of $2 - 1/N$

A.1 The Necessity of a Speedup of $2 - 1/N$

In this section, we show a lower bound of $2 - 1/N$ on the speedup of any $N \times N$ CIOQ switch that mimics OQ switching, even when the OQ switch uses FIFO. Given the sufficiency of a speedup of two presented in Section 2.4, the algorithms that we have presented in this paper are almost optimal. In fact, the difference of $1/N$ can be ignored for all practical purposes.

Since a speedup between one and two represents a non-integral distribution of phases, we first describe how scheduling phases are distributed. A speedup of $2-1/N$ corresponds to having one *truncated* time slot out of every N time slots. The truncated time slot has only one scheduling phase, whereas the other $N-1$ time slots have two scheduling phases each. In Figure A-1, we show the difference between one-phased and two-phased time slots. For the purposes of our lower bound, we need to assume that the scheduling algorithm does not know in advance whether a time slot is truncated.

Recall from Section 2.3 that a cell is represented as P-t, where P represents which output port the cell is destined to, and t represents the departure time for the cell. For example, the cell C-7 must be scheduled for port C before the end of time slot 7.

The input traffic pattern that provides the lower bound for an $N \times N$ CIOQ switch is given below. The traffic pattern spans several time slots; the last of which

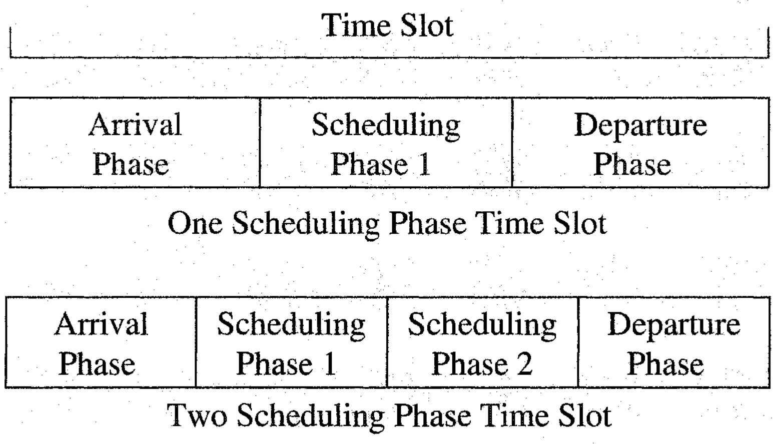

Figure A-1: One scheduling phase and two scheduling phase time slots.

is truncated.

1. In the first time slot, all input ports receive cells destined for the same output port, P_1.

2. In the second time slot, the input port that had the lowest departure time in the previous time slot does not receive any more cells. In addition, the rest of the input ports receive cells destined for the same output port, P_2.

3. In the i^{th} time slot, the input ports that had the lowest departure time in each of the $i - 1$ previous time slots do not receive any more cells. In addition, the rest of the input ports must receive cells destined for the same output port, P_i.

We can repeat the above traffic pattern as often as required to create arbitrarily long traffic patterns. In Figure A-2, we show the above sequence of cells for a 4×4 switch. The arrival events to the OQ switch are depicted on the left, and the departure events are on the right. For simplicity, we present the proof of our lower bound on this 4×4 switch. The proof immediately extends to $N \times N$ for $N \geq 4$.

Figure A-3 shows the only possible schedule for transferring these cells across in seven phases. Of the four time slots, the last one is truncated, giving a total of seven phases. Cell A-1 must leave the input side during the first phase since the CIOQ switch does not know whether the first time slot is truncated. Similarly, cells B-2, C-3, and D-4 must leave during the third, fifth, and seventh phases respectively (see Figure A-3(a)). Cell A-2 must leave the input side by the end of the third phase, but

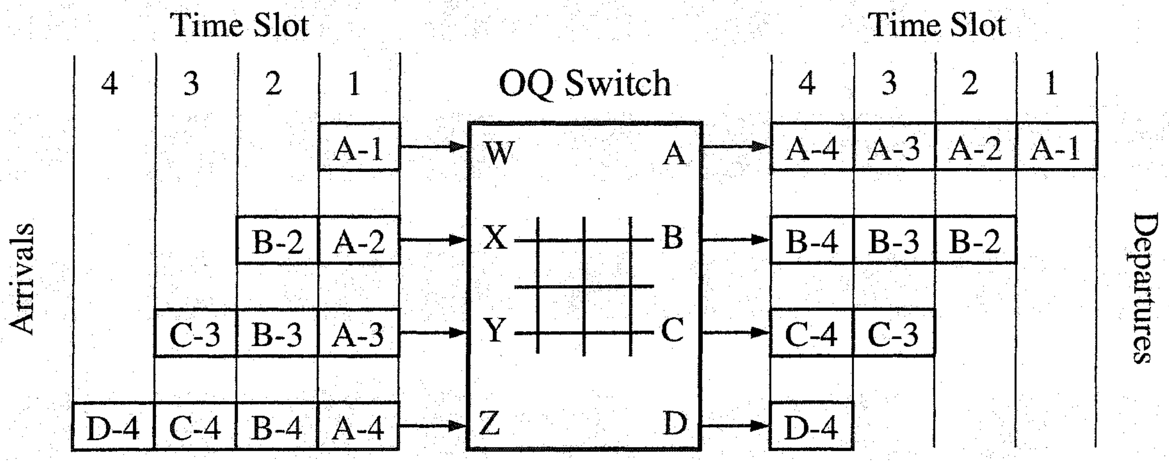

Figure A-2: Lower bound input traffic pattern for a 4 × 4 switch.

Phase	PA	PB	PC	PD
1	**A-1**			
2				
3		**B-2**		
4				
5			**C-3**	
6				
5				**D-4**

(a)

Phase	PA	PB	PC	PD
1	A-1			
2	**A-2**			
3		B-2		
4		**B-3**		
5			C-3	
6			**C-4**	
5				D-4

(b)

Phase	PA	PB	PC	PD
1	A-1			
2	A-2			
3	**A-3**	B-2		
4		B-3		
5		**B-4**	C-3	
6			C-4	
5				D-4

(c)

Phase	PA	PB	PC	PD
1	A-1			
2	A-2			
3	A-3	B-2		
4	**A-4**	B-3		
5		B-4	C-3	
6			C-4	
5				D-4

(d)

Figure A-3: Scheduling order for the lower bound input traffic pattern in Figure A-2.

it cannot leave during the first or the third phase because of contention. Therefore, it must depart during the second phase. Similarly, cells B-3 and C-4 must depart during the fourth and sixth phases respectively (see Figure A-3(b)). Continuing this elimination process (Figure A-3(c), (d)), there is only one possible scheduling order. For this input traffic pattern, the switch needs all seven phases in four time slots which corresponds to a minimum speedup of 7/4 (or $2 - 1/4$).

Theorem 14 *A minimum speedup of* $2 - 1/N$ *is necessary for an* $N \times N$ *CIOQ switch operating under* ***any*** *algorithm which is not allowed to consider the number of scheduling phases in a time slot.*

The proof of Theorem 14 is a straight-forward extension of the 4×4 CIOQ switch example.

A.2 The Sufficiency of a Speedup of $2 - 1/N$ to Mimic a FIFO OQ Switch

We now show that it is possible to mimic a FIFO OQ switch using a speedup of $2 - 1/N$. Specifically, we show that identical behavior can be achieved by a CIOQ switch which follows the general framework described in Section 2.4. It uses a scheme that we call Last In Highest Priority (LIHP) to determine input priorities for incoming cells. As the name suggests, LIHP places a newly arriving cell right at the *front* of the input priority list. The analysis in this section borrows heavily from ideas described in Section 2.5.

In this section, we use a slightly different time slot structure. A "normal" time slot has an arrival phase followed by two scheduling phases and then a departure phase, whereas a "truncated" time slot has an arrival phase, a scheduling phase, and then a departure phase. Since the speedup is $2 - 1/N$, we assume that there are $N - 1$ normal time slots between two truncated time slots. The CIOQ switch does not need to know which time slots are truncated.

At any time instant and for any cell c, let $NTS(c)$ denote the number of truncated time slots between now and the time when this cell leaves the OQ switch, inclusive.

Recall from Section 2.3 that $L(c) = OC(c) - IT(c)$ is the slackness of cell c, where $OC(c)$ and $IT(c)$ refer to the output cushion and input thread of the cell, respectively.

The following lemma holds for CIOQ switches that use LIHP and operate at a speedup of $2 - 1/N$.

Lemma 15 *If the OQ switch being mimicked is FIFO, then $L(c) \geq NTS(c)$ after the first scheduling phase and just before the arrival phase, for all cells c waiting on the input side.*

Proof: Suppose the lemma has been true till the beginning of time slot $t - 1$. We prove that the lemma holds at the end of the first scheduling phase and at the end of the departure phase in time slot t.

We first consider the end of the first scheduling phase. Cells which were already present on the input side at the beginning of time t satisfy $L \geq NTS$, as NTS does not change (a property of FIFO – the departure time of a cell from the OQ switch gets fixed upon arrival, and does not change), and L can only go up (see Lemma 3 for an explanation of why L can not decrease) during the arrival and the scheduling phases. Now consider a cell c which arrives during time slot t. Let $k = NTS(c)$. Since the slackness of a cell is at least zero upon arrival (remember that the input thread of an arriving cell is zero in LIHP), the slackness at the end of the first scheduling phase must be at least one. Therefore, c trivially satisfies the lemma if $k \leq 1$. Suppose $k \geq 1$. At most N cells could have arrived during the current time slot, and therefore, there must have been a cell d in the system with a NTS of $k - 1$ and with the same output port as c at the *beginning* of time t (this is where we use the fact that the truncated time slots are spaced at least N apart). If d is waiting on the input side, then $OC(d) \geq L(d) \geq k - 1$. Since the OQ switch is FIFO, $OC(c) \geq OC(d)$. But the input thread of the arriving cell c must be zero. Hence, the slackness of c is at least $k - 1$ after the arrival phase, and consequently, at least k after the first scheduling phase. The case where d is waiting at the output side is similar, and we omit the details.

Now concentrate on the end of time slot t. If this time slot turns out to be normal, then the slackness of any cell does not decrease during the second scheduling phase and the departure phase. Else, the slackness of any cell can go down by at most one. However, the NTS value goes down by one for *all* cells in the system, and the lemma continues to hold. ■

Theorem 16 *A speedup of* $2 - 1/N$ *suffices for a CIOQ switch that uses LIHP to mimic a FIFO OQ switch.*

Proof: Suppose it is time for cell c to leave the OQ switch, and suppose that the CIOQ switch has successfully mimicked a FIFO OQ switch so far. Clearly, $OC(c)$ must be zero. If c has already crossed over to the output side, then we are done. So suppose c is still queued at its input port. If the current time slot were truncated, then $L(c)$ would be at least one (Lemma 15). But then the input thread would be negative, which is not possible. Therefore, the current time slot has two scheduling phases. Invoking Lemma 15 again, $L(c)$ must be at least zero after the first scheduling phase. Since $OC(c)$ is zero, the input thread of c must be zero too. Cell c, therefore, is at the front of both its input and its output priority lists, and will cross the switch in the second scheduling phase just before the departure phase. This completes the proof of the theorem. ■

Appendix B

Acyclic Dependency Graph

In this appendix, we will prove Lemma 6 which states that the dependency graph for a scheduling phase is acyclic when CCF is combined with the DTC strategy.

Proof: The proof is by contradiction. Assume there does exist a cycle in the dependency graph on active cells. Pick a smallest cycle in this graph. If there is an edge from cell a to cell b, then b must be ahead of a either in the input queue ordering or in the output queue ordering. We call the edge an "input" edge in the former case and an "output" edge in the latter; ambiguities are resolved arbitrarily. The smallest cycle must have alternating input and output edges because two successive input or output edges could be collapsed into one resulting in a smaller cycle. If there is an output edge from a to b, then the output cushion of b is at most as large as that of a. But a and b are both active, and the input thread of an active cell must equal its output cushion. Therefore, the input thread of b is no larger than the input thread of a. Also, if there is an input edge from a to b, then the input thread of b must be strictly smaller than that of a; that is, a appears in b's input thread. The smallest cycle must have at least two edges as there can be no self loops in the dependency graph. Consequently, the cycle must contain at least one input edge. But this implies that as we traverse the cycle once the input thread of the cell where we start must be larger than the input thread of the cell where we end. Since we start and end at the same cell as we traverse a cycle, this implies that the input thread of this cell must be less than itself. This is clearly impossible. Hence our assumption that there exists a cycle in the graph cannot be true, and the lemma is proved. ■

Appendix C

Non-Negative Slackness with GBVOQ

In this appendix, we will prove Lemma 7 which states that the slackness of a newly arriving cell is non-negative when using the GBVOQ algorithm.

Proof: Consider any cell a that is inserted with a slackness of $L(a)$. Following the arrival phase, $L(a)$ increases by at least one in each of the two scheduling phases. And in the departure phase, $L(a)$ will decrease by one. Therefore, at the end of the time slot, $L(a)$ increases by at least one. For example, if arriving cell a, is inserted with a slackness of zero, then at the end of the time slot, the slackness of cell a will be at least one.

From Lemma 3 and the claim that the slackness of an arriving cell will increase by one at the end of the time slot relative to the slackness of the cell when it arrived, we know that if the slackness of a cell is less than one, then its slackness must have been negative when the cell was inserted. Let t be the first time that an arriving cell is inserted with negative slackness. Consider two cases.

- **Case 1**: If cell c was inserted at the head of the priority list, $IT(c)$ is zero. Since the output cushion is defined as a non-negative value, the slackness of the cell is non-negative when inserted which contradicts our assumption.

- **Case 2**: If cell c was not inserted at the head of the priority list, cell c must be inserted immediately behind another cell, c', destined to the same output as cell c. Since c' was inserted before time t, it must have been inserted with non-negative slackness. At the end of the time slot cell c' was inserted, its slackness increased by one. From Lemma 3, the slackness of cell c' is still at least one at time t. But since $IT(c) = IT(c') + 1$, and $OC(c) = OC(c')$, then $L(c) = L(c') - 1 \geq 0$. So the slackness of the cell c must also be non-negative when inserted which again contradicts our assumption.

■

Appendix D

Achieving 100% Throughput

In this appendix, we will prove that a buffered crossbar with a speedup of two using arbitrary input and output scheduling algorithms achieves 100% throughput.

Lemma 17 *Consider a system of queues whose evolution is described by a discrete time markov chain (DTMC) which is aperiodic and irreducible with state vector* $Y_n \in \mathbb{N}^M$*. Suppose that a lower bounded, non-negative function* $F(Y_n)$*, called Lyapunov function,* $F : \mathbb{N}^M \rightarrow \mathbb{R}$ *exists such that* $\forall Y_n, E[F(Y_{n+1})|Y_n] < \infty$*. Suppose also that there exist* $\gamma \in \mathbb{R}^+$ *and* $C \in \mathbb{R}^+$*, such that* $\forall ||Y_n|| > C$*,*

$$E[F(Y_{n+1}) - F(Y_n)|Y_n] < -\gamma, \tag{D.1}$$

then all states of the DTMC are positive recurrent and for every $\epsilon > 0$*, there exists* $B > 0$ *such that* $\lim_{n\rightarrow\infty} Pr\{\sum_{i,j} X_{i,j}(n) > B\} < \epsilon$*.*

Proof: This is a straightforward extension of Foster's criteria and follows from [30, 31, 32, 33]. ■

We will use the above lemma in proving Theorem 8.

Theorem 8 *Under an arbitrary scheduling algorithm, the buffered crossbar gives 100% throughput with speedup of two.*

Proof: In the rest of the proof we will assume that all indices i, j, k vary from $1, 2, ..N$. Denote the occupancy of VOQ_{ij} at time n by $X_{ij}(n)$. Also, let Z_{ij} denote the combined occupancy of the VOQ_{ij} and the crosspoint B_{ij} at time n. By definition, $Z_{ij}(n) = X_{ij}(n) + B_{ij}(n)$.

Define,

$$f_1(n) = \sum_{i,j} X_{ij}(n)\Big(\sum_k X_{ik}(n)\Big) \tag{D.2}$$

$$f_2(n) = \sum_{i,j} Z_{ij}(n)\Big(\sum_k Z_{kj}(n)\Big) \tag{D.3}$$

$$F(n) = f_1(n) + f_2(n) \tag{D.4}$$

Observe that from (D.2)

$$\begin{aligned} f_1(n) &= \sum_{i,j} X_{ij}(n)\Big(\sum_k X_{ik}(n)\Big) \\ &= \sum_{i,j,k} X_{ij}(n)X_{ik}(n) \end{aligned}$$

Denote $D_{ij}(n) = 1$ if a cell departs from VOQ_{ij} at time n and zero otherwise. Also, let $A_{ij}(n) = 1$ if a cell arrives to VOQ_{ij} and zero otherwise. Then, $X_{ij}(n+1) = X_{ij}(n) + A_{ij}(n) - D_{ij}(n)$. Henceforth, we will drop the time n from the symbol for $D_{ij}(n)$ and $A_{ij}(n)$, and refer to them as D_{ij} and A_{ij} respectively, since in the rest of the proof, we will only be concerned with the arrivals and departures of cells at time n.

Therefore, $[f_1(n+1) - f_1(n)]$

$$= \sum_{i,j,k} [X_{ij}(n+1)X_{ik}(n+1) - X_{ij}(n)X_{ik}(n)]$$

Then we get $[f_1(n+1) - f_1(n)]$

$$\begin{aligned}
&= \sum_{i,j,k} (X_{ij}(n) + A_{ij} - D_{ij})(X_{ik}(n) + A_{ik} - D_{ik}) - \\
&\qquad X_{ij}(n)X_{ik}(n) \\
&= \sum_{i,j,k} (A_{ij} - D_{ij})X_{ik}(n) + (A_{ik} - D_{ik})X_{ij}(n) + \\
&\qquad (A_{ij} - D_{ij})(A_{ik} - D_{ik}) \\
&= \sum_{i,j,k} 2(A_{ik} - D_{ik})X_{ij}(n) + (A_{ij} - D_{ij})(A_{ik} - D_{ik})
\end{aligned}$$

Since $|A_{ij} - D_{ij}| \leq 1$ and similarly $|A_{ik} - D_{ik}| \leq 1$, we get[1]

$$E[f_1(n+1) - f_1(n)] \leq N^3 + \sum_{i,j,k} 2E[A_{ik} - D_{ik}]X_{ij}(n) \tag{D.5}$$

Denote $E_{ij}(n) = 1$ if a cell departs from the combined queue of VOQ_{ij} and the crosspoint B_{ij}, and zero otherwise. Note that $E_{ij}(n) = 1$ only when a cell departs from the crosspoint B_{ij} to the output at time n, since all departures to the output must occur from the crosspoint. Also recall that the arrival rate to the combined queue, VOQ_{ij} and B_{ij}, is the same as the arrival rate to VOQ_{ij}. So we can write $Z_{ij}(n+1) = Z_{ij}(n) + A_{ij}(n) - E_{ij}(n)$. Again we will drop the time n from the symbol for $E_{ij}(n)$ and $A_{ij}(n)$, and refer to them as E_{ij} and A_{ij} respectively.

Then, similar to the derivation in (D.5), we can derive using (D.3),

$$E[f_2(n+1) - f_2(n)] \leq N^3 + \sum_{i,j,k} 2E[A_{kj} - D_{kj}]Z_{ij}(n) \tag{D.6}$$

[1]This is in fact the conditional expectation given knowledge of the state of all queues and crosspoints at time n. For simplicity in the rest of the proof (since we only use the conditional expectation), we will drop the conditional expectation sign and simply use the symbol for expectation as its meaning is clear.

So from (D.5) and (D.6), $E[F(n+1) - F(n)]$

$$\begin{aligned}
&\leq 2N^3 + 2\sum_{i,j,k}\Big(E[A_{ik} - D_{ik}]X_{ij}(n) \\
&\qquad + E[A_{kj} - E_{kj}]Z_{ij}(n)\Big) \\
&= 2N^3 + 2\sum_{i,j}\Big(X_{ij}(n)\sum_k E[A_{ik} - D_{ik}] \\
&\qquad + Z_{ij}(n)\sum_k E[A_{kj} - E_{kj}]\Big)
\end{aligned}$$

Re-substituting $Z_{ij} = X_{ij} + B_{ij}$, we get $E[f(n+1) - f(n)]$,

$$\begin{aligned}
&\leq 2N^3 + 2\sum_{i,j}\Big(X_{ij}(n)\sum_k E[A_{ik} - D_{ik}] \\
&\qquad + \big(X_{ij}(n) + B_{ij}(n)\big)\sum_k E[A_{kj} - E_{kj}]\Big) \\
&= 2N^3 + 2\sum_{i,j}\Big(X_{ij}(n)\sum_k E[A_{ik} - D_{ik} + A_{kj} - E_{kj}] \\
&\qquad + B_{ij}(n)\sum_k E[A_{kj} - E_{kj}]\Big)
\end{aligned}$$

We can substitute $R_{ij} = \sum_k E[A_{ik} - D_{ik} + A_{kj} - E_{kj}]$ and $S_j = \sum_k E[A_{kj} - E_{kj}]$ and re-write this as,

$$E[F(n+1) - F(n)] \leq 2N^3 + 2\sum_{i,j}\Big(X_{ij}(n)R_{ij} + B_{ij}(n)S_j\Big) \tag{D.7}$$

But, we also have from equation 3.1,

$$E[C_{ij}(n+1) - C_{ij}(n)] \equiv R_{ij} \tag{D.8}$$

$$E[\sum_k\Big(Z_{kj}(n+1) - Z_{kj}(n)\Big)] \equiv S_j \tag{D.9}$$

In Section 3.2, it was shown that for a buffered crossbar with speedup of two, R_{ij} is strictly negative when $X_{ij}(n) > 0$ and the traffic is admissible. So the first product term inside the summation sign in equation (D.7)

$$X_{ij}(n)R_{ij} \leq 0 \tag{D.10}$$

Similarly, if the traffic is admissible, then $\sum_k E[A_{kj}] < 1$. Also, when $B_{ij}(n) = 1$, then from (3.1) and *case 1* of theorem 8 in section 3.2, we know that the output j will receive at least one cell and so at least one cell must have departed one of the crosspoints destined to output j at time n. And so when the traffic is admissible and $B_{ij}(n) = 1$, then $S_j < 0$. This implies that the second product term inside the summation sign in equation (D.7),

$$B_{ij}(n)S_j \leq 0 \tag{D.11}$$

In both cases, $X_{ij}(n)R_{ij}$ and $B_{ij}(n)S_j$ are equal to zero only if $X_{ij} = 0$ and $B_{ij} = 0$ respectively. Now we want to use Lemma 17 and show that the whole right hand side of equation (D.7) is strictly negative. All that needs to be done is to ensure that one of the *VOQs* X_{ij} in the summation in equation (D.7) is large enough so that $2X_{ij}(n)R_{ij}$ can negate the positive constant $2N^3$.

In order to show this, let, $\lambda_{max} = \max(\sum_k \lambda_{ik}, \sum_k \lambda_{kj}), i, j \in (1, 2, ..N)$. Choose any $\gamma' > 0$, and let

$$F \equiv \sum_{ijk} X_{ij}X_{ik} + Z_{ij}Z_{kj} >$$
$$N^3\left[\left(\frac{(1+\gamma')N^3}{1-\lambda_{max}}\right)^2 + \left(1 + \frac{(1+\gamma')N^3}{1-\lambda_{max}}\right)^2\right] \equiv C$$

where, C corresponds to the constant in Lemma 17. Recall that $Z_{ij} \leq X_{ij} + 1$. Then the above inequality can only be satisfied if there exists X_{ij} such that:

$$X_{ij} > \frac{(1+\gamma')N^3}{1-\lambda_{max}}$$

As shown in section 3.2, when $X_{ij} > 0$,

$$R_{ij} \leq -(2 - 2\lambda_{max}) < -(1 - \lambda_{max})$$

Therefore, we have

$$X_{ij}R_{ij} < -(1 + \gamma')N^3$$

If we substitute this in equation D.7, then for all n such that $F(n) > B$,

$$E[F(n+1) - F(n)] < -2\gamma' N^3$$

Let γ correspond to the variable in Lemma 17 and set $\gamma = 2\gamma' N^3$. Also it is easy to see that,

$$E[F(n+1)|F(n)] < \infty$$

From Lemma 17, for every $\epsilon > 0$, there exists $B > 0$ such that $\lim_{n\to\infty} Pr\{\sum_{i,j} X_{i,j}(n) > B\} < \epsilon$. From definition 3, the scheduling algorithm gives 100% throughput. ■

Appendix E

Mimicking an OQ Switch

E.1 Mimicking a Restricted PIFO OQ Switch

In this section, we will prove that a buffered crossbar with a speedup of two can mimic a restricted PIFO OQ switch. Before we prove Theorem 10, we will need the following lemmas.

Lemma 9 *The slackness $L(c)$ of a cell c waiting on the input side is non-decreasing from time slot to time slot.*

Proof: Let $L(c)$ be the slackness of cell c which belongs to VOQ_{ij} at the beginning of a time slot. During the arrival phase, $IT(c)$ can increase by at most one because an arriving cell might be inserted ahead of c in its input priority list. During the departure phase, $OC(c)$ will decrease by at most one. So, $L(c)$ can decrease by at most two in a single time slot.

From Lemma 9, $L(c)$ increases by at least one per scheduling phase. With two scheduling phases per time slot, $L(c)$ increases by at least two. Taking into account arrivals, departures, and both scheduling phases, $L(c)$ cannot decrease from time slot to time slot. ■

Lemma 10 *The slackness $L(c)$ of a newly arriving cell c is non-negative.*

Proof: Consider any cell x that is inserted with a slackness of $L(x)$. Following the arrival phase, $L(x)$ increases by at least one in each of the two scheduling phases.

In the departure phase, $L(x)$ will decrease by one. Therefore, at the end of the time slot, $L(x)$ increases by at least one. For example, if arriving cell x is inserted with a slackness of zero, then the slackness of cell x will be at least one at the end of the time slot.

From Lemma 9 and the fact that the slackness of an arriving cell will increase by one at the end of the time slot relative to the slackness of the cell when it arrived, we know that if the slackness of a cell is less than one, then its slackness must have been negative when the cell was inserted. Let t be the first time that an arriving cell is inserted with negative slackness. Consider two cases:

- **Case 1**: If cell c was inserted at the head of the priority list, $IT(c)$ is zero. Since the output cushion is defined as a non-negative value, the slackness of the cell is non-negative when inserted which contradicts our assumption.

- **Case 2**: If cell c was not inserted at the head of the priority list, cell c must be inserted immediately behind another cell, c', destined to the same output as cell c. Since c' was inserted before time t, it must have been inserted with non-negative slackness. At the end of the time slot cell c' was inserted, its slackness increased by one. From Lemma 9, the slackness of cell c' is still at least one at time t. But since $IT(c) = IT(c') + 1$, and $OC(c) = OC(c')$, then $L(c) = L(c') - 1 \geq 0$. So the slackness of the cell c must also be non-negative when inserted which again contradicts our assumption.

■

Theorem 2 *(Sufficiency) A buffered crossbar with a speedup of two can mimic the restricted PIFO-OQ switch regardless of the incoming traffic pattern.*

Proof: Suppose that the CIOQ switch has successfully mimicked the OQ switch up until time slot $t-1$. Consider the beginning of time slot t. We must show that any cell reaching its departure time is either: (1) already at the output side of the switch or (2) will be transferred to the output during time slot t.

From Lemma 9 and Lemma 10, we know that a cell always has a non-negative slackness. Therefore, when a cell reaches its departure time (i.e., its output cushion has reached zero), its input thread must also equal zero. This means either: (1) that the cell is already at its output and may depart on time, (2) that the cell is in the crosspoint buffer, or (3) that the cell is simultaneously at the head of its input priority list (because its input thread is zero) and has the earliest departure time (because it has reached its departure time). In case (3), the input scheduling phase is guaranteed to transfer the cell to the crosspoint. In both cases (2) and (3), since the cell is in the crosspoint after the input scheduling phase and has the earliest departure time, it will be selected in the output scheduling phase. The cell will then reach the output during the time slot, and therefore the cell departs on time. ■

E.2 Mimicking an OQ Switch Using a WRR Policy

In this section, we will prove Theorem 11 which states that a buffered crossbar with a speedup of two can mimic an OQ switch using a WRR policy. In what follows, consider the following virtual finish time assignment policy when a cell arrives to the crosspoint. Assume a cell c arrives to the crosspoint B_{ij} at time t. Without loss of generality, let this be the k^{th} cell from input i to output j.

Case 1: If the $k-1^{th}$ cell is still present in output j of the buffered crossbar, then the output of the buffered crossbar will assign the virtual finish time, $F_i^{k-1} + \frac{1}{\phi_i}$.

Case 2: If the $k-1^{th}$ cell is not present in output j and the k^{th} cell is not transferred to the crosspoint in the scheduling phase immediately after its arrival, then the output of the buffered crossbar will assign a virtual finish time of $F_i^{k-1} + \frac{1}{\phi_i}$.

Case 3: If the $k-1^{th}$ cell is not present in the output j of the buffered crossbar and the k^{th} cell is transferred to the crosspoint immediately after its arrival, then the output of the buffered crossbar will assign the virtual finish time, $V(t) + \frac{1}{\phi_i}$.

We further assume that the buffered crossbar has a speedup of two, and the output picks cells with the smallest virtual finish time from the non-empty crosspoints.

Lemma 3 *The virtual finish time of every cell c is the same in the WRR-buffered crossbar switch and the WRR-OQ switch.*

Proof: Assume that the buffered crossbar has correctly calculated the virtual finish time of all cells which have arrived to the crosspoints up until time $t-1$ and the outputs have chosen the cells from their crosspoints which have the smallest finish time in every scheduling phase. From the results in section 3.3, this means that the buffered crossbar with a speedup of two has mimicked the WRR-OQ switch up until time $t-1$. Let t be the first time that the virtual finishing time of a cell calculated is different from the virtual finishing time calculated by the WRR-OQ switch. Consider that cell c which arrives to the crosspoint B_{ij} at time t was incorrectly calculated. Without loss of generality, let this be the k^{th} cell from input i to output j. We consider three cases.

Case 1: If the $k-1^{th}$ cell is still present in output j of the buffered crossbar, then this means that it was also present in the WRR-OQ switch when the k^{th} cell arrived. So both the WRR-OQ switch and the output of the buffered crossbar will assign the same virtual finish time, $F_i^{k-1} + \frac{1}{\phi_i}$, which contradicts our assumption.

Case 2: If the $k-1^{th}$ cell is not present in output j and the k^{th} cell is not transferred to the crosspoint in the scheduling phase immediately after its arrival, then it must have been inserted behind the $k-1^{th}$ cell in the input priority list or inserted to the head of the input priority list since the $k-1^{th}$ cell is currently in the crosspoint. Since the buffered crossbar switch has mimicked the WRR-OQ switch up until time $t-1$, this means that the $(k-1)^{th}$ cell was also present in the WRR-OQ switch at time a_i^k. The output of the buffered crossbar assigns a virtual finish time of $F_i^{k-1} + \frac{1}{\phi_i}$ which matches the virtual finish time assigned by the WRR-OQ switch. The assignment is the same which contradicts our assumption.

Case 3: If the $k-1^{th}$ cell is not present in the output j of the buffered crossbar and the k^{th} cell is transferred to the crosspoint immediately after its arrival, then since the buffered crossbar switch has mimicked the WRR-OQ switch up until time $t-1$, neither switch has cells in the system from input i destined to output j. So both the WRR-OQ switch and the output of the buffered crossbar will assign the same virtual finish time, $V(t) + \frac{1}{\phi_i}$, which again contradicts our assumption.

So the virtual finish time of a cell at time t can also be correctly calculated. ■

The above lemma and the fact that the WRR-OQ switch is a special case of the restricted PIFO-OQ policy imply the following theorem.

Theorem 4 *(Sufficiency) A buffered crossbar can mimic an OQ switch using a weighted round-robin policy with speedup of two, regardless of the incoming traffic pattern.*

E.3 Mimicking a PIFO OQ Switch

In this section, we will prove that a buffered crossbar with a speedup of three can mimic any PIFO OQ switch. Before we prove Theorem 12, we will need the following lemmas.

Lemma 5 *After the modified arrival phase, all cells in the crosspoints B_{ij} will have earlier departure order than any cell queued at input i destined for output j.*

Proof: Assume that the above property holds up until time $t-1$. Let t be the first time that any cell c in the crosspoint does not have the earliest departure order as compared to any cell queued at input i destined for output j. At time t, there can be at most one newly arriving cell c to an input. If the arriving cell has a earlier departure order than the cell in the corresponding crosspoint, then the modified arrival phase allows cell c to swap with the cell in the corresponding crosspoint which contradicts our assumption. ■

Lemma 6 *The slackness $L(c)$ of a cell c decreases by at least one in each scheduling phase.*

Proof: Since Lemma 5 guarantees that the cell in the crosspoint has the earliest departure order compared with any cell queued in the corresponding input queue, Lemma 9 still holds. ■

Lemma 7 *The slackness $L(c)$ of a cell c waiting on the input side is non-decreasing from time slot to time slot.*

Proof: Given Lemma 6, the only other difference as compared to Lemma 9 is in the modified arrival phase. Irrespective of whether a swap occurred or not, there is only one newly arriving cell to deal with, i.e., if a swap does not occur, then it is a cell which just arrived at the input, else if a swap occurs, then the newly arriving cell is the cell from the swapped crosspoint. The rest of the proof is similar to Lemma 9. ∎

Lemma 8 *The slackness $L(c)$ of a newly arriving cell c is non-negative.*

Proof: As described in Lemma 7, we only need to be concerned about inserting one newly arriving cell to the priority list at the input irrespective of whether a swap occurred or not. The rest of the proof is similar to Lemma 10. ∎

Theorem 4 *A buffered crossbar can mimic a PIFO-OQ switch (and hence give delay guarantees) with speedup of three regardless of the incoming traffic pattern.*

Proof: Given Lemma 7 and Lemma 8, the proof is exactly the same as the proof for Theorem 10. ∎

Appendix F

Buffered Crossbar with N Cells Per Output

In this appendix, we will prove Theorem 13 which states that a buffered crossbar with a speedup of two and N cells of buffering per output can mimic a PIFO-OQ switch with a fixed delay of $N/2$ time slots.

Proof: An input can receive at most $p + N - 1$ grants over any p consecutive scheduling phases. If the input adds new grants to the tail of a *grant FIFO*, and reads one grant from the head of the grant FIFO in each scheduling phase, then the grant FIFO will never contain more than $N - 1$ grants. Each time the input takes a grant from the grant FIFO, it sends the corresponding cell to the set of N crosspoints for its output. Because the grant FIFO is served once per phase, a cell that is granted at scheduling phase p will reach the output crosspoint by phase $p + N - 1$.

We need to verify that the per-output buffers in the crossbar never overflow. If the crosspoint scheduler issues a grant at phase p, then the corresponding cell will reach the output crosspoint between phases p and $p + N - 1$. Therefore, during scheduling phase p, the only cells which can be in the output crosspoint are cells which were granted between phases $p - N$ to $p - 1$. With N buffers per output, the buffers will never overflow, and each cell faces a delay of at most N scheduling phases, i.e., $N/2$ time slots (because $S = 2$). ■

Bibliography

[1] P. Gupta, "Algorithms for routing lookups and packet classification," Ph.D. Thesis, Stanford University, 2000.

[2] C. Partridge *et al*, "A fifty gigabit per second IP router," *IEEE/ACM Transactions on Networking*, Vol. 6, No. 3, pp. 237–248, June 1998.

[3] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "The Tiny Tera: A packet switch core," *Hot Interconnects V*, Stanford University, August 1996.

[4] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm," *ACM Computer Communication Review*, pp. 3–12, 1989.

[5] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 3, pp. 344–357, June 1993.

[6] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *IEEE/ACM Transactions on Networking*, Vol. 4, No. 3, pp. 375–385, June 1996.

[7] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division switch," *IEEE Transactions on Communications*, Vol. 35, No. 12, pp. 1347–1356, December 1987.

[8] T.E. Anderson, S.S. Owicki, J.B. Saxe, and C.P. Thacker, "High speed switch scheduling for local area networks," *ACM Transactions on Computer Systems*, Vol. 11, No. 4, pp. 319–352, November 1993.

[9] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, Vol. 47, No. 8, pp. 1260–1267, August 1999.

[10] Y. Tamir and H.C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 1, pp. 13–27, January 1993.

[11] A. Mekkittikul and N. McKeown, "A starvation-free algorithm for achieving 100% throughput in an input-queued switch," *Proceedings of ICCCN*, pp. 226–231, October 1996.

[12] A. Mekkittikul and N. McKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," *Proceedings of INFOCOM*, pp. 792–799, April 1998.

[13] I. Iliadis and W.E. Denzel, "Performance of packet switches with input and output queueing," *Proceedings of ICC*, pp. 747–753, April 1990.

[14] A.L. Gupta and N.D. Georganas, "Analysis of a packet switch with input and output buffers and speed constraints," *Proceedings of INFOCOM*, pp. 694–700, April 1991.

[15] Y. Oie, M. Murata, K. Kubota, and H. Miyahara, "Effect of speedup in nonblocking packet switch," *Proceedings of ICC*, pp. 410–414, June 1989.

[16] J.S.-C. Chen and T.E. Stern, "Throughput analysis, optimal buffer allocation, and traffic imbalance study of a generic nonblocking packet switch," *IEEE Journal on Selected Areas Communication*, Vol. 9, No. 3, pp. 439–449, April 1991.

[17] B. Prabhakar and N. McKeown, "On the speedup required for combined input and output queued switching," Stanford University Technical Report, STAN-CSL-TR-97-738, November 1997.

[18] D. Gale, and L.S. Shapley, "College admissions and the stability of marriage," *American Mathematical Monthly*, Vol. 69, pp. 9–15, 1962.

[19] Rojas-Cessa, E. Oki, Z. Jing, and H.J. Chao, "CIXB-1: Combined input-one-cell-crosspoint buffered switch," *IEEE Workshop on High Performance Switching and Routing*, July 2001.

[20] Rojas-Cessa, E. Oki, and H.J. Chao, "CIXOB-k: Combined input-crosspoint-output buffered packet switch," *IEEE Globecom*, pp. 2654-2660, November 2001.

[21] L. Mhamdi and M. Hamdi, "MCBF: A high-performance scheduling algorithm for buffered crossbar switches," *IEEE Communications Letters*, Vol. 7, No. 9, pp. 451–453, September 2003.

[22] D. Stephens and H. Zhang, "Implementing distributed packet fair queueing in a scalable switch architecture," *Proceedings of INFOCOM*, pp. 282–290, March 1998.

[23] T. Javidi, R.B. Magill, and T. Hrabik, "A high throughput scheduling algorithm for a buffered crossbar switch fabric," *Proceedings of IEEE International Conference on Communications*, pp. 1586–1591, June 2001.

[24] R.B. Magill, C. Rohrs, and R. Stevenson, "Output-queued switch emulation by fabrics with limited memory," *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. 4, pp. 606–615, May 2003.

[25] N. Chrysos and M. Katevenis, "Weighted fairness in buffered crossbar scheduling," *IEEE Workshop on High Performance Switching and Routing*, June 2003.

[26] S. Iyer, R. Zhang, and N. McKeown, "Routers with a single stage of buffering," *ACM SIGCOMM*, September 2002.

[27] A. Aziz, A. Prakash, and V. Ramachandran, "A near optimal scheduler for switch-memory-switch routers," *ACM Symposium on Parallelism in Algorithms and Architectures*, June 2003.

[28] K. Yoshigoe and K. J. Christensen, "Design and evaluation of a parallel-polled virtual output queued switch," *Proceedings of the IEEE International Conference on Communications*, pp. 112–116, June 2001.

[29] N. McKeown, C. Calamvokis, and S.T. Chuang, "A 2.5Tb/s LCS switch core," Hot Chips, August 2001.

[30] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, Vol. 37, No. 12, pp. 1936–1949, December 1992.

[31] H.J. Kushner, Stochastic Stability and Control, Academic Press. 1967.

[32] G. Fayolle, "On random walks arising in queuing systems: ergodicity and transience via quadratic forms as lyapunov functions - Part I," Queueing Systems, Vol. 5, pp. 167–184, 1989.

[33] E. Leonardi, M. Mellia, F. Neri, and M.A. Marsan, "On the stability of input-queued switches with speed-up," *IEEE/ACM Transactions on Networking*, Vol. 9, No. 1, pp. 104–118, February 2001.