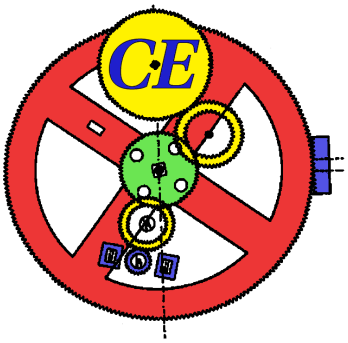


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2010

MSc THESIS

Performance Analysis and Cost-Performance Tradeoffs of a High Performance Partially Buffered Crossbar Switch

Nikolaos Skalis

CE-MS-2010-10

Abstract

Why is it hard to build high-speed routers? Because high-speed routers are like marriages; they are unpredictable, provide no guarantees, and become vulnerable in adversity. High-speed networks including the Internet backbone suffer from a well-known problem; packets arrive on high-speed routers much faster than commodity memory can support. On a 10 Gb/s link, packets can arrive every 32 ns, while memory can only be accessed once every 50 ns. If we are unable to bridge this performance gap, then (1) We cannot create Internet routers that reliably support links >10 Gb/s. (2) Routers cannot support the needs of real-time applications such as voice, video conferencing, multimedia, gaming, etc., that require guaranteed 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!

The emphasis is put on the switching architecture of a router. This thesis lays down a theoretical foundation for the Partially Buffered Crossbar switches and is about managing and resolving the preferences and contention for memory between packets from participating inputs and outputs in a switch. By combining the theory of fluid models, Lyapunov functions and the pigeonhole principle, the requirements for devising practical algorithms which can provide guarantees and emulate the performance of the ideal Output Queued switch and approximate the optimal Maximum Weight Matching scheduler are drawn up. The solutions described in this thesis, relax the memory access and bandwidth constraint, in fact, there is no better switching architecture described till now in terms of memory requirements and practicality regarding its achieved performance. Moreover, this thesis derives the first study of scheduling unicast and multicast traffic simultaneously in a Partially Buffered Crossbar switch.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Performance Analysis and Cost-Performance Tradeoffs of a High Performance Partially Buffered Crossbar Switch

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Nikolaos Skalis
born in Lamia, Greece

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Performance Analysis and Cost-Performance Tradeoffs of a High Performance Partially Buffered Crossbar Switch

by Nikolaos Skalis

Abstract

Why is it hard to build high-speed routers? Because high-speed routers are like marriages; they are unpredictable, provide no guarantees, and become vulnerable in adversity. High-speed networks including the Internet backbone suffer from a well-known problem; packets arrive on high-speed routers much faster than commodity memory can support. On a 10 Gb/s link, packets can arrive every 32 ns, while memory can only be accessed once every 50 ns. If we are unable to bridge this performance gap, then (1) We cannot create Internet routers that reliably support links >10 Gb/s. (2) Routers cannot support the needs of real-time applications such as voice, video conferencing, multimedia, gaming, etc., that require guaranteed 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!
The emphasis is put on the switching architecture of a router. This thesis lays down a theoretical foundation for the Partially Buffered Crossbar switches and is about managing and resolving the preferences and contention for memory between packets from participating inputs and outputs in a switch. By combining the theory of fluid models, Lyapunov functions and the pigeonhole principle, the requirements for devising practical algorithms which can provide guarantees and emulate the performance of the ideal Output Queued switch and approximate the optimal Maximum Weight Matching scheduler are drawn up. The solutions described in this thesis, relax the memory access and bandwidth constraint, in fact, there is no better switching architecture described till now in terms of memory requirements and practicality regarding its achieved performance. Moreover, this thesis derives the first study of scheduling unicast and multicast traffic simultaneously in a Partially Buffered Crossbar switch.

Laboratory	:	Computer Engineering
Codenumber	:	CE-MS-2010-10

Committee Members :

Advisor:	Koen Bertels, CE, TU Delft
Advisor:	Lotfi Mhamdi, CE, TU Delft
Chairperson:	Koen Bertels, CE, TU Delft
Member:	Fernando A. Kuipers, NAS, TU Delft
Member:	Stephan Wong, CE, TU Delft
Member:	Zaid Al-Ars, CE, TU Delft

Contents

List of Figures

Acknowledgements

I am indebted to many for their advice and assistance throughout my time in Delft.
I would like to thank Kees Goossens and Lotfi Mhamdi for giving me the opportunity to study about "switches and switching", a subject that I found very interesting from the beginning.
I would like also to thank Sundar Iyer for his truly inspiring and seminally described work regarding the load balancing and parallelism principles in the design of routers.
Special thanks to TU Delft, for giving me the opportunity to interact with some extraordinary and talented people from all over the world and to the many picturesque towns of the Netherlands where I traveled.
My time at Delft has been a fantastic period. I would like to thank all my friends who have given me many wonderful, meaningful, creative and of course, hilarious moments. I am deeply grateful to my friends for their love, support and patience.
The journey would not be possible without the efforts of my family; my father Panagiotis, my mother Magda and my brother Giannis, for always encouraging me to make the most out of my life and for providing their moral support to my work during these years. I dedicate this thesis to them.

Nikolaos Skalis
Delft, The Netherlands
June 1, 2010

1 Introduction

The Internet is widely considered the most reachable platform for the network infrastructure. Whereas the availability of unlimited bandwidth has triggered a plethora of services like file sharing and streaming media, there is no getting away from the fact that the Internet is based on statistical multiplexing which is facilitated through the packet mode or packet oriented communication. The statistical multiplexing principle implies the sharing of a link that is able to adapt in some way to the instantaneous traffic demands of the nodes connected to that link. Since resources are shared, traffic bandwidth must be able to be allocated on demand and fairly between different users of that bandwidth. Given that service providers wish for high capacity routers that provide guaranteed performance and that the latest backbone routers are designed to scale well [3], researchers are continually exploring faster switching technologies.

The Internet consists of end-hosts, links, and routers. This thesis focusses on the design and architecture of high performance Internet routers, the focus is on the switching architecture, that make up the backbone of the Internet — in particular, how to design them to have a guaranteed performance and yet still be implementable.

A router consists of several processing stages. At the very least, a router has two main processing stages; address lookup and switching. In the address lookup stage, a router must decide for each incoming packet where to send it next, that is, finding the address of the next-hop router as well as the egress port through which the packet should be sent. In the switching stage, a router tries to resolve the output port contention when more than one packet is destined for the same output port at the same time.

Analysis of the traffic-performance relation, linking capacity, demand and performance, for a range of streaming and elastic traffic types leads us to believe adequate performance can be assured much more simply than in the classical Quality-of-Service (QoS) architectures and more reliably than in an over-provisioned best effort network. Although flows may be clearly identifiable from a higher level system view, for instance from a software task-graph, this information is not communicated to the lowest level hardware. A *flow-aware* router [4] is distinguished from a traditional router in that it is capable of keeping track of flows passing by and applying different classes of service to each flow.

How a packet is processed by a flow-aware router is depicted in Figure 1.1 from when it arrives on the ingress line until it departs on the egress line. At the switching phase, packets are segmented into cells, traverse the switch fabric as cells, and reassembled back into packets again before they leave the router[1]. Switching consists of the operations that guide an incoming packet to the output buffer, the last possible waiting room before the packet is placed on the link towards the next hop router. The destination and the service class of the packet are recorded in the packet header. The path from input port

[1] A common technique used in high-performance routers is to segment the incoming variable-length packets into fixed-length packets (*cells*).

to output port is a well-ordered concatenation of elementary queueing operations over several stages directed by internal routing [5].

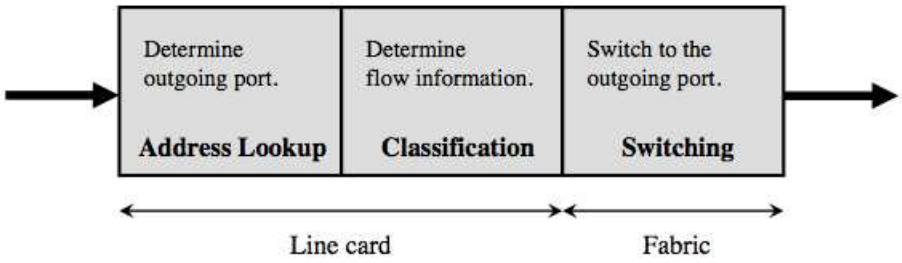


Figure 1.1: Datapath of a packet through a flow-aware router.

Network operators need to design and plan their networks. Most service providers offer excellent quality of service simply by keeping link utilization low (less than 50%, say). Packet delays are then very small and loss almost negligible. No commercial router today can guarantee that 100% of its capacity is available to the network operator, which in turn makes it hard to plan a network.
In summary, network operators want routers which provide throughput, bandwidth, and delay guarantees. Routers usually employ a crossbar switching fabric, as it is non-blocking, memoryless and introduces no delay. The bottleneck that prevents routers from providing these guarantees is scheduling in the switching stage. The throughput of a switching architecture is mainly affected by the queueing architecture and the scheduling algorithm. A crossbar-based router requires a centralized scheduler to determine when cells are to traverse the switch fabric. The scheduling problem becomes more and more difficult as line rates and number of ports increase.

1.1 Routers As the Backbone of the Internet

Today's Internet is an amalgamation of thousands of commercial and service provider networks. It is not feasible for a single service provider to connect two distant nodes on the Internet. Therefore, service providers often rely on each other to connect the dots. Router design is often guided by the economic requirements of service providers. Service providers would like to reduce the infrastructure and maintenance costs while, at the same time, increasing available bandwidth and reliability. To this end, network backbone has a set of well-defined, narrow requirements. One strategy to deal with the exponential growth of the traffic in Internet is to design router with higher and higher switching capacity. Routers in the backbone should simply move traffic as fast as possible. Backbone links rates are evolving from today's OC-48 (2.5 Gbps) to OC-192 (10 Gbps) and even OC-768 (40 Gbps) [6], with a rate of increase of about 30% per year [6]. This means that, for a minimum size TCP/IP packet of 40 bytes, the number of packets to be processed is evolving from 8 to 32 and even 125 million of packets/s. So the average time spent to elaborate a single packet (doing at least routing and switching) is decreasing from 125 ns to 30 ns and even 8 ns; hence, the switching process at high speed must be implemented in hardware.

1.1.1 Integration of the cell switch inside a router

A high-performance Internet Protocol (IP) router built around an Asynchronous Transfer Mode (ATM) cell-switch will be used as a reference model.
An IP router is designed to minimize the state information on individual packets. The IP protocol sits on top of the data-link protocol at the input and at the output of the router, thus an IP router does not look into the actual data contents that the packet carries. Input IP packets are segmented into ATM cells, that will be transferred to output ports by a high-performing ATM switching fabric. The designers of ATM utilized small data cells to reduce jitter (delay variance, in this case) in the multiplexing of data streams. Once cells are delivered to an output port, they are reassembled into the IP packet, which is transmitted on the output line.

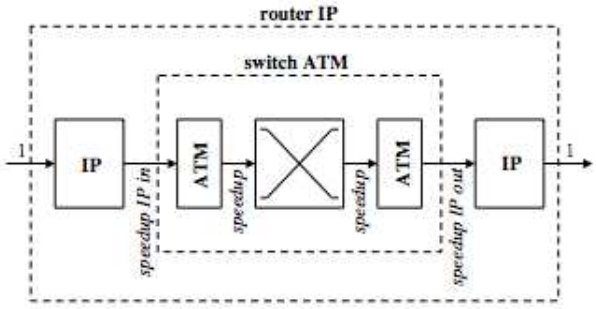


Figure 1.2: Speedup definitions for the packet switch built around a cell switch.

Figure 1.2 depicts the possible speed variations inside a router with an integrated cell-switch.

- **speed-UP-in:** specifies the rate at which cells are moved from the input IP module to the input of the cell-switch. When a packet is segmented into k cells, all these cells are sequentially and consecutively (assuming no contention) transferred to the cell-switch input in k time slots, we say that speed-UP-in=1.

- **speedup (S):** up to S cells can be read from each input and written to each output of the switch in one time slot. is the number of cells per slot that can be read from the inputs of the cell-switch, and it is also the number of cells per slot that can be written onto a switch output.

- **speed-UP-out:** specifies the rate at which cells are moved from the output of the cell-switch to the output IP module.

The input and output IP modules that appear in Figure 1.2 consist of a number of queues that serve variable-length packets in store-and-forward mode. This is where the packet segmentation and reassembly functionality is provided. The ATM modules are comprised of input and output cell queues of the crossbar-based switch.

1.2 Background

Consider a switch with N input and N output ports. We will denote R (usually denoted in Gb/s) to be the rate at which cells of size C arrive at every input and depart from every output. We normalize time to the arrival time between cells (C/R) at any input, and refer to it as a *time slot*.

1.2.1 The ideal switch

The ideal switch is typically modeled as a centralized shared memory switch. The memory requirements for the switching architecture in question are:

1. *Bandwidth:* A total memory bandwidth of $2NR$ Gb/s is required, as the centralized shared memory must be able to store all N arriving cells and transmit up to N cells at the same time.

2. *Access Time:* Given that the memory bandwidth is equal to the ratio of the width used for a memory access and the random access time (equals $A_t = C/2NR$ seconds), if $N = 8$, $R = 10\ Gb/s$, $C = 64\ bytes$ and a 32-bit-wide memory with 50 ns random access time is used, then the centralized shared memory must be accessed every 3.2 ns.

3. *Capacity:* If we assume an Internet RTT (where RTT is the round-trip time for flows passing through the switch) of approximately 0.25 s, $N = 8$ and $R = 10\ Gb/s$ then $2.5\ Gb \times 8 = 20\ Gb$ of memory are required[2].

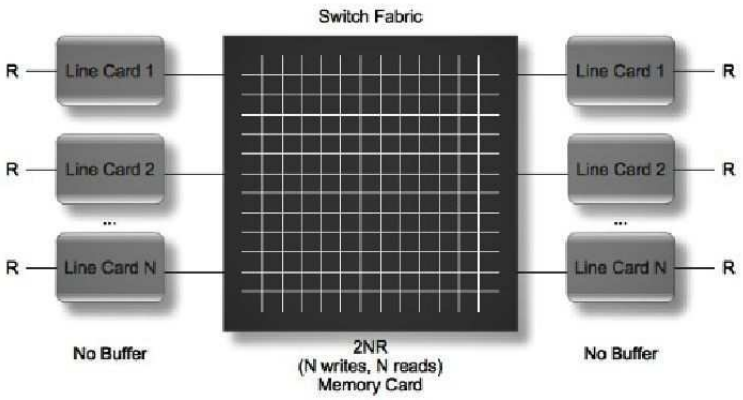


Figure 1.3: The architecture of a centralized shared memory router.

If the memory meets these three requirements [2], then the switch exhibits minimum cell latency because the cells face no constraints on how and when they arrive and depart.

[2]As a rule of thumb, the buffers in a switch are sized to hold approximately $RTT \times R$ bits of data.

1.2.2 A note about memory technology

Two main memory technologies are available in the market today; Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM). SRAM is more expensive, but faster and significantly less power hungry (especially idle) than DRAM. It is therefore used where either bandwidth or low power, or both, are principal considerations. The advantage of DRAM is its structural simplicity; only one transistor and a capacitor are required per bit, compared to six transistors in SRAM. This allows DRAM to reach very high density.

"With today's CMOS technology, the largest available commodity SRAM [7] is approximately 72 Mb, has a bandwidth of 72 Gb/s, an access time of 2 ns, and costs \$70." [2]

So, in order to meet the capacity requirement, the ideal switch would need more than 275 SRAMs, and the memories alone would cost over $\$19K$ — greatly exceeding the selling price of an Enterprise router today! Although possible, the cost would make this approach impractical.

"The largest commodity DRAM [8] available today has a capacity of 1 Gb, a bandwidth of 36 Gb/s, an access time of 50 ns, and costs \$5." [2]

We cannot use DRAMs because the access rate is an order of magnitude above what is required. If our switch has more than $N = 16$ ports, the access time requirement would make even the fastest SRAMs inapplicable.
The centralized shared memory switch is an example of an output queued (OQ) switch. In an OQ switch, cells are immediately forwarded to the destined output ports once they arrive at the inputs. Since in an OQ switch each output port has a memory, there is no contention from different outputs, the OQ switch has better quality-of-service (QoS) control ability, which comes at the cost of a memory speed constraint. Each memory must have an access rate equal to $(N + 1)$ times the line rate in order to cater for N writes and one read per time slot, limiting, thus, the switch size.

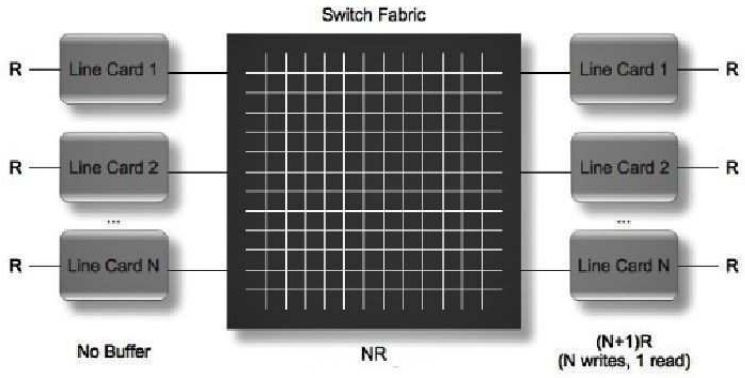


Figure 1.4: Output queued switch.

While this approximately halves the access time requirement compared to the centralized shared memory switch, this approach is still extremely impractical at high speeds.

1.2.3 Why is memory access time a hard problem ?

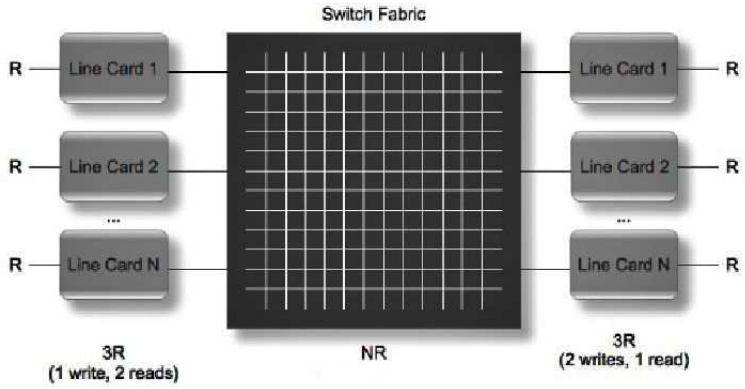
"In 1997, the extremely small-size data accesses, equal to the size of the smallest-size 64-byte packet, required by networking was identified as a fundamental upcoming problem." [2]

The random access time of commercial DRAMs has decreased by only 1.1 times every 18 months (slower than Moore's Law) [9]. In contrast, as line rates increase (usually at the rate of Moore's Law), the time it takes these small-size packets to arrive grows linearly smaller, and as a result the problem becomes harder and harder. Designing high-speed and scalable switches mandates us to alleviate the memory access rate problem.

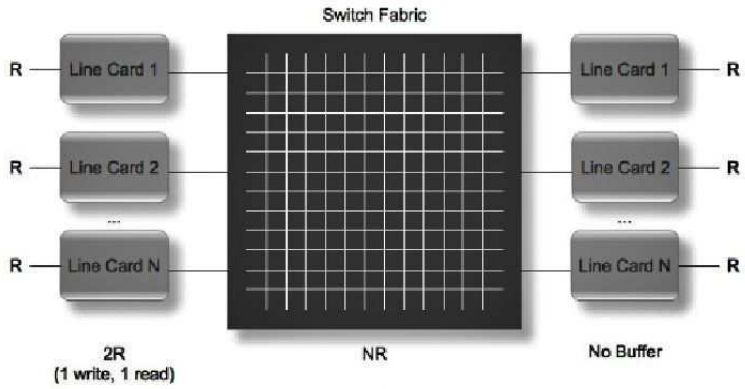
"By 2005, routers had to be built to support the next-generation 40 Gb/s line cards. By then, this had become a pressing problem in immediate need of a solution." [2]

1.2.4 Buffering strategies to alleviate the memory access time problem

Reviewing the buffering strategy in crossbar switches; cell switches based on a crossbar architecture with virtual-output-queueing (VOQ) are attractive for use in high speed networks, as the bandwidth of the input buffers is twice the line rate (at most one cell can be transferred to an input and at most one buffered cell can be transferred through the crossbar). Another popular switch architecture is the combined input-output queued (CIOQ) switch, shown in Figure 1.5(a). A CIOQ switch buffers cells twice — once at the input, and again at the output. This switch can behave identically to an output queued switch if the memory on each line card runs at rate $3R$, the switching interconnect runs at rate $2NR$, and it implements a complex scheduling algorithm [10].
In an input queued (IQ) switch with VOQs, the switching interconnect needs to carry up to N cells from the inputs to the respective outputs and needs a bandwidth of $2NR$. More important, each memory only needs to run at a rate $2R$ (instead of $2NR$), enabling higher-capacity routers to be built. So, in IQ switches, the switching fabric and the input line interface can operate at a rate that does not grow with the switch size. An unbuffered IQ switch, Figure 1.5(b), requires a centralized scheduler to resolve two main blocking problems, namely input and output contention. Input contention results from the constraint that an input can send at most one cell every time slot. Similarly, output contention arises from the constraint that an output can receive at most one cell every time slot. These blockings make the task of the scheduler complex and the cells delay unpredictable, as the scheduler implements complex stable marriage algorithms [11].



(a)



(b)

Figure 1.5: (a) Combined input output queued switch. (b) Input queued switch.

"Given N men and N women, where each person has ranked all members of the opposite sex with a unique number between 1 and N in order of preference, marry the men and women off such that there are no two people of opposite sex who would both rather have each other than their current partners. If there are no such people, the marriages are *stable*.

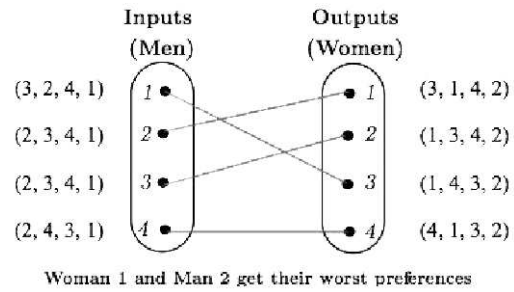


Figure 1.6: A stable marriage.

An easier way to understand this is captured in the following quote [12] which presupposes a married woman who professes interest in marrying another man. If she receives the reply, ‘‘Madam, I am flattered by your attention, but I am married to someone I love more than you, so I am not interested’’, and this happens to any woman who wants to switch (or vice versa) then the set of marriages is said to be stable.
The algorithm was first applied for pairing medical students to hospital jobs. Gale and Shapely [11] proved that there is always a set of stable marriages, irrespective of the preference lists. The latter fact is key to analyzing crossbar routers.’’ [2]

Tassiulas and Ephremides [13] and McKeown et al. [14] proved that an IQ router with VoQs can achieve 100% throughput with a maximum weight matching (MWM) algorithm, if the input traffic is i.i.d and admissible[3]; the outputs are allowed to be non-uniformly loaded. Dai and Prabhakar [15] generalized this result and showed that MWM can achieve 100% throughput provided that the input traffic satisfies the strong law of large numbers and is admissible. However, MWM is extremely complex to implement and has a time complexity, $O(N^3 logN)$. One would expect that the maximum size matching (MSM) algorithm, which maximizes the instantaneous bandwidth of the crossbar (the most efficient algorithm has a lower time complexity $O(N^{2.5})$, would also be able to achieve 100% throughput. However, contrary to intuition, MSM is known to be unfair (if ties are broken randomly), can lead to starvation, and hence cannot achieve 100% throughput [14].
The existence of crosspoint queueing [16] relaxes the output contention constraint, making the scheduling task much simpler. Buffered crossbars (CICQ), Figure 1.7, use distributed and independent schedulers (one per input/output port) to switch cells from the input to the output ports of the switch that do not have to resolve two constraints in a time slot. A scheduling cycle consists of input scheduling, output scheduling and flow control to prevent crosspoint buffer overflow. The scheduling simplification comes at the expense of a costly crossbar, which is hard to scale, as the crossbar has to contain N^2 crosspoint buffers, where N is the number of input/output ports of the switch. The number of crosspoint buffers grows quadratically with the switch size and linearly with round trip delays [17]. This makes buffered crossbar switches highly expensive and hence less appealing.

[3]Please see Appendix A for a more formal definition.

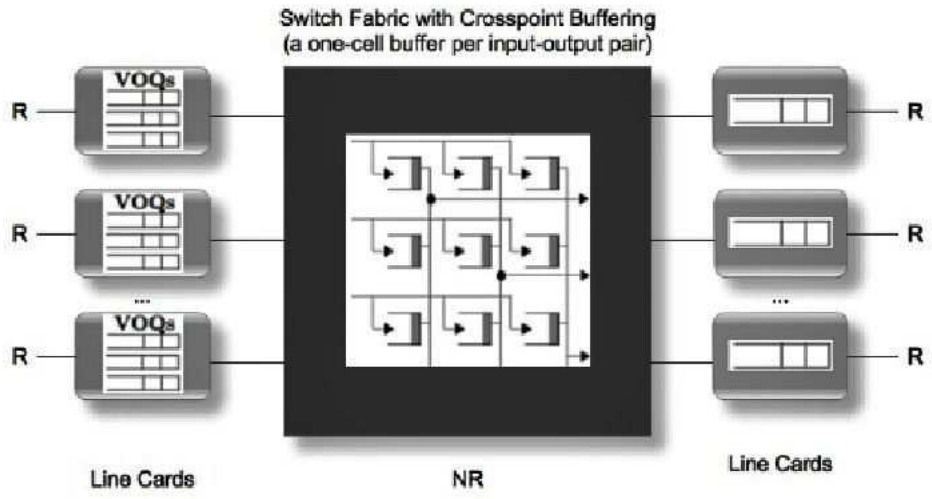


Figure 1.7: Crosspoint buffered crossbar switch.

The Partially Buffered Crossbar (PBC) switch [1] was designed to be the best compromise between unbuffered crossbars and fully buffered crossbars. First, it overcomes the high cost of fully buffered crossbars that use N^2 internal buffers, by using a low number of internal buffers (B) per output irrespective of N. Second, it overcomes the scheduling complexity experienced by unbuffered crossbars by means of distributed and pipelined scheduling algorithms, whereas the input scheduling phase resembles a scheduling cycle in unbuffered crossbars, as it is based on request-grant-accept handshaking protocol. A more detailed description of the PBC switching architecture is described in Chapter 2.

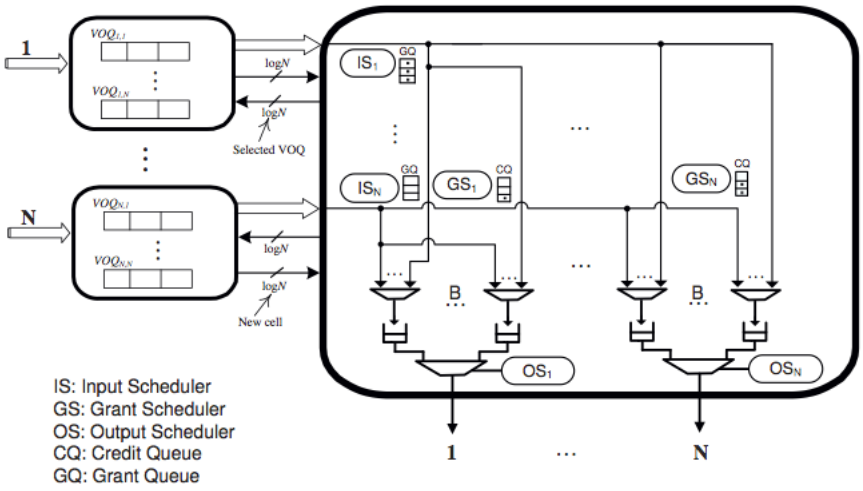


Figure 1.8: The Partially Buffered Crossbar (PBC) switching architecture [1].

1.3 Motivation

A switch designer is constrained by requirements and technological feasibility and must also decide how to distribute functionality over chips in order to optimize the overall system cost and power. Next, we draw attention to some issues concerning the design of a cell switch:

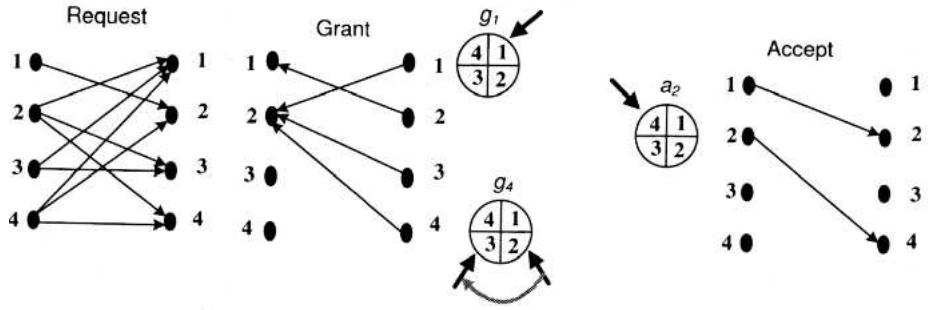
- Instead of increasing the port speed, the aggregate throughput of a switch gets larger and larger by increasing the number of ports.
- The demand of throughput/power density in terms of Gb/s per $Watt$ per physical volume is increasing. The throughput/power density of all the switch's components must be scaled up analogously to the increase in aggregate bandwidth. There is no getting away from the fact that, the increase in density offered by CMOS technology is invested in reducing size or power density and in increased functionality, such as level-4 routing, firewalling, MPLS, and DiffServ, rather than increasing speed.
- Although the access time will be reduced over time, the rate of improvement is much slower than Moore's Law [18]. Although Moore's law doubles performance every 18-24 months at constant cost, this performance is mostly due to increased density rather than increased clock speeds. From one CMOS generation to the next, taking global wiring into account, switch-chip clock speeds can be increased by only 5-10% [19]. To exploit Moore's law, more parallelism is required, typically at constant clock speeds. In turn, this results in more levels of pipelining in the control path and a higher degree of parallelism in the datapath. Similarly, on-chip memory speed does not increase, and memory busses become wider in each new generation.
- The increased RTT results in more packets in flight, and, independently of the switch architecture chosen, this needs to be accounted for with buffers in the system to ensure that the system is both work-conserving and lossless. In general, the amount of buffering required to ensure both losslessness and work-conservation must be scaled proportionally to the RTT. The flow-control mechanism selected has a large impact on buffer sizes. As a result, RTT has become a major design parameter for switch architectures [4].
- While internal speedup is a good solution[5], it does incur significant cost — the crossbar is more expensive (S times higher throughput), the buffer memories are

[4]The introduction of a significant RTT has implications for the egress buffer sizing. Even if there is no internal speedup, egress buffers usually are still necessary to accommodate packets when the downstream links are blocked (e.g., because of congestion in the downstream node). However, if the downstream link is 100% available, any input should be allowed to utilize this bandwidth fully, without the possibility of packets being lost, i.e., under completely unbalanced traffic both work-conservation and losslessness must be ensured. The egress buffer size per output must be scaled proportionally to RTT, speedup, and number of traffic classes to satisfy these requirements.

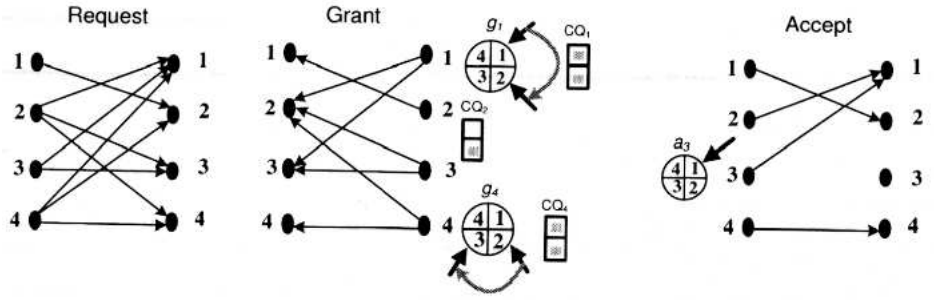
[5]The crossbar port rate is higher than line rate by a factor of S, considerably greater than 1. In this way, (1) since an average utilization of the crossbar outputs suffices for the egress lines to get fully utilized imperfect crossbar scheduling is acceptable, (2) due to the technique of segmenting variable-size packets into fixed-size cells, the required rate increase can be accommodated and (3) the emphasis is

more expensive $((1 + S)/2$ times higher throughput).

- The switch performance is essentially depended on the complexity of the centralized scheduler in IQ switches. By introducing a small amount of buffering in the crossbar, we can make the scheduler's job much simpler. The intuition is that when a cell is switched, it can wait in the buffer; it doesn't have to wait until both the input and output are free at the same time. But, the number of internal buffers grows quadratically with respect to the switch size and linearly with the RTT [17]. As shown in Figure 1.9 [1], a grant arbiter can send at most one grant per input in order to avoid input contention, similarly, an accept arbiter can accept at most one grant in order to avoid output contention. The scheduling process in a PBC switch allows up to B cells, destined to the same output, to be stored to the internal buffers. In this way, the output contention is relaxed whereas the input contention constraint is still enforced, bypassing the need of a complex centralized scheduler (as in IQ switches) and introducing some kind of awareness between the input and the output scheduler as opposed to the completely independent schedulers used in CICQ switches.



(a)



(b)

Figure 1.9: (a) The iSLIP scheduling algorithm. (b) A PBC scheduling cycle [1].

shifted to the egress-line subsystem, since queues now tend to build up on the output side of the crossbar (combined input-output queueing (CIOQ)) for providing QoS.

1.4 Organization

We saw (in section 1.2) that an ideal OQ switch requires a memory access rate that is proportional to the product of the number of ports N and the line rate of each port R. We will look at ways to reduce the access rate of the memory on high-speed routers. This is done by trading off memory access time with memory bandwidth and by devising schedulers that minimize the memory requirements, while the maximization of the switch's performance is still the main objective. Moreover, this work conducts the first study on integrated unicast and multicast traffic support in the PBC switching architecture that has recently been proposed and shown to be a good compromise between both the unbuffered and fully buffered crossbar switch architectures.
Additionally, in Appendix B, we present a system-level buffer planning algorithm that can be used to customize the design of a PBC switch. Given the traffic characteristics and the total budget of the available buffering space, our algorithm automatically assigns the internal buffer size for each output, such that the overall performance is maximized.

1.4.1 Mapping the MWM Optimal Scheduling to a PBC Switch

In Chapter 2, we introduce the PBC switching architecture and we investigate whether it is possible to map the bufferless MWM optimal scheduling algorithms to the PBC architecture and implement it in a pipelined and distributed fashion. It has so far been not possible to achieve this task for fully buffered crossbars and the reason is mainly attributed to the physically distributed internal buffers as well as their scheduling. We show how the PBC switch can practically run optimal scheduling and achieve almost 100% throughput under any admissible input traffic pattern without any speedup.

1.4.2 Performance Guarantees in a PBC Switch

In Chapter 3, we describe how a PBC switch with only a two-cell internal buffering per output port coupled with an adaptive frame-based scheduler can achieve 100% throughput, guarantees a minimum level of fairness that a flow perceives. We show the tradeoff between the Weighted-Max-Min-Fairness (WMMF) rates and the average cell latency compared to an OQ switch. Because we do not require a centralized scheduler and we do not impose any memory speed constraint with respect to the switch size, the proposed switching architecture and scheduling mechanism that allocates the available bandwidth with fairness in mind are considered practical. No commercial backbone router today can make hard guarantees on throughput [20].

1.4.3 A PBC Switch Supporting Integrated Unicast and Multicast Traffic

The growing proportion of multicast traffic on the Internet is stressing the need for efficient multicast support alongside the unicast traffic. Only little has been done on the support of integrated unicast and multicast traffic flows. Chapter 4 conducts the first study on integrated unicast and multicast traffic support in the PBC switching architecture. We consider the problem of scheduling cells in a PBC switch when both

unicast and multicast traffic are present. We concurrently schedule unicast and multicast cells and compare the performance of our schedulers with state-of-the-art algorithms used in IQ and CICQ switches. Our performance study shows that the PBC switch with its integrated scheduler can efficiently handle integrated unicast and multicast traffic flows, while maintaining the advantages of both the unbuffered and the fully buffered crossbar architectures.

1.5 Summary of Results

This thesis lays down the theoretical foundation for the PBC switches, by combining the theory of fluid models, Lyapunov functions and the pigeonhole principle. It describes practical algorithms which can emulate the performance of an OQ switch and approximate the optimal Maximum Weight Matching scheduler. It also helps us derive bounds on average VOQs occupancy and provide throughput guarantees. The solutions described in this thesis relax the memory access and bandwidth constraint, in fact, there is no better switching architecture described till now in terms of memory requirements and practicality regarding its achieved performance. Moreover, this thesis derives the first study of scheduling unicast and multicast traffic simultaneously in a PBC switch.

Mapping the MWM Optimal Scheduling to a PBC Switch

2

In this chapter, we describe in detail the PBC switching architecture and we apply a general methodology, mainly based upon Lyapunov functions, to derive bounds on averages of queue lengths. Although the stability properties (i.e., the limit throughput) of IQ and CIOQ cell-based switches were already studied for several classes of scheduling algorithms, very few analytical results concerning queue lengths are available in the technical literature. We attempt to address these issues for the case of the PBC switch. We concentrate on Maximum Weight Matching (MWM) that has been proved to maximize throughput. The MWM algorithm is perceived to be very good scheduling algorithm in general and simulations have suggested that it performs better than most of the known algorithms in terms of delay. But it is very complex to implement. Hence many simple to implement approximations to MWM have been proposed.

2.1 The Partially Buffered Crossbar (PBC) Architecture

In the following, we introduce the PBC [1] switching architecture together with its scheduling process.

2.1.1 Switch model

Each input port contains N VOQs, one per output port. The crossbar fabric contains a small number of internal buffers (B) organized per output. The building blocks of a PBC switch are the input scheduler (IS), the grant scheduler (GS), the output scheduler (OS), the credit queue (CQ) and the grant queue (GQ), as depicted in Figure 1.8. Each input i maintains an input scheduler. IS_i is responsible for transferring the cells from the linecard to the internal buffers. Each output has a grant scheduler, GS_j, that tracks the grants an output sends and together with the IS_i are responsible for the flow control of the fabric's internal buffers.
The communication mechanism between the IS_i and the GS_j is the grant queue, GQ. $GQ_{ij} = 1$ when GS_j sends a grant to IS_i and when the IS_i accepts the grant then $GQ_{ij} = 0$. A credit queue with B entries is maintained per output, CQ_j, that tracks the availability of the internal buffers of that output. CQ_j is decremented whenever a grant is sent to an input, and incremented during output scheduling. Lastly, the output scheduler OS_j arbitrates the departure of cells from the internal buffers to the output j in a First-Come-First-Serve (FCFS) manner.

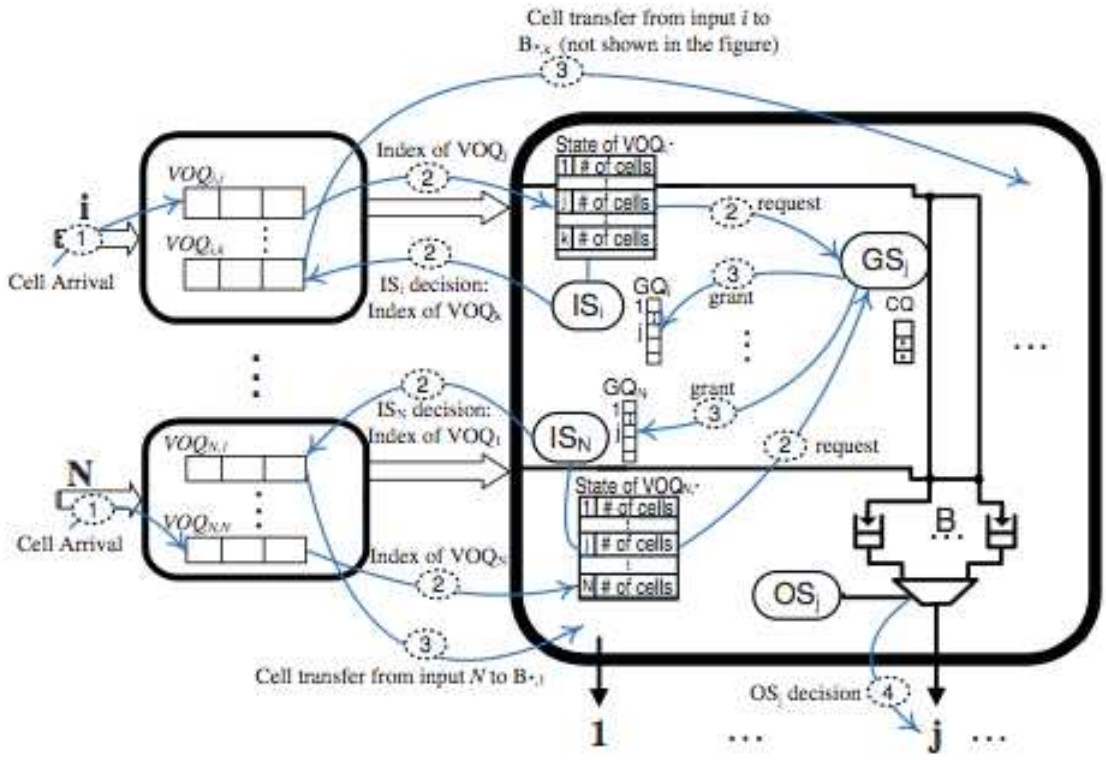


Figure 2.1: The dynamics of the PBC switch [1].

2.1.2 Scheduling process

The scheduling process in the PBC switch combines ideas from the scheduling disciplines used in IQ and CICQ switching architectures. More specifically, input scheduling works like in a unbuffered crossbar adopting the classical request-grant-accept handshaking protocol, whereas like in buffered crossbars, there are two scheduling phases; input and output scheduling. The sequence of events is displayed in Figure 2.1 [1].
During time slot t, the input (linecard) sends a $logN$ bit signal to IS_i about a new cell arrival. According to the internal buffers availability (CQ_j) and how the grant scheduler functions, a $logN$ bit signal is sent back to the input i indicating the selected VOQ_{ij}. At the same time, IS_i selects and transfers a Head-of-Line (HoL) cell based on the decision of the GS_j, but, note that the decision in question was taken during the previous time slot. In other words, the outcome of GS_j at time t is only valid during time slot $(t+1)$ or later. This in turn allows us to pipeline the input and the grant scheduling phases, as they are now completely decoupled in time.

2.2 The Maximum Weight Matching (MWM) Algorithm

Throughput and delay are used to evaluate a switch's performance. *Throughput* is defined to be the average number of cells transmitted in a time slot, and *delay* is defined to be the time experienced by a cell from arrival to departure. We say that a switch is *stable*, when the expected queue length is bounded: $E\left[\sum_{ij} L_{ij}\right] < \infty,\ \forall t$. If a switch is stable under any independent and admissible input traffic, then the switch can achieve 100% throughput.
Algorithms that obey in sophisticated scheduling disciplines are required to schedule cells to a crossbar in each time slot. In order to fully exploit the underlying interconnection architecture, the crossbar, the scheduling algorithm must adhere to disciplines that allow an efficient way to schedule cells on a per time slot basis. The latter can be modeled as a bipartite graph matching problem, according to which, there are N nodes that stand for the N input ports and another N nodes that stand for N output ports. If there is an input request for a particular output, this is denoted by an edge connecting the input-output pair of nodes. A scheduler is responsible for selecting a set of the edges from at most N^2 edges, where each input is connected to at most one output and each output is connected to at most one input. A matching (a set of edges) of input-output can be represented as a permutation matrix $M = (M_{ij}),\ i, j \leq N$, where $M_{ij} = 1$ if input i is matched to output j in the matching.
In a bipartite graph, we define w_{ij} as the weight of edge e_{ij} from input i to output j. Weight of a VOQ refers usually to the length of the VOQ (backlocked cells). The maximum weight matching (MWM) M for a bipartite graph is one that maximizes $\sum_{e_{ij} \in M} w_{ij}$. Figure 2.2 shows an example of a maximum weight match.

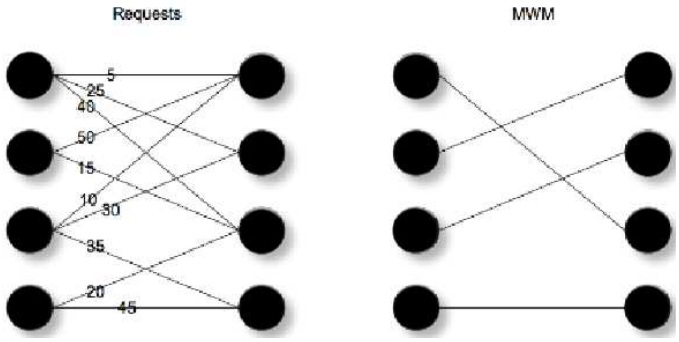


Figure 2.2: Maximum weight match example.

Theorem 2.1 **A maximum weight matching algorithm achieves 100 percent throughput under any admissible traffic [14][13].**

‘‘The criteria to choose a scheduling algorithm mainly are; (1) *Efficiency.* The algorithm should achieve high throughput and low delay. In other words, select a set of matches with more edges in each time slot. (2)

Fairness. The algorithm should avoid the starvation of each VOQ. (3) *Stability.* The expected occupancy of each VOQ should remain finite for any admissible traffic pattern. (4) *Implementation Complexity.* The algorithm should be easy for hardware implementation. High implementing complexity will cause long scheduling time, which further limits the line speed of the switch.'' [3]

What has to be noticed is that MWM can be solved in time $O(N^3 logN)$ [21], which is, too large for a high-speed packet switch. We can reduce the complexity of computing maximum weight matches, by devising approximations of MWM.

2.3 Related Work

In an IQ switch, a switch scheduler is necessary. The difficulty of the switch scheduling comes from two aspects. First, the scheduling has to be performed in a very short time. Second, co-existence of input and output contentions make the switch scheduling more complicated than in the case of an OQ switch. The switch scheduler must make sure that matchings between inputs and outputs are conflict-free at any time. Theoretically, the MWM algorithm [14] is proved to achieve 100% throughput for any admissible traffic. But its complexity is $O(N^3 logN)$, too complicated to be implemented in hardware. Furthermore, multicasting [22] and QoS guarantee [23] impose additional complexity on the switch scheduling.
Many algorithms [24][25][26][27][28] have been proposed to reduce the complexity. However, due to the time constraint, most of those proposed algorithms are still too complicated to be implemented in hardware. Buffers inside the crossbar (CICQ) separate the input contentions from the output contentions, so that each input and output scheduler can make decisions independently. Algorithms such as OCF-OCF [16], RR-RR [29] have been proposed. LQF-RR [30] was also proved to achieve 100% throughput when the arrival rate of each input-output pair $\lambda \leq 1/N$ for an $N \times N$ switch.

2.4 Use Memory as a Basis for Comparison

We believe that memory is useful as a barometer when comparing different switching architectures [2] for the following reasons:

1. A high capacity switch can be achieved by relaxing the memory access rate and memory bandwidth required, alleviating these two constraints is, and will continue to be, essential.

2. The leading Internet router vendor, Cisco Systems, spends roughly $800M p.a. on memory components. So, regarding the cost-efficiency of a switch, the memory components play a major role, as they alone contribute approximately 20% of the cost of materials, on average.

3. More power is consumed by a switch with higher memory bandwidth (high-speed

memory I/O contributes to approximately 33% of the overall power on Ethernet switches and Enterprise routers).

2.5 Approximation Class of Algorithms to MWM: 1-APRX

In the work of Shah and Kopikare [31], a class of approximations to MWM, 1-APRX, was proposed and defined as follows. Let the weight of a schedule obtained by a scheduling algorithm A be W^A. Let the weight of the maximum weight match for the same switch state be W^*. A is defined to be a 1-APRX to MWM, if the following property is always true: $W^A \geq W^* - f(W^*)$, where $f(\cdot)$ is a sub-linear function, that is, $\lim_{x\to\infty}[f(x)/x] = 0$ for any switch state.

Theorem 2.2 **Let $W^*(t)$ denote the weight of maximum weight matching scheduling at time t, with respect to switch state $Q(t)$ (where $Q(t) = [Q_{ij}(t)]$ and $Q_{ij}(t)$ denotes the number of cells in VOQ_{ij}). Let A be a 1-APRX to MWM and $W^A(t)$ denote its weight at time t. Further, A has a property that,**

$$W^A(t) \geq W^*(t) - f(W^*(t)) \ \forall t,$$

where $f(\cdot)$ is a sub-linear function. Then the scheduling algorithm A is stable under any admissible Bernoulli i.i.d. input traffic.

The above theorem can be used to prove the stability of some matching algorithms that are not MWM and with lower complexity.

2.6 The EX-DROP-PR scheduling algorithm

Exhaustive service match is a class of matching algorithms inspired by exhaustive service polling systems. In an exhaustive service matching algorithm, when an input is matched to an output, all the cells waiting in the corresponding VOQ will be served continuously before any other VOQ related to that input and that output can be served. The stability of EX-DROP-PR is achieved due to two efforts — (1) Unlike most other matching algorithms, which try to find efficient matches in each time slot, exhaustive service matching achieves efficiency by minimizing the matching overhead over multiple time slots. (2) Cells forwarded to outputs are held in reassembly buffers that can only leave the switch when all cells belonging to the same packet are received so that the packet is reassembled. The total delay a packet suffers, from the time it arrives at the input to the time it departs at the output, includes the cell delay incurred traversing the switch and the time needed for packet reassembly. In exhaustive service matching, since all the cells belonging to the same packet are transferred to the output continuously, the packet delay is significantly reduced.
In EX-DROP-PR, an input (output) is busy if it is matched to an output (input), otherwise it is free. At the beginning of each time slot, each input (output), which was busy (i.e., matched) in the previous time slot, checks its state by checking the corresponding VOQ. If the VOQ has just been emptied, the input (output) changes its state to free and

increments its pointer to one location beyond the matched output (input). Otherwise, the input (output) keeps its state as busy and does not update its pointer. A detailed description of the EX-DROP-PR algorithm is as follows;

1. *Request.* Each free input sends a request to every output for which it has a queued cell. Each busy input sends a request to the matched output.

2. *Grant.* If an output (either free or busy) receives any requests, it chooses one of them in a fixed round-robin order starting from the current position of the pointer. The output notifies each input whether or not its request was granted. Note that the output pointer points to the granted input if the grant is accepted in the accept phase.

3. *Accept.* If an input receives any grant, it sets its state to busy, and accepts one of the multiple grants in a fixed round-robin order starting from the current position of the pointer. The input pointer then points to the matched output.

The EX-DROP-PR runs on top of the DROP-PR scheduler in the sense that is responsible for the flow control (how and when to update the grant/accept pointers) applied.

2.6.1 The DROP-PR scheduling algorithm

In DROP-PR [1], the grant pointers are initially set to different positions and are always incremented by one, and irrespective of the accept/drop outcome. When selecting a cell for input scheduling, DROP-PR takes into account the occupancy of the internal buffers belonging to an output. Note that when a grant scheduler grants an input request, it sends back the grant with an additional priority bit (P), which informs the granted input whether or not the grant comes from an output with empty internal buffers (prioritized output). During the input scheduling phase (second pipeline stage), priority is given to grants where $P = 1$. It is this prioritization given to outputs with empty internal buffers that ensures output j will receive a cell when an input i has a cell destined to it, and thus, drops the assumption we made earlier. In this way, we manage to balance the internal buffers utilization, achieving thus higher throughput, as we give priority to the outputs that are idle at that time slot. The specification of DROP-PR is as follows;

Algorithm 1 DROP-PR

Grant Phase :
All grant pointers, g_j when scheduling VOQs, are initialized to different positions.

. For each output, j, do

. Set CQ_j equals number of non-full internal buffers of output j.

. Set the priority bit, P, to the logic OR of CQ_j entries.

. While there are credits in CQ_j do

- Starting from g_j index, send a grant to the first input, i, that requested this output (set $GQ_{ij} = 1$ and add bit P).
- Decrement CQ_j by one.

. Move the pointer g_j to location $(g_j + 1)(mod\ N)$.

Input Scheduling Phase :
All input pointers, a_i, are initialized to different positions.
For each input, i, do

. Starting from a_i index, select the first non empty VOQ_{ij} for which $GQ_{ij} = 1$ and bit $P = 1$ and send its HoL cell to the internal buffer.

. If no HoL cell is selected, Then

- Starting from a_i index, select the first non empty VOQ_{ij} for which $GQ_{ij} = 1$ and send its HoL cell to the internal buffer.

. Drop the remaining grants (reset GQ: $GQ_{i*} = 0$).

. Move the pointer a_i to location $(a_i + 1)(mod\ N)$.

2.7 The Bounding Methodology

Consider a system of N discrete time queues of infinite capacities, where exogenous arrival processes at different queues can be correlated, and services at different queues are not independent of each other. In this system of queues, customers arrive at one queue, wait, obtain service, and then leave. Let X_n be the row vector of queue lengths at time slot n; that is, $X_n = (x_n^1, x_n^2, ..., x_n^N)$, where x_n^i is the number of customers in queue i at time slot n.
The queue length evolution is described by the expression $x_{n+1}^i = x_n^i + a_n^i - d_n^i$, where a_n^i represents the number of customers arrived at queue i in time slot n from outside, and d_n^i represents the number of customers that depart from queue i in time slot n and leave the system. We assume that $E[(a_n^i)^k] < \infty,\ \forall k > 0$.
Let $A_n = (a_n^1, a_n^2, ..., a_n^N)$ be the vector of the numbers of arrivals at the different queues, and $D_n = (d_n^1, d_n^2, ..., d_n^N)$ be the vector of the numbers of departures from queues. With this notation, the evolution of the system of queues can be described as $X_{n+1} = X_n + A_n - D_n$.
We assume that vectors A_n are independent, that is, $P\{A_{n+1} = \hat{A}|A_i\} = P\{A_{n+1} = \hat{A}\}\ \forall i \leq n$.

Theorem 2.3 **A system of queues is said to be *weakly stable* if, for every**

$\epsilon > 0$, **there exists** $C > 0$ **such that**

$$\lim_{n\to\infty} P\{||X_n|| > C\} < \epsilon$$

Theorem 2.4 **A system of queues is said to be *strongly stable* if**

$$\lim_{n\to\infty} supE[||X_n||] < \infty$$

We assume that the process describing the evolution of the system of queues is an irreducible DTMC, whose state vector at time n is $Y_n = (X_n, K_n)$, $Y_n \in \mathbb{N}^M, X_n \in \mathbb{N}^N$, $K_n \in \mathbb{N}^{N'}$ and $M = N + N'$. Y_n is the combination of vector X_n and a vector K_n of N' integer parameters.
Let H be the state space of the DTMC, obtained as a subset of the Cartesian product of the state space H_X of X_n and the state space H_K of K_n.
If all states Y_n are positive recurrent, the system of queues is weakly stable; however, the converse is generally not true, since queue lengths can remain finite even if the states of the DTMC are not positive recurrent, due to instability in the sequence of parameter vectors $\{K_n\}$.
Note that most systems of discrete time queues of practical interest can be described with models that fall in the DTMC class. The following general criterion for the (weak) stability of systems falling into this class is therefore useful.

Theorem 2.5 **Given a system of queues whose evaluation is described by a DTMC with state vector** $Y_n \in \mathbb{N}^M$**, if a lower-bounded function** $V(Y_n)$**, called Lyapunov function,** $V : \mathbb{N}^M \to \mathbb{R}$**, can be found such that**

$$E[V(Y_{n+1})|Y_n] < \infty$$

and there exist $\epsilon \in \mathbb{R}^+$ **and** $C \in \mathbb{R}^+$ $\forall Y_n$ **such that**

$$E[V(Y_{n+1} - V(Y_n))|Y_n] < -\epsilon \ \forall ||Y_n|| > C \tag{2.1}$$

then all states of the DTMC are positive recurrent, and the system of queues is weakly stable.

Proof. This theorem is a straightforward extension of Foster's Criterion; please see [32][13][33]. □

Note that, interpreting $V(X_n)$ as the system potential energy, Equation 2.7, shows that, for large values of X_n, the system potential energy on average is decreasing. Thus a negative feedback exists which is able to keep the system stable.

2.8 Throughput Analysis of a PBC Switch

For the convenience of analysis, we assume that in each time slot, a PBC switch performs the following steps in sequence; (1) apply EX-DROP-PR as input scheduling, (2) transfer

cells from VOQs to B_js, (3) apply Oldest Cell First (OCF) as output scheduling, and (4) transfer cells from B_js to the outputs.
To prove that the PBC switch is rate stable, it suffices to show that the corresponding fluid model is weakly stable (see Theorem 3 in [15]), i.e., for every fluid model solution (D, T, Z) with $Z(0) = 0$, $Z(t) = 0$ for $t \geq 0$.
A switch operating under a matching algorithm is said to be *rate stable* (equivalent to achieving up to 100% throughput), if, with probability one, $\lim_{n\to\infty} \frac{D_{ij}(n)}{n} = \lambda_{ij}\ i, j = 1, ..., N$ for any arrival process satisfying the strong law of large numbers (SLLN): with probability one, $\lim_{n\to\infty} \frac{A_{ij}(n)}{n} = \lambda_{ij}\ i, j = 1, ..., N$ where, λ_{ij} is the arrival rate at VOQ_{ij}.

Theorem 2.6 **A PBC switch with two internal buffers per output[1] operating under the EX-DROP-PR algorithm with any work-conserving output schedulers is rate stable under any admissible traffic that satisfies SLLN.**

Proof. Consider the fluid model of a PBC switch operating under the EX-DROP-PR algorithm. Let (D, T, Z) be a fluid model solution with $Z(0) = 0$ and $Z_{ij}(t)$ be the total amount of fluid queued at VOQ_{ij} and B_j at time t. From Lemma 1 in [15], to show that the fluid model is weakly stable, it suffices to show that $\dot{Z}(t) \leq 0$ for any $Z(t) > 0$. Intuitively, we need to show that the rate of change of queue length is negative when there are backlogs in queues.
In each time slot n, for any $Z_{ij}(n) > 0$, there are 4 cases:

1. $B_j(n) > 0$ and $output_j$ is matched by the scheduler.
2. $B_j(n) > 0$ and $output_j$ is not matched.
3. $B_j(n) = 0$ and $output_j$ is matched by the scheduler.
4. $B_j(n) = 0$ and $output_j$ is not matched.

Note that the scheduler satisfies the following properties; (1) each input with an edge is matched to an output with the smallest weight, so that the total weight (number of cells at B_*) of a matching is minimum (2) in the case of multiple matchings having the same weight, choose the one with the maximum number of matched outputs.

[1]What is the minimum buffer size required to support persistent, not-contenting connections ? In traditional CICQ design, each of the the N^2 internal buffers is dimensioned to cater for up to one round trip time (RTT) worth of cells. Having in mind that, the RTT in a PBC switch is defined as be the delay between the issuing of a credit by a credit scheduler, until the cell injected owning to that credit exits the switch, and the corresponding credit returns to the same credit scheduler and is re-issued again. The amount of buffering per output is required to be just $\geq B \times RTT$. The input scheduling from the grant scheduling is decoupled and, as so, allows a two-stage pipelined arbitration. In the first pipeline stage, each credit scheduler independently produces a grant and increases by one the corresponding grant pipeline counter. In parallel, each grant scheduler (second pipeline stage) independently selects one among the grants accumulated up to previous time slot inside its grant counters — not yet considering the concurrent outcomes of the credit schedulers. To keep its corresponding link busy, each credit (output) or grant (input) scheduler needs to serve a new request or grant, respectively per time slot. Thus, the maximum available “thinking time" for each scheduler is one time slot, and 2 cell times is the delay of a request going through both credit and grant scheduling. We’ll mathematically prove the sufficiency of $B = 2$ in Chapter 3.

- In cases 1.2.3., let $M_j(t) = \sum_k Z_{kj}t$ be the total amount of fluid destined for $output_j$ and queued at VOQ_{*j} and B_j at time t. $M_j(n+1) - M_j(n)$ is the difference in the number of arrivals at all inputs destined to $output_j$ at $(n+1)$ and the number of departures for $output_j$ at time n. The number of arrivals equals to $\sum_k (A_{kj}(n+1) - A_{kj}(n))$. Because the output scheduler (OCF) is work-conserving and either $B_j(n) > 0$ or $output_j$ is matched by the scheduling algorithm, a cell must leave the switch for $output_j$ at the end of time slot n. So,

$$M_j(n+1) - M_j(n) \leq \sum_k (A_{kj}(n+1) - A_{kj}(n)) - 1 \tag{2.2}$$

  Applying the fluid limit procedure and assuming that the traffic is admissible $\sum_i \lambda_{ij} \leq 1$, $\sum_j \lambda_{ij} \leq 1$

$$\dot{M}_j(n) = \sum_k \dot{Z}_{kj}(t) \leq \sum_k \lambda_{kj} - 1 \leq 0 \tag{2.3}$$

- In case 4., let $L_i(t) = \sum_k Z_{ik}t$ be the total amount of fluid queued at $input_i$ at time t. $L_i(n+1) - L_i(n)$ is the difference in the number of arrivals at $input_i$ at $(n+1)$ and the number of departures from $input_i$ at time n. The number of arrivals equals to $\sum_k (A_{ik}(n+1) - A_{ik}(n))$. Because $Z_{ij}(n) > 0$, $B_j(n) = 0$ and $output_j$ is not matched, $input_i$ must be matched to $output_k$ such that $Z_{ik}(n) > 0$, $j \neq k$ and $B_k(n) = 0$. We claim that k is only matched to i, because if not, we can simply re-match i to j to increase the number of matched outputs without increasing the total weight, and thus contradict to the fact that the scheduling algorithm gives the maximum number of matched outputs. Hence, a cell must leave the switch from $input_i$ at the end of time slot n. So

$$L_i(n+1) - L_i(n) \leq \sum_k (A_{ik}(n+1) - A_{ik}(n)) - 1 \tag{2.4}$$

  Applying the fluid limit procedure and assuming that the traffic is admissible $\sum_i \lambda_{ij} \leq 1$, $\sum_j \lambda_{ij} \leq 1$

$$\dot{L}_i(t) = \sum_k \dot{Z}_{ij}(t) \leq \sum_k \lambda_{ik} - 1 \leq 0 \tag{2.5}$$

Combined all cases, we have $\dot{Z}_{ij}(t) = \sum_{ij} \dot{Z}_{ij}(t) \leq 0$ for any $Z(t) > 0$. Therefore, the fluid model is weakly stable and the PBC switch using a scheduler that satisfies the aforementioned property is rate stable. □

2.9 Is it possible to implement a 1-APRX in a pipelined and distributed fashion ?

In this section, we consider a 1-APRX of MWM algorithm, which is an implementable version of the MWM algorithm. Every K^{th} time slot, a MWM schedule is computed

according to the switch state at time slot K. This schedule is used repeatedly for the next K time slots. We will refer to this kind of scheduling as Bursty MWM ($bMWM$) originally proposed in [31]. The reason why we chose this algorithm is because the weight of the matching keeps changing (due to the partial updates of the crossbar configuration) from being exactly equal to the weight of MWM to that of the weight that is a constant $2KN$ away from the weight of MWM.

Theorem 2.7 **Let W_b denote the weight of schedule obtained by bMWM, and W^* denote the weight of MWM schedule for the same switch state. Then, $W_b \geq W^* - 2KN$.**

Proof. Please see [31]. □

From now on, we will use the terms MWM and 1-APRX interchangeably. Let $t_n \in \mathbb{N}^+$ be a non-defective sequence of regeneration instants (or stopping times) for the evolution of the system of queues, i.e., for any t_n, the evolution of the system following t_n is conditionally independent of the evolution of the system before t_n given the state $Y(t_n)$; moreover, $z_n = t_{n+1} - t_n$.
Now, we say that a PBC switch under EX-DROP-PR follows an *incremental MWM schedule* [34] if at each stopping time a new matching is selected according to the outcome of a MWM algorithm whose weights are proportional to queue lengths. Between two consecutive stopping times t_n and t_{n+1}, partial updates of the switching configuration are allowed. These reconfigurations are performed according to the outcome of a MWM algorithm whose weights are proportional to queue lengths, operating on a subset of input and output ports.

Theorem 2.8 **A PBC switch with two internal buffers per output following an incremental MWM schedule is stable under any admissible i.i.d. input traffic pattern.**

Proof. What we are going to prove is that, the input traffic is formed by variable length frames (occupancy of VOQs) with i.i.d. random size having finite average!
The evolution of the system of discrete-time queues in the PBC switch is represented by a DTMC whose state is defined by the vector of queue lengths X_{t_n}, between consecutive stopping times, the system evolution satisfies the following equation:

$$X_{t_{n+1}} = X_{t_n} + \sum_{i=0}^{z_n-1} (A_{t_{n+i}} - D_{t_{n+i}}) \; where \; z_n = t_{n+1} - t_n$$

Note that all $D_{t_{n+i}}, i < z_n$, refer to the same matching, however, they need not be all equal, since some queue scheduled for transmission at time t_n may become empty before the next stopping time. Remember that, whenever some queues selected for transfer become idle (i.e., either they are or become empty) a partial update of the switching configuration is allowed, according to an MWM algorithm among idle ports, whose weights are proportional to queue lengths.

By using the Lyapunov function $V(X_{t_n}) = X_{t_n}X_{t_n}^T$:

$$E[V(X_{t_{n+1}}|X_{t_n})] - V(X_{t_n}) =$$
$$= E\left[2\sum_{i=0}^{z_n-1}(A_{t_{n+i}} - D_{t_{n+i}})X_{t_n}^T + \sum_{i=0}^{z_n-1}(A_{t_{n+i}} - D_{t_{n+i}})\sum_{i=0}^{z_n-1}(A_{t_{n+i}} - D_{t_{n+i}})^T\right]$$

Thus, under the assumption that $E[A_nA_n^T]$ is finite (which corresponds to assuming finite frame length variances), since also $E[D_{t_{n+i}}D_{t_{n+i}}^T]$ is finite:

$$\lim_{||X_n||\to\infty}\frac{E[V(X_{t_{n+1}}|X_{t_n})] - V(X_{t_n})}{||X_{t_n}||} = \lim_{||X_n||\to\infty}\frac{2E\left[\sum_{i=0}^{z_n-1}(A_{t_{n+i}} - D_{t_{n+i}})X_{t_n}^T\right]}{||X_{t_n}||}$$

Define now $D_\delta = \sum_{i=0}^{z_n-1} D_{t_{n+i}} - z_nD_{t_n}$ as noted before, this difference is due to the fact that some queues may become empty before changes in the switch configuration even when a partial update of the switching configuration is allowed.

$$\frac{E\left[\sum_{i=0}^{z_n-1}(A_{t_{n+i}} - D_{t_{n+i}})X_{t_n}^T\right]}{||X_{t_n}||} = \frac{E\left[\sum_{i=0}^{z_n-1}A_{t_{n+i}}X_{t_n}^T - z_nD_{t_n}X_{t_n} - D_\delta X_{t_n}\right]}{||X_{t_n}||}$$

Wald's Lemma can be applied, since t_n is a sequence of stopping times, therefore obtaining:

$$\frac{E\left[\sum_{i=0}^{z_n-1}A_{t_{n+i}}X_{t_n}^T - z_nD_{t_n}X_{t_n} - D_\delta X_{t_n}\right]}{||X_{t_n}||} = \frac{E[z_n](E[A_n] - D_{t_n})X_{t_n}^T - E[D_\delta]X_{t_n}^T}{||X_{t_n}||}$$

Note that $E[D_\delta]X_{t_n}^T \geq -ME[z_n^2]$, since at most M components of D_δ can be non-null, no component of D_δ can exceed the value of z_n and, finally, a component of D_δ can be non-null only if the corresponding queue length at time t_n is smaller than z_n. Moreover, for each admissible load and non-null queue length vector $(E[A_n] - D_{t_n})X_{t_n}^T < 0$ as proved in [14].

$$\lim_{||X_n||\to\infty}\frac{E[V(X_{t_{n+1}}|X_{t_n})] - V(X_{t_n})}{||X_{t_n}||} = \lim_{||X_n||\to\infty}\frac{2E[z_n](E[A_n] - D_{t_n})X_{t_n}^T - 2E[D_\delta]X_{t_n}^T}{||X_{t_n}||} =$$
$$= 2E[z_n]\lim_{||X_n||\to\infty}\frac{(E[A_n] - D_{t_n})X_{t_n}^T}{||X_{t_n}||} < -\epsilon E[z_n]$$

□

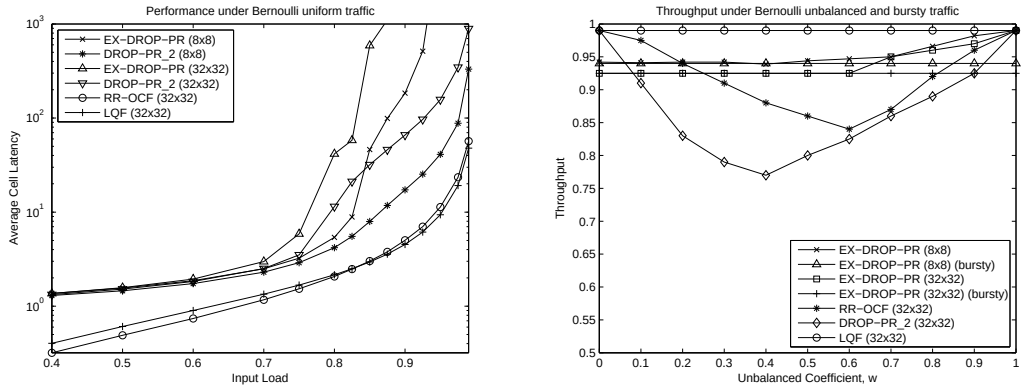
2.10 Performance Results

This section presents the performance evaluations of a PBC switch, using a distinct arbitration scheme; EX-DROP-PR and DROP-PR with two internal buffers per output ($B = 2$), a CICQ and an IQ switch deploying the RR-OCF and LQF schedulers, respectively. The LQF algorithm belongs to the MWM class of schedulers, thus it is optimal

but its complexity is $O(N^3 logN)$, and was proposed in [14]. The traffic models considered have destinations with bursty and unbalanced Bernoulli distributions. We fixed the burst size to be equal to 16 cells[2]. The unbalanced traffic model uses a probability, ω, as the fraction of input load directed to a single predetermined output, while the rest of the input load is directed to all outputs with uniform distribution. Let us consider input port i, output port j, and the offered input load for each input port ρ. The traffic load from input port i to output port j, ρ_{sd} is given by

$$\rho_{ij} = \begin{cases} \rho(\omega + \frac{1-\omega}{N}) & \text{if } i = j \\ \rho(\frac{1-\omega}{N}) & \text{otherwise.} \end{cases}$$

The simulation does not consider the segmentation and reassembly delays.
We analyze the stability of 8×8 and 32×32 PBC switch, by varying the unbalanced coefficient ω. Figure 2.3 shows that EX-DROP-PR provides 94% and 92.5% throughput under the complete range of ω, when the switch size is 8 and 32 respectively. We can see that EX-DROP-PR exhibits higher throughput than DROP-PR and is almost stable for all the values of the unbalanced coefficient. This results in a feasible implementation of a EX-DROP-PR with $B = 2$ PBC switch. The performance of EX-DROP-PR and DROP-PR schedulers is displayed, note that under Bernoulli uniform traffic the VOQs are served in a exhaustive manner less frequently than under any other traffic scheme. Although the EX-DROP-PR scheduling algorithm has a higher average cell latency, and given that all the cells belonging to the same packet are transferred to the output continuously, the overall packet delay is expected to be significantly reduced [3]. In order to have the optimal solution as a reference, the performance of LQF is also plotted.



(a) (b)

Figure 2.3: (a) Performance under Bernoulli uniform arrivals. (b) Throughput performance under Bernoulli unbalanced and bursty traffic.

The high throughput of EX-DROP-PR is the product of increasing or decreasing service for a VOQ in proportion to its received and missed service, respectively. EX-DROP-PR

[2]The choice of an expected duration of 16 cells per burst is arbitrary, but is representative. The same qualitative results are obtained for different burst lengths.

ensures service to the queues with high loads by exhaustively servicing them, and to the other queues by using the pointer update scheme used in both scheduling algorithms that serves inputs in a round-robin fashion.
What has to be commented also is that, when the switch operates with a speedup of 1 and DROP-PR is deployed as a scheduling algorithm, the switch appears to be non-stable when it is loaded by more than 77%. This is to be compared with the stability achieved when EX-DROP-PR is used as a function of the input load. More specifically, EX-DROP-PR increases the stability region of the switch, meaning that the switch can now be loaded by 92.5%. LQF achieves 99% throughput under any circumstances. Moreover, the performance results verify the important property of scalability that a PBC switch with $B = 2$ and EX-DROP-PR (as a scheduler) has, given the observed performance when $N = \{8, 32\}$. There is no getting away from the fact that the proposed switching architecture performs better than a fully-buffered crossbar (having one-cell buffer crosspoint) and scheduling in RR fashion at the inputs and based on the Oldest Cell First (OCF) principle at the outputs in terms of throughput — implying that we utilize the switch fabric more efficiently while we require $(32\times32)-(32\times2) = 960$ one-cell buffering less (when $N = 32$).

2.11 Conclusion

The fact that input and output scheduling in a PBC switch are performed separately allows us to analyze the queues at the inputs of the PBC switch while considering an output scheduler at the buffered crossbar that keeps inputs uninhibited (i.e., an input is able to send one cell each time slot) and to analyze the buffered crossbar while assuming that input scheduling can select any VOQ that has an internal buffer available.
We proved that a PBC switch operating under EX-DROP-PR with any work-conserving output schedulers can achieve 100% throughput for any admissible traffic. We showed that requiring only a two-cell amount of buffering per output, meaning that we fully decoupled the switch size and the amount of internal memory required to achieve maximum throughput. The solution provided in this chapter is $(\frac{N^2}{N\times2})$ times more efficient in terms of memory requirements when compared to the CICQ switching architecture with one buffer per crosspoint [30], while both achieve to maximize throughput. Using analytical techniques based upon Lyapunov functions, we derived original bounds for the average of the lengths of input queues. These bounds complement the previously available theoretical results for IQ and CIOQ cell-based switch architectures, that mostly refer to throughput.

Performance Guarantees in a PBC Switch

3

Most of today's high capacity switches and Internet routers do not provide performance guarantees. This is attributed to their underlaying interconnection topology (i.e. the crossbar) and/or to their impractically complex scheduling algorithms. This chapter derives a study for a PBC switch to practically provide throughput and fairness guarantees. We show how a PBC switch with a two-cell internal buffering per output and a speedup of two can provide guaranteed performance on throughput. We propose a scheduling algorithm, named AF-DROP-PR, that enables us to derive bounds on the average cell latency and on the guaranteed fairness among the competing flows. AF-DROP-PR maximizes the total throughput while guaranteeing service fairness among all flows. We explore how the allocation bandwidth of individual flows is tradedoff with the overall fairness index. We conjecture that the PBC switch, with its pipelined and distributed AF-DROP-PR scheduling, is attractive not only for providing performance guarantees, but also for its high capacity and low cost.

3.1 Towards Buffered Crossbars

IQ switch architectures with VOQs scale up to very high speeds and have been a subject of intense research in the past decade. IQ switches require switch matrix scheduling algorithms to match input ports to output ports. In Chapter 1, we introduced the CIOQ switch. CIOQ switches are widely used, partly because they have localized buffers that enable these routers to be easily upgraded. However they are hard to scale because the scheduler in a crossbar-based CIOQ switch implements complex stable marriage algorithms [11]. With the continued increase in density of VLSI, sufficient buffering at crossbar crosspoints for one cell has become feasible to implement. The scheduler's job much simpler by introducing a small amount of buffering in the crossbar, because when a cell is switched, it can wait in the buffer; it doesn't have to wait until both the input and output are free at the same time. Now, the scheduler doesn't have to resolve two constraints at the same time, reducing this way the complexity of the scheduler from $O(N^2)$ to $O(N)$ and removing the need for a centralized scheduler. In a buffered crossbar, the input schedulers (independently and in parallel) pick a cell from their inputs to be placed into an empty crosspoint. There are N input and N output schedulers can be distributed to run on each input and output independently, eliminating the single centralized scheduler. They can be pipelined to run at high speeds, by decoupling the accept from the grant phase during the switch's scheduling cycle, making buffered crossbars appealing for high-speed routers.
Obviously, simplifying the scheduler requires a more complicated crossbar that is capable of maintaining N^2 cell buffers on-chip. Buffered crossbars are possible because of large improvements in gate count; nowadays, crossbar capacity is limited by I/O speed

(number of pins required to get data on and off the chip), not die size [19]. This leaves room on the die for a buffer at each crosspoint.

Improvements in process technology, and reductions in geometries, mean that the logic required for N^2 crosspoints is small compared to the chip size required for N inputs and N outputs. The chips are pad-limited, with an under-utilized die. A buffered crossbar can use the unused die for buffers, and further improvements in on-chip eDRAM memory technology [35] allow for storing a large amount of memory on-chip. With upcoming 45 nm chip technology, ̃128Mb of eDRAM can easily fit on an average chip. This can easily support a 256×256 router (with 256-byte cells), making a buffered crossbar a practical proposition[1].'' [2]

3.2 Mandating Guarantees for High-Speed Switches

Our objective would be to build a high-speed switch that is capable of giving performance guarantees. There are no switches (having bandwidth greater than 10 Gb/s) that do provide guarantees. There are typically two types of performance guarantees; statistical (the most common example being 100% throughput) and deterministic (the most common examples being work-conservation and delay guarantees) [2].
If someone buys a router with less than 100% throughput, he couldn't be in the position of knowing what the true capacity of the switch actually is. This makes it particularly hard for network operators to plan a network and predict its network performance.

3.2.1 A note about work-conservation

- A switch is said to be **work-conserving** if an output will always serve a cell when a cell is destined for it.

If a switch is work-conserving, then it has 100% throughput, because the outputs cannot carry a higher workload. Additionally, the expected cell latency is minimiized, since, on average, cells leave earlier in a work-conserving switch compared to any other switch.

3.3 Related Work

Maximum size matching (MSM) finds the match containing the maximum number of edges, thus it can maximize the instantaneous bandwidth of the switch. It has been proved [14] that MSM can lead to instability and unfairness (if ties are broken randomly). In [26], an MSM scheduling algorithm that employs a weighting scheme, called Longest Port First (LPF), achieves 100% throughput in an IQ switch. The author in [37] devised a scheduler that uses the concept of adaptable-frame size coupled with round-robin arbitration for a buffered crossbar with one-cell crosspoint buffers. This scheme does not compare any weights or priorities and is able to provide nearly 100% throughput

[1] Some detailed considerations regarding the implementation of buffered crossbars are described in [36].

under the unbalanced traffic model. The characteristic of this scheduler is that it is based upon maintaining a balance between two competing interests; trying to maximize total throughput, while at the same time allowing all flows at least a minimal level of service. A study of the fairness properties and the factors that affect the stabilization delay after a change in the offered load and the weight factors in buffered crossbars is presented in [38]. In [10] [39] rate and delay guarantees are provided using a complex and rather impractical scheduler. Dai and Prabhakar [15] studied how guaranteed performance on throughput can be obtained using a centralized MSM algorithm and a speedup of two.
Our work differs from previous art both at the architectural and the scheduling level. The PBC architecture relies on separate internal buffers per output port interconnected by parallel multiplexers, dropping this way the requirement of an expensive shared memory (in terms of the memory access rate) per output. On the scheduling level, the synchronization between the grant and the accept schedulers is avoided by means of a distributed and pipelined scheduling algorithm.
While emulating a Push-In-First-Out (PIFO) OQ switch is usually the way to show that a switch can provide guarantees, this is accomplished either by using a complex scheduler or by using an excessive memory bandwidth inside the crossbar. Our results show, that the underlying switching architecture is capable of achieving the same average cell latency and be completely fair, as an OQ switch but, the average cell latency must be tradedoff with the optimal WMMF rates of the flows by regulating the amount of service that a flow receives.

3.4 Work Conservation without Emulation

A work-conserving switch gives a deterministic performance guarantee — it always keeps its outputs busy. For a switch to be work-conserving, we only need to ensure that its outputs are kept busy. So, in such a switch, the order of departures of cells from outputs is irrelevant. It is sufficient that some cell leave the output at all times.

3.4.1 On the speedup required for achieving 100% throughput

An OQ switch is able to provide QoS by scheduling the departure of cells accordingly to latency constraints and ensuring that its outputs never idle (given that there are cells destined to them). We will require that the solution on the speedup: (1) allows a PBC switch to achieve approximately the same average cell latency as an OQ switch and also to provide 100% throughput, (2) this is accomplished for any arbitrary input traffic pattern and (3) is independent of the switch size.

Theorem 3.1 **A PBC switch can achieve 100% throughput with a speedup of two for any Bernoulli i.i.d. admissible traffic.**

Proof. Similar to [20], we define a super queue, $SVOQ_{ij}$ to track the evolution of each VOQ_{ij} . Let $SVOQ_{ij}$ denote the sum of the cells waiting at input i, and the cells destined to output j, comprised by the cells that reside in VOQ_{ij} and in B_j.

$$SVOQ_{ij} = \sum_k VOQ_{ik} + \sum_k (VOQ_{kj} + CQ_j)$$

What has to be shown is that the expected value of every super queue is bounded. Let's assume, for now, that if VOQ_{ij} is not empty and j is an output with empty internal buffers ($CQ_j = B_j$), input i will send the HoL cell of VOQ_{ij} to the internal buffers of output j. We'll drop that assumption in the next section. There are two cases :

1. $0 < CQ_j \leq B_j$: output j will receive a cell, so $\sum_k (VOQ_{kj} + CQ_j)$ decreases by 1.
2. $0 \leq CQ_j < B_j$: Input i will send a cell, so $\sum_k VOQ_{ik}$ decreases by 1.

The number of cells sent to B_j during a time slot is equal to $(B_j - CQ_j)$, so $\sum_k VOQ_{kj}$ decreases by $(B_j - CQ_j)$ but CQ_j increases by the same quantity. Hence, $\sum_k (VOQ_{kj} + CQ_j)$ does not change after the input scheduling phase. During the output scheduling phase, output j will receive one more cell and $SVOQ_{ij}$ will decrease by two per time slot. It is because the expected change in $SVOQ_{ij}$ is negative over a time slot, given the traffic is admissible and Bernoulli i.i.d., the expected value of $SVOQ_{ij}$ is bounded. □

It is important to mention at this point that the round-trip delay (RTT) is defined as the time from when a credit is generated until it is consumed by an input port and its associated cell is transferred to the internal buffer. Note that with our PBC request-grant protocol, only one RTT window is enough to achieve full-line rate to any requesting input. This is because of the shared credit queue (CQ) maintained by each grant scheduler and controlling the access from all inputs. As a result, the amount of buffering per output is required to be just $\geq B \times RTT$, and each output j must have an internal buffer of size B_j greater or equal to one RTT worth of cells. Next, we'll show that two cell times is the delay of a request going through both credit and grant scheduling.

3.4.2 A work-conserving PBC switch

For a switch to be work-conserving, what is necessary is to ensure that its outputs will always serve cells as long as there are cells destined to them. Cell delay on average, is minimized, since cells leave earlier in a work-conserving switch than in any other switch. In the following, we'll find the conditions under which a PBC switch is work-conserving. In the context of switching theory, the pigeonhole principle, seminally described in [2], will be used in order to dictate these conditions. Note that, in a work-conserving switch the order of departure of cells is insignificant.

Theorem 3.2 **A PBC switch is work-conserving with a total memory bandwidth of $2 \times N \times R$.**

Proof. (Using constraint sets.) Let's denote as decision slots to be the slots that comprise a time slot (if there are N cells to be scheduled, N decisions have to be made). If we denote the speedup with S, the total memory bandwidth is $S \times N \times R$, where $S = 2$ as shown in the previous section. We define two constraint sets; when cells are written/read to/from the internal buffers of an output j (B_j).

- B_j Write Set (B_jWS): The internal buffers of output j are busy for $\lceil B_j/S \rceil$ timeslots, when a cell is written into B_j. Let B_jWS be the set of internal buffers that store a new cell. In other words, B_jWS is the set of internal buffers that

have initiated a write operation in the previous $\lceil B_j/S \rceil - 1$ decision slots. Hence, $|B_jWS| \leq \lceil B_j/S \rceil - 1$.

- B_j Read Set (B_jRS): The internal buffers of output j are busy for $\lceil B_j/S \rceil$ timeslots, when a cell is read from B_j. Let B_jRS be the set of internal buffers that read a new cell. In other words, B_jRS is the set of internal buffers that have initiated a read operation in the previous $\lceil B_j/S \rceil - 1$ decision slots. Hence, $|B_jRS| \leq \lceil B_j/S \rceil - 1$.

Assuming that VOQ_{*j} have cells destined to output j, and the same case holds for all outputs. To keep all outputs busy, when input scheduling, the internal buffers of the switch must meet the following constraints;

1. The internal buffer B_j must not be busy writing a cell. So $B_j \notin B_jWS$.
2. The internal buffer B_j must not be busy reading another cell; i.e., $B_j \notin B_jRS$.

Choosing an internal buffer B_j means that the following constraints must be met;

$$B_j \notin B_jWS \wedge B_j \notin B_jRS$$

A sufficient condition to satisfy this is :

$$B_j - |B_jWS| - |B_jRS| > 0 \Longrightarrow B_j - 2(\lceil B_j/S \rceil - 1) > 0$$

This is satisfied if $B_j \geq 2$. □

3.5 The Adaptive Frame Prioritized DROP scheduler (AF-DROP-PR)

We propose a round–robin frame-based input scheduling algorithm. The output scheduling used is based on FCFS policy. This arbitration scheme is round-robin based. A frame is related to a VOQ, and it is the number of one or more cells in a VOQ that are eligible for arbitration (as allowed by the flow-control mechanism). A new frame gets assigned a size, in number of cells, each time the previous frame is serviced. The frame size is determined by the cell occupancy of the VOQ at time of the frame service completion. This is called *captured-frame size*. A VOQ is said to be in *on-service* status if the VOQ has a frame size of two or more cells and the first cell of the frame has been transmitted. An input is said to be on-service if there is at least one on-service VOQ. A VOQ is said to be *off-service* if the last cell of the VOQ's frame has been sent, or no cell of the frame has been sent to the internal buffers. Note that for frame sizes of one cell, the associated VOQ is off-service during the matching of its one-cell frame. An input is said to be off-service if all VOQs are in off-service status. At the time t_c of selecting the last cell of a frame of VOQ_{ij}, the next frame is assigned a size equal to the occupancy of VOQ_{ij}. Cells arriving at VOQ_{ij} at any time t_d, where $t_d > t_c$, are not considered for selection until the current frame is totally served and they are included in a new captured frame. For each VOQ, there is a captured frame-size counter, CF_{ij}[2]. The value of CF_{ij} indicates

[2]We called captured frame size as it is the equivalent of having a snapshot of the occupancy of a VOQ at a given time t. The occupancy is then considered for determining the frame size.

the frame size, that is, the maximum number of cells that a VOQ_{ij} has as candidates in the current and future time slots, one per time slot. CF_{ij} takes a new value when the last cell of the current frame of VOQ_{ij} is selected. CF_{ij} decreases its count each time a cell is selected, other than the last.
The output scheduler is pointer persistent, meaning that if an accepted grant g_j points to input i, it keeps pointing to same input unless VOQ_{ij} is set to off-service. The specification of AF-DROP-PR is described below; Although the scheme presented above is

Algorithm 2 AF-DROP-PR

Grant Phase :
All output pointers, g_j, are initialized to different positions.
For each output, j, do

. Set CQ_j equals number of non-full internal buffers of output j.

. Set the priority bit, P, to the logic OR of CQ_j entries.

. While there are credits in CQ_j do

- If output j is paired with an input i (VOQ_{ij} is in on-service status), send a grant to input i (set $GQ_{ij} = 1$ and add bit P).
- Else, starting from g_j index, send a grant to the first input, i, that requested this output (set $GQ_{ij} = 1$ and add bit P).
- Decrement CQ_j by one.

. If VOQ_{ij} is in off-service status, move the pointer g_j to location $(g_j + 1)(mod\ N)$.

Input Scheduling Phase:
All input pointers, a_i, are initialized to different positions.
For each input, i, do

. If input i is paired with an output j (VOQ_{ij} is in on-service status), send its HoL cell to the internal buffer.

. Else, starting from a_i index, select the first non empty VOQ_{ij} for which $GQ_{ij} = 1$ and bit $P = 1$ and send its HoL cell to the internal buffer.

. If no HoL cell is selected, Then

- Starting from a_i index, select the first non empty VOQ_{ij} for which $GQ_{ij} = 1$ and send its HoL cell to the internal buffer.

. Drop the remaining grants (reset GQ: $GQ_{i*} = 0$).

. If VOQ_{ij} is in off-service status, move the pointer a_i to location $(a_i + 1)(mod\ N)$.

a mixture of the DROP-PR [1] and the RR-FO [37] scheduling algorithms, it provides overall good performance while the amount of service a flow receives is almost proportional to that a flow demands, it has two drawbacks; (1) it does not discriminate the inputs when accepting grants and one of them is part of virtual circuit and (2) it is not a maximum size algorithm concerning the number of virtual circuits that establishes. Next, we'll describe how these limitations can be overcome in order to fully exploit the underlying switching architecture when servicing VOQs that are in on-status and ensure that the algorithm will do its best in order for a flow to be serviced exactly as much as it demands as soon as possible.
We say that a virtual circuit between input i and output j is established when VOQ_{ij} is

in on-status. Given that $B_j = 2$ and $S = 2$, the first limitation is addressed by balancing the creation of the virtual circuits so that an output j should not be able to establish two virtual circuits in one time slot. This is accomplished simply by dropping the rest of the inputs that received a grant. The second limitation is addressed by adopting a prioritization mechanism similar to DROP-PR, but now considering to create up to N virtual circuits among all inputs. In other words, as DROP-PR tries to keep busy all the outputs of the switch using the priority bit mechanism, the same technique is applied in order to keep busy all outputs while making certain that each output is paired to a different input and this input-output pair is a virtual circuit.
Subsequently, we'll explore how significant is the role of the captured-frame size both in the bandwidth a flow allocates and the fairness index and in the average cell latency of the traffic flowing through the PBC switch.

3.6 Bounded Bandwidth Allocations

A common approach to provide throughput guarantees is to show that a switching architecture along with its scheduling algorithm is capable of emulating a PIFO OQ switch. It is because WRR/WFQ scheduling functions in a PIFO manner. Mimicking a PIFO OQ switch means that a cell must be switched out to the output port earlier than the departure time of the counterpart cell in the OQ switch. We argue that the latter can be accomplished by, using only a two-cell amount of buffering per output port in a PBC switch with a speedup of two and employing the AF-DROP-PR scheduling algorithm that services the flows proportionally to their demand. The cost that we have to pay though is the time window in which this is accomplished is larger compared to the work in [20]. However, the time window is controlled by upper–bounding the frame size of a VOQ, CF_{ij}. We showed that when B equals two and with a speedup two the architecture in question achieves 100% throughput and the same average cell latency as an OQ switch. The phenomenon according to which the fair share of a flow at the output differs from its fair share at the input, is surpassed by establishing virtual circuits for as much time as is required in order to service that flow having its fair share in mind. Moreover, the use of virtual circuits alleviated the blockings between inputs contending for the same output. In this section, we'll study the fairness properties of the proposed switching architecture and the delay bounds achieved based on the guaranteed rate per flow.
We'll adopt the definitions originally presented in [40]. In a switch where flows make unequal demands for bandwidth, fairness is measured by closeness of the allocations to respective demands. If d_i is the demand of the i^{th} user and a_i is the corresponding allocation, then, the fraction of demand of the i^{th} flow is;

$$x_i = \begin{cases} \frac{a_i}{d_i} & \text{if } a_i < d_i \\ 1 & \text{Otherwise}^3 \end{cases}$$

The flow perception of fairness equals

$$f(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{x_i}{x_f}$$

where the fair allocation mark x_f equals

$$x_f = \frac{\sum_{i=1}^{N} x_i^2}{\sum_{i=1}^{N} x_i}$$

Thus, the scheduling algorithm is (x_i/x_f) fair concerning the i^{th} flow. When the enhanced AF-DROP-PR is used as a scheduling algorithm, the actual allocation of the bandwidth refers to the captured–frame size of VOQ_{ij}, CF_{ij}.
In the following, we'll consider the allocation of the excess bandwidth [40].

Theorem 3.3 **If each user is given an additional amount c of the resource, their individual perception of fairness increases and so does the overall fairness index.**

$$f(x_1 + c, x_2 + c, ..., x_n + c) \geq f(x_1, x_2, ..., x_n)$$

If a single user j is given a small allocation Δx_j without changing other allocations, the new allocation is more (less) than before iff j is a discriminated (favored) user, i.e.,

$$f(x_1, x_2, ..., x_{j-1}, x_j + \Delta x_j, x_{j+1}, ..., x_n) > f(x_1, x_2, .., x_{j-1}, x_j, x_{j+1}, ..., x_n) \textbf{ if } x_j < x_i$$

and

$$f(x_1, x_2, ..., x_{j-1}, x_j + \Delta x_j, x_{j+1}, ..., x_n) < f(x_1, x_2, .., x_{j-1}, x_j, x_{j+1}, ..., x_n) \textbf{ if } x_j > x_i$$

What the above theorem states is that, as a particular flow's bandwidth allocation is increased from zero, the fairness at first increases, and then, it reaches a maximum after which additional bandwidth allocation to the same flow has as a consequence the rest of the flows to perceive unfairness. What has been just described is the optimal WMMF allocation metric.
A minimum level of fairness is guaranteed when we restrict the individual bandwidth allocations [40].

Theorem 3.4 **If $x_{min} \leq x_i \leq x_{max}$ for all i and $x_{min} > 0$, then the fairness is guaranteed to be above a certain lower bound, and**
a. The minimum fairness occurs when a fraction of γ of the users (i.e. γ n users in all) receive x_{min} and the remaining receive x_{max}. Here

$$\gamma = \frac{K}{K+1}\textbf{, where } K = x_{max}/x_{min}$$

[3] Allocating more bandwidth to a flow than what it demands is vain.

b. Fairness

$$f \geq \frac{4K}{(K+1)^2} \tag{3.1}$$

Let's quantify the tradeoff that is introduced; for a 32×32 switch if the average VOQ_{*j} occupancy is around 50 cells ($d_{ij} = 50$) and we upper-bound the fraction of demand of the ij-flow to $N = 32$ then x_i and x_f decrease by $\frac{(50-32)}{32\times50} = 0.0112$ as far as the time window in question is concerned, but the flow perception of fairness stays the same. Note that by upper-bounding the fraction of demand, we force the restriction $K \in [0.0312, 0.0200]$, and we bound the minimum fairness achieved to $f = 0.9518$. The stabilization delay depends on the frame size and the frame size difference among VOQ_{*j}, meaning that smaller frame sizes and larger frame size differences incur faster stabilization.

3.7 Implementation Complexity

The implementation complexity of AF-DROP-PR is low because of the following reasons; (1) a two-cell internal buffer per output is sufficient to make the switch deliver high performance, (2) the arbitration scheme is round-robin based. AF-DROP-PR performs no comparisons among different queues. Arbiters do not differentiate queues as there are no priorities or weights considered. The provision of the CF counter to a queue is the major hardware addition in AF-DROP-PR compared to the implementation of DROP-PR. The CF counter, as well as the arbiter pointers are updated, at most, once in a time slot in both schemes.
The round-robin schemes presented here have lower complexity than weight-based schemes. Weight-based schemes, such as those that consider cell age or queue occupancy length, need to perform comparisons among all contending queues, which can be a large number, to select the proper queue. AF-DROP-PR follows a predetermined order (round-robin) and checks the flag (on-service flag) which is updated in previous time slots. Hardware-wise, round-robin schemes have no magnitude comparators needed in weight-based schemes.
Alone the PBC switching architecture has to maintain up to B separate buffers per output, which run at the same bandwidth as the external line rate. This is done in order to maintain the low memory bandwidth requirement and it is to be compared with the shared internal buffers requiring a memory access rate of $N \times R$ in [20]. However, maintaining B separate buffers per output mandates the use of up to B×(N-to-1) parallel multiplexers per output. The demultiplexor selects an internal buffer and sends the arriving cell to that buffer, where it is queued until it departs.

‘‘Hardware chips today are already limited by interconnect pins. Using current 65nm ASIC fabrication technology, any chip with over 1500 interconnect pins becomes prohibitively expensive. If each memory has approximately 50 pins (fairly typical of most commodity memory today), it is hard for a single ASIC to interface to (and load- balance over) >30 memories.’’ [2]

Given that; (1) the crossbar chip is bound by I/O count and power consumption

and not by the crosspoints logic, implying the underutilization of the crossbar chip die [20][41], (2) using the crossbar chip extra (unused) logic for either N^2 distributed or B.N shared internal memories running B times the external line speed results is excessive power consumption and can be costly and (3) on-chip wires are inexpensive [42] and therefore meeting the bandwidth goal can be achieved by adding more wires and multiplexers in parallel. We believe that the cost and feasibility of the PBC (with B×N separate internal buffers and N×B×(N-to-1) parallel multiplexers), where $B = 2$, is lower than that of traditional CICQ design (with N^2 internal buffers and N×(N-to-1) parallel multiplexers).

3.8 Conclusion

By introducing a small amount of buffering in the crossbar and employing a frame-based scheduling algorithm, the resulting PBC switching architecture is simple as it uses distributed and pipelined schedulers and is scalable as it requires much less memory bandwidth when compared to fully buffered crossbars. Note that the memory access rate of the internal buffers is decoupled of the switch size. AF-DROP-PR was devised to fully exploit the underlying architecture, yielding full output utilization and providing WMMF bandwidth allocations in a relaxed time window that depends on the frame sizes assigned to the flows and on the frame-size differences among them. Bounds on the average cell latency and the overall fairness index were derived, revealing this way a tradeoff between these two metrics.
This chapter conducts the first study on how to guarantee performance in PBC switches. Given the high potential of buffered crossbars, our work addresses the scalability issue of crossbar based routers.

A PBC Switch Supporting Integrated Unicast and Multicast Traffic

4

Much of recent Internet growth is attributed to the increasing convergence of media services (broadcasting, cable TV and on-demand multimedia) to packet-based networks with an evolving number of online multi-party applications such as online gaming, IPTV, video teleconferencing etc. [43]. All these applications and services carry point-to-multipoint (multicast) traffic and have now to coexist alongside conventional point-to-point (unicast) traffic. With this increasing shift in the Internet traffic nature, the Internet infrastructure (high speed IP routers, Gigabit Ethernet switches and Frame Relay switches) has remained the same. Therefore, in order to keep pace with growing integrated Internet traffic, there is an urgent need for network nodes to support simultaneous and concurrent unicast and multicast traffic flows.
The growing proportion of multicast traffic on the Internet is stressing the need for efficient multicast support alongside the unicast traffic. Only little has been done on the support of integrated unicast and multicast traffic flows. This chapter conducts the first study on integrated unicast and multicast traffic support in the PBC switching architecture that has recently been proposed and shown to be a good compromise between both the unbuffered and fully buffered crossbar switch architectures. We propose two integrated schedulers and test their performance under a wide range of input traffic and switch settings. Our performance study shows that the PBC switch with its integrated scheduler can efficiently handle integrated unicast and multicast traffic flows, while maintaining the advantages of both the unbuffered and the fully buffered crossbar architectures.

4.1 Background: Multicast Traffic

So far we have focussed on how a switching fabric can efficiently transfer unicast packets. But it is becoming increasingly important for a switch to efficiently support multicast traffic. In this chapter, we consider how our switching fabric can be modified to support multicast traffic at very high performance.
A simple way to service the input queues is to replicate the input cell over multiple cell times, generating one output cell per time slot. However, this approach has two disadvantages; (1) Each input must be copied multiple times, increasing the required memory bandwidth. (2) Input cells contend for access to the switch multiple times, reducing the bandwidth available to other traffic at the same input.

‘‘Unicast traffic has a very ‘‘nice’’ property; to approach saturation on all the output ports of the switch, it is necessary to receive cells at all the N inputs. Thus, under unicast traffic, the instantaneous aggregate

load of the switch is always less or equal to than the total admissible capacity of the switch. On the contrary, with multicast traffic, cells arriving at just one input can bring all the switch outputs close to saturation; this implies that when all the switch inputs are active, for some time periods the instantaneous aggregate switch load can be N times the total admissible capacity of the switch (consider for example the possibility of sequences of broadcast cells arriving at all inputs); in other words, multicast traffic, even if admissible, can temporarily ''flood'' the switch, and cannot be scheduled with the approach proposed in [10].'' [34]

Higher throughput can be attained if we take advantage of the natural multicast properties of a crossbar switch. In the PBC switch (which we design to copy an input cell to any number of idle outputs in a single cell time), we maintain two types of queues; unicast cells are stored in VOQs, and multicast cells are stored in a separate multicast queue (MQ). As a simple example of how a crossbar switch can support multicast, consider the switch shown in Figure 4.1[1]. Queue MQ_1 has an input cell destined for outputs $\{1, 2, 3, 4\}$ and queue MQ_N has an input cell destined for outputs $\{3, 4, 5, 6\}$. The set of outputs to which an input cell wishes to be copied is called the fanout of that input cell. For example, in Figure 4.1, the HoL cell of the multicast queue is said to have a fanout of 4, meaning that, the single input cell at the head of queue MQ_1 will generate 4 output cells.

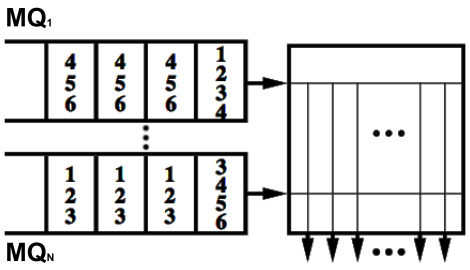


Figure 4.1: A $N \times N$ crossbar that supports multicast.

Multicast cells, arriving at input port, i, are queued in MQ_i. The switch model is similar to the PBC switch architecture presented in Chapter 2, but the input integrated scheduler (IIS) used in our current switch model resides in the input port, as opposed to the input schedulers for the conventional PBC which are embedded inside the crossbar fabric. The reason for this is the flow control mechanism that is now required to carry N bit signals for the multicast cells fanout destinations (informing the otherwise embedded input scheduler about the arrival of a new multicast cell). However, the flow control is now carrying the grant signals sent by the grant schedulers, GS_j to an input, equalling to N bit flow control signals per input.

[1]Each input maintains a special FIFO queue for multicast cells.

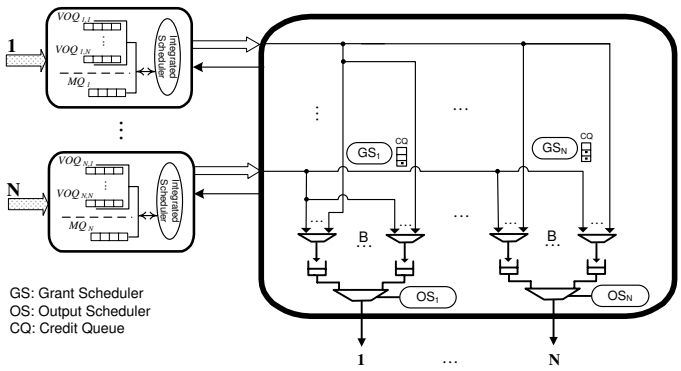


Figure 4.2: The PBC switching architecture with integrated scheduling.

4.1.1 Multicast scheduling disciplines

The basic strategies normally used by a scheduling discipline are:

- **No fanout splitting** — This discipline may favor cells with small fanout, because any multicast cell is transferred through the switching fabric once, only when all fanout destinations can be reached in the same time slot. If any of the fanout destinations cannot be reached because the multicast cell loses contention for an output port, the cell cannot be transferred, and it will contend again in next time slot, thus this strategy is non-work-conserving.

- **Fanout splitting** — Only those fanout destinations that could not be reached in previous time slots are considered in the next time slot. Multicast cells may be delivered to output ports over a number of time slots.

In [44] it was proved that the multicast scheduling problem is $NP-hard$, both with and without fanout splitting.
Another issue to address when both unicast and multicast are present is fairness. A traffic type must not be allowed to monopolize switch resources; however, it is also important to guarantee that all connections of a given traffic type receive service. A multicast scheduling policy is said to be *fair* if each input cell is held at the HoL for no more than a fixed number of cell times (this number can be different for different inputs). This fairness constraint can also be thought of as a starvation constraint. It is up to the scheduling policy to determine for how long a multicast cell will remain as a HoL cell of the multicast queue, the term that best describes this, is;

- **Residue:** The residue is the set of all output cells that lose contention for output ports and remain at the HoL of the input queues at the end of each cell time. Given a set of requests, every work-conserving policy will leave the same residue.

4.2 Related Work

While there has been extensive research work on either unicast traffic support or pure multicast traffic support by backbone routers and packet switches, only little has been done on the design of switching architectures and scheduling of integrated schedulers for both unicast and multicast traffic flows. The few existing results are mainly conducted on crossbar-based switches, due to their low cost, relative scalability and particularly their intrinsic multicast capabilities. The scheduling algorithms employed were a combination of earlier algorithms, proposed for unicast and multicast, now integrated into one algorithm. Examples include the ESLIP algorithm [45] that has been proposed for the IQ crossbar architecture and the MURS algorithm [46] proposed for the CICQ switching architecture. The input queueing structure adopted for unicast traffic is the widely known virtual output queueing (VOQ), since it avoids the HoL blocking problem [47]. As for the multicast proportion of traffic, either 1 or few multicast queues are used, since avoid the HoL problem would require maintaining up to $(2^N - 1)$ FIFO queues per input, where N is the number of output ports of the switch [48].
A multicast matching consists of edges that connect a single input to one or more outputs. The problem has been studied from a theoretical point of view in [49], and in [44] its computational hardness is established. In [50] the optimal scheduling discipline is defined, but neither the discipline itself nor the assumed queuing architecture are practically implementable. Nevertheless in [51] the authors propose algorithms with reasonable complexity and relatively good performance.

4.3 Multicast Scheduling in PBC switches

4.3.1 The mcastDROP scheduler

To support multicast, we need a new scheduling algorithm — one that can examine the contents of the multicast queues at each input, and select which output cells will be delivered in each time slot. For this purpose, we have ported the ESLIP algorithm [45] to the PBC switching architecture resulting in the mcastDROP scheduler.
At the beginning of each time slot, the mcastDROP scheduler examines the contents of the multicast queue and all inputs and outputs are initially unmatched. To keep track of which connection to favor, all the outputs share a common pointer for multicast cells, g_M. The flow control is the same as in DROP-PR and the scheduling decisions are taken based on the following three steps:

- *Request.* Each input sends a request to every output for which it has a queued multicast cell.
- *Grant.* If there are multicast requests, it chooses the input that appears next in a fixed, round-robin schedule starting from the global multicast pointer g_M. The output notifies each input whether or not its request was granted.
- *Accept.* Each input port services all the grants that it receives, choosing the output that appears next in a fixed, round-robin schedule starting from the accept pointer, a_i. If this connection completes the fanout, the pointers a_i and g_M are updated.

4.3.2 The mcastWBA scheduler

The WBA scheduler, proposed by Prabhakar [51], is a state-of-the art algorithm for multicast scheduling for IQ switches. In order to port the WBA algorithm to the PBC switch, we used the same flow control as in DROP-PR. This algorithm works by assigning weights to input cells based on their age and fanout at the beginning of every cell time. Once the weights are assigned, each output chooses the heaviest input from among those subscribing to it. It should be stated here that if an algorithm aims to be fair, it may not achieve the best possible residue concentration[2], and therefore sacrifices throughput. Each scheduling cycle consists of the following steps:

- *Request.* The weight of the HoL multicast cell at an input port is $(age - 2 \times fanout)$, where age is the number of cycles for which the multicast cell has been held at HoL. The input sends a request to output ports in the HoL multicast cell's destination set along with this weight.
- *Grant.* Each output port looks at all the requests that have been made to it and grants the request with the largest weight.
- *Accept.* Each input port services all the grants that it receives, based on the decision of IIS_i.

This algorithm can be divided into two main parts; (1) every input computes a request weight, and (2) every output chooses the input making a request with the highest weight. Since calculating an input's weight does not depend on the weight of any other input, this may be performed in parallel. Similarly, each output may choose the input with the highest weight independently, and may be performed in parallel.

```
‘‘Because the single FIFO queuing architecture causes HoL blocking,
therefore the schedulers must carefully choose which inputs to serve
to mitigate its effects.  For example, it is shown in [51] that
‘‘concentrating the residue’’ at every time slot (which roughly means
providing full service to as many inputs as possible) greatly helps in
draining the queues fast.  This is the reason why the performance of
multicast scheduling algorithms varies considerably.’’  [52]
```

4.4 Integrated Scheduling in PBC switches

We opted for an integration policy that gives equal priority to multicast and to unicast traffic. Hence, the algorithm implemented can be formulated as follows; (1) start with an empty matching, (2) if the time slot is even, schedule the MQs and then schedule the VOQs, (3) if the time slot is odd, schedule the VOQs and then schedule the MQs.
As the two schedulers run in parallel and independently, the matchings they produce in general are overlapping, meaning that they have conflicting edges. Our integration policy

[2] A multicast scheduling policy is said to be *concentrating* if, at the end of every cell time, it leaves the residue on the smallest possible number of input ports.

decides which unicast and multicast edges will be part of the integrated matching, by prioritizing the two schedulers (based on the parity of the time slot) in order to obtain a consistent crossbar configuration. Our motive behind this kind of integration is to achieve good link utilization, any remaining resources can be assigned to either unicast or multicast. The integration policy preserves the matching produced initially by the prioritized scheduler, but tries to enlarge it by adding edges using the non-prioritized scheduler for that time slot. Prioritizing a request for an input-output pair simply translates ANDing it with a bitmask of 1s or 0s depending on the parity of the time slot.

4.4.1 The DROP_mix and DROP-WBA integrated schedulers

Similar to previous integrated algorithms, DROP_mix and DROP-WBA are composed of a unicast scheduler and a multicast scheduler integrated together. We used the DROP-PR scheduler [1] for unicast scheduling and a FIFO multicast scheduling similar to [48]. We chose DROP-PR since it has good delay performance by prioritizing outputs with empty internal buffers, resulting in more balanced internal buffers occupancies. Our integration policy regarding to scheduling combines DROP-PR with either mcastDROP or mcastWBA. During even time slots, unicast traffic has priority, while multicast traffic is prioritized during odd time slots. The specification of DROP_mix along with DROP-WBA appears below.

4.5 Performance Results

This section presents the simulation study of two PBC switching systems of 8×8 and 16×16. The experimental results are structured in three parts. First, we study the performance of a PBC under pure multicast traffic when mcastDROP and mcastWBA are deployed. Second, we compare the performance of DROP_mix to the existing solutions for both the unbuffered IQ and fully buffered CICQ crossbars. We used ESLIP algorithm for the IQ and the MURS_mix for the CICQ switch. The second part of the experiments is dedicated to studying the performance of DROP_mix under different internal settings. The performance metrics studied here are the average cell latency and throughput. Simulations run for 1 million time slots and statistics are gathered when fourth of the total simulation length has elapsed.
We studied the performance of our set of algorithms under the Bernoulli uniform and bursty uniform. Arriving cells can be either unicast or multicast. Cells arrive with a rate denoted by λ. Since the traffic is uniform, λ is the input load of the switch. The departure rate is denoted by μ. Similarly, μ is the output load of the switch. We consider admissible traffic, no input or output is oversubscribed. Because the traffic is a combination of unicast and multicast flows, the input load consists of a multicast fraction f_m and a unicast fraction f_u, where $\{(f_m, f_u)|f_m = 1 - f_u\}$. The fanout set, Φ, of multicast cells has cardinality (fanout number) $|\Phi|$ which is uniformly distributed between 1 and N (depending of the switch size, N can be 8 or 16) and all outputs have equal chances to be the destination of a multicast cell. Based on the above, the relationship between the switch input and output loads is expressed by:

$$\mu = \lambda(f_u + |\Phi|f_m)$$

Algorithm 3 Integrated Scheduling

Grant Phase, (DROP-PR) :
All grant pointers, g_j when scheduling VOQs, are initialized to different positions.

. For each output, j, do

. Set CQ_j equals number of non-full internal buffers of output j.

. Set the priority bit, P, to the logic OR of CQ_j entries.

. While there are credits in CQ_j do

- Starting from g_j index, send a grant to the first input, i, that requested this output (set $GQ_{ij} = 1$ and add bit P).
- Decrement CQ_j by one.

. Move the pointer g_j to location $(g_j + 1)(mod\ N)$.

Grant Phase, (MCAST) :
All the outputs share common grant pointers for multicast.

. While there are credits in CQ_j do

- mcastDROP — Starting from g_M index, send a grant to the input, i, that requested this output (set $GQ_{ij} = 1$).
- mcastWBA — Send a grant to the input, i, with the largest weight= $age - 2 \times fanout$ (set $GQ_{ij} = 1$).
- Decrement CQ_j by one.

. $diff = (max(selected\_input + N - g_M))(mod\ N)$.

. Move the pointer g_M to location $(g_M + 1)(mod\ N)$.

Input Scheduling Phase, (DROP-PR) :
All input pointers, a_i, are initialized to different positions.
For each input, i, do

. Starting from a_i index, select the first non empty VOQ_{ij} for which $GQ_{ij} = 1$ and bit $P = 1$ and send its HoL cell to the internal buffer.

. If no HoL cell is selected, Then

- Starting from a_i index, select the first non empty VOQ_{ij} for which $GQ_{ij} = 1$ and send its HoL cell to the internal buffer.

. Drop the remaining grants (reset GQ: $GQ_{i*} = 0$).

. Move the pointer a_i to location $(a_i + 1)(mod\ N)$.

Input Scheduling Phase, (MCAST) :
For each input, i, do

. Starting from a_i index, select the MQ_i for which $GQ_{i*} = 1$ and send its eligible cells belonging to the fanout of the mcast HoL cell to the internal buffers.

. Move the pointer a_i to location $(a_i + 1)(mod\ N)$.

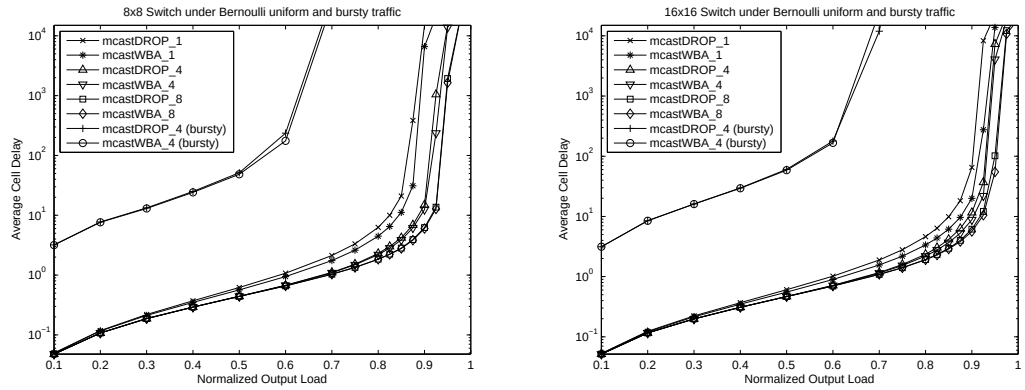
In our simulation, we averaged the cells fanout set[3]. Following our settings and substituting f_u with f_m, we have $\mu = \lambda(1 + 7f_m)$. For example, if we set f_m to be 0, the traffic is all unicast. When we set it to 1, the traffic becomes pure multicast. Whereas, if we

[3]Note that when the size of the switch is 8×8, the average fanout size becomes 4 and $\mu = \lambda(1 + 3f_m)$ to be $|\Phi| = 8$.

fix μ to 1 for example (switch fully loaded), we can vary f_m and see its effect on the throughput. When $f_m = 0.5$, the incoming traffic is evenly distributed between unicast and multicast flows.

4.5.1 mcastDROP vs. mcastWBA

For comparison, we show the performance of mcastDROP vs. mcastWBA when they are deployed in an IQ ($B = 1$) and a PBC switch. The fanout weight of the mcastWBA is set equal to twice the weight for HoL cell age. Figure 4.3 shows the performance of the two multicast schedulers with a different number of internal buffers for a 8×8 and 16×16 switch.



(a) An 8×8*switch*. (b) An 16×16*switch*.

Figure 4.3: Average cell delay of mcastDROP and mcastWBA with different switch sizes, varying B and pure multicast uniform and bursty input traffic ($f_m = 1$).

As expected, due to the fairness constraint, the mcastWBA exhibits better average cell delay when compared to the multicast version of the DROP-PR algorithm in all scenarios. This becomes obvious especially when the switch size is 8. But when the switch size gets increased, the difference in the performance gets smaller and smaller because increasing N also means that we increase the fanout cardinality of the multicast cells to be scheduled, therefore, the switch is now "flooded" for smaller values of the normalized output load. The same trend, but in a greater scale, is observed under bursty traffic[4]. This is because, it is harder for a scheduling policy to concentrate the residue on the minimum number of inputs when the arrivals of the multicast cells are correlated.
The two algorithms also differ slightly in the maximum possible throughput sustainable by the switch. In Table 4.1, it can be seen how the number of the internal buffers per output improves the matching efficiency of the switch when the normalized output load is maximum (meaning that the switch is "flooded"). Under the bursty arrival input traffic pattern, a 8×8 switch achieves 71.8% throughput, irrespectively of the scheduling discipline and the number of Bs (for a 16×16 the achieved throughput equals 67.6%).

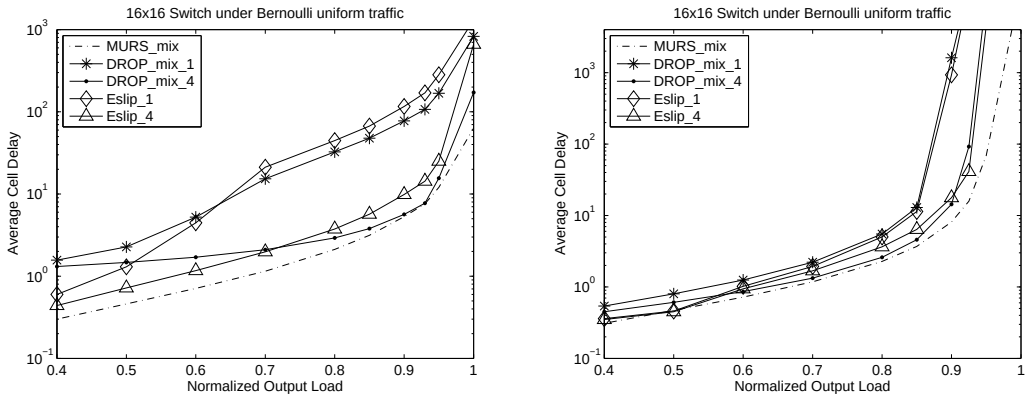
[4]The choice of an expected duration of 16 cells per burst is arbitrary, but is representative. The same qualitative results are obtained for different burst lengths.

	8×8 switch		16×16 switch	
B	mcastDROP	mcastWBA	mcastDROP	mcastWBA
1	0.88	0.892	0.91	0.926
2	0.902	0.908	0.921	0.932
4	0.926	0.929	0.937	0.942
8	0.95	0.951	0.953	0.956

Table 4.1: Throughput performance of a PBC switch under uniform traffic for different switch sizes and different Bs.

4.5.2 DROP_mix vs. MURS_mix and ESLIP

Figure 4.4(a) depicts the average delay performance of DROP_mix when $B = 1, 4$, ESLIP when the number of iterations equals 1 and 4 and MURS_mix for a fully buffered crossbar, under purely unicast traffic ($fm = 0$). As the Figure shows, MURS_mix has better delay performance than the other two. This shows the superior performance of the CICQ as compared to the IQ and PBC architecture. But, what is important to note is the tradeoff between the computation intensive nature of the ESLIP algorithm against the small internal buffering of the PBC switch. On the one hand, it is known that ESLIP converges to a MWM matching after $\log N$ iterations of the algorithm. Whereas, DROP_mix achieves the same average cell latency using $\log N$ buffers in just one iteration. Because DROP_mix works in a two-stage pipeline, it suffers some initial delay. It is essential to point out that under heavy load DROP_mix performs better than ESLIP. Decoupling the input scheduling from the grant scheduling done by the two-stage pipelined arbitration, and avoiding thus, the need for synchronized coordination between the grant and the input (accept) schedulers on a time slot basis as does ESLIP, has a profound impact on the average cell latency under unicast scheduling.



(a) (b)

Figure 4.4: (a) Average cell delay of DROP_mix, MURS_mix and ESLIP under Bernoulli uniform unicast traffic ($f_m = 0$). (b) Average cell delay of DROP_mix, MURS_mix and ESLIP under Bernoulli uniform unicast traffic ($f_m = 0.5$).

In order to better analyze the behavior of each scheduling algorithm, we tested the algorithms under the same settings as above using a multicast fraction of 0.5. Since we examine the case of using one multicast queue per input, pipelining the input scheduling phase does not provide much advantages, given the i.i.d. characteristic of the input arrival traffic pattern. That is the reason why the swap on the performance delay curve between the ESLIP and the DROP_mix is not observed. Although, both algorithms exhibit almost the same performance delay curve.
We wanted to compare the performance of DROP_mix to MURS_mix and ESLIP for different mixed traffic settings. However, checking all possibilities of mixed traffic requires tuning f_m from 0 to 1 and observing the throughput. To this end, we fixed the output load, μ, to be 100% (fully loaded system) and recorded the throughput of each algorithm as f_m varies from 0 to 1. Figure 4.5 compares the maximum achievable throughput of $DROP\_mix\_4$, MURS_mix and $ESLIP\_4$ under different switch sizes (8×8 and 16×16). We can see from Figure 4.5 that the total throughput achieved by all of the scheduling algorithms in question is always higher than 0.9. As it was expected, the throughput progressively decreases with respect to output load, but it is noticeable increased when multicast traffic dominates over unicast. This is not the case for the ESLIP scheduler, as the throughput gets stabilized for high values of the multicast fraction coefficient. DROP_mix is to be compared with the $FILM$ scheduling algorithm, introduced in [52], as the same trend is observed as far as the throughput is concerned.

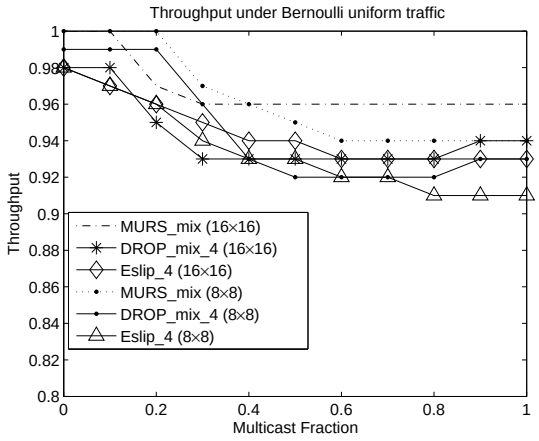


Figure 4.5: Throughput performance of $DROP\_mix\_4$, MURS_mix and $ESLIP\_4$ under different switch sizes and different multicast fractions.

4.5.3 DROP_mix with increasing internal buffers, B

We tested different PBC switch sizes, each with different numbers of internal buffers and we measured the average cell delay performance of the DROP_mix algorithm with different internal buffer settings under uniform and bursty traffic. In Figure 4.6, it can be easily seen that the total throughput is increased as B increases, due to the relaxation of the contention that the per output buffering offers. However, the switch size goes

against to the maximum throughput observed because a larger N implies a larger average fanout, and thus contention gets increased. Secondly, increasing B does not boost the performance by an order of magnitude as it is the case when scheduling unicast traffic, but it always lowers the average cell latency, as depicted in Figure 4.7(a) and Figure 4.7(b). The same behavior is observed under bursty uniform arrivals.

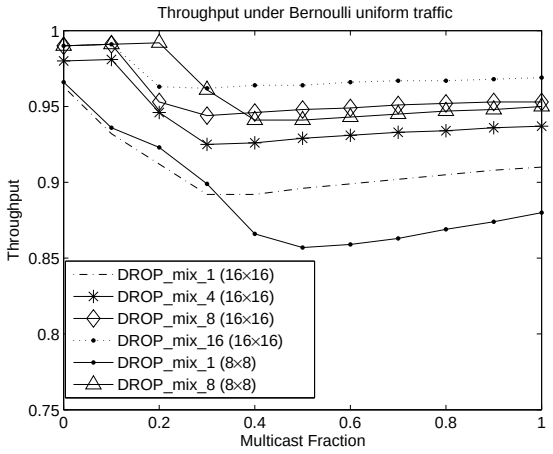
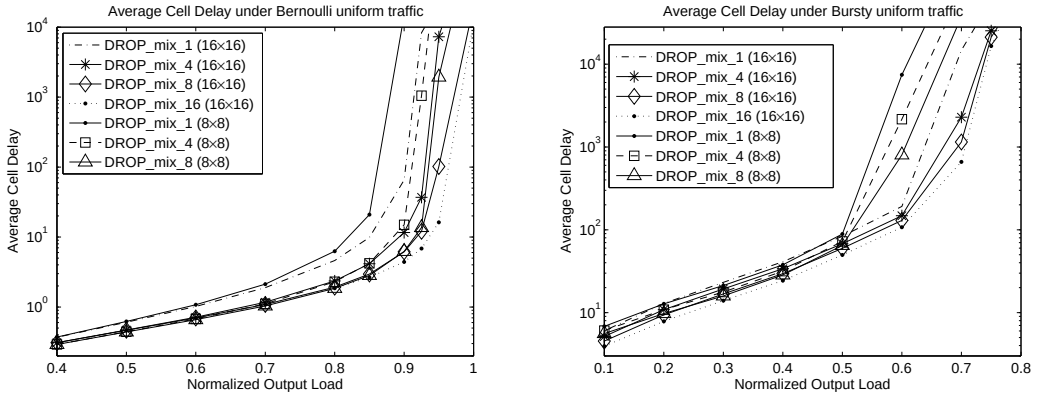


Figure 4.6: Throughput performance of DROP_mix with different switch sizes, different multicast fractions and varying B.



(a) (b)

Figure 4.7: (a) Average cell delay of DROP_mix with different switch sizes, varying B and mixed uniform input traffic ($f_m = 0.5$). (b) Average cell delay of DROP_mix with different switch sizes, varying B and mixed bursty input traffic ($f_m = 0.5$).

In the PBC switching architecture, an output can send up to B grants and an input can receive up to N grants, one from each output. When unicast scheduling, the grants an output can send are shared among all N inputs, as an input can receive only one grant from an output. When multicast scheduling, the grants an output can send are shared

among all the inputs N multiplied by the sum of the fanouts of HoL multicast cells. As a result, the number of cells contending for the same output is significantly increased when considering multicast traffic, whereas the number of cells that are actually transferred to the internal buffers of that output stays fixed. This is why increasing B, lowers the average cell latency but not as much as when when unicast traffic is scheduled.

4.6 Conclusion

The PBC switching architecture has been recently proposed and shown to take the best of both the unbuffered IQ and fully buffered CICQ switching architectures. It maintains a small number of internal buffers per fabric output, making its cost comparable to that of unbuffered crossbars. However, these shared internal buffers bring about significant advantages; they allow distributed and pipelined scheduling algorithm, as opposed to the centralized scheduling in unbuffered crossbars. the PBC has much lower hardware cost compared to a fully buffered crossbar. This chapter conducts the first study of integrating unicast and multicast traffic scheduling in PBC switches. We proposed a couple of scheduling algorithms, that are well tailored for the PBC switch and tested their performance under a wide range of traffic patterns and switch size settings. We studied different tradeoffs in terms of internal buffering and input traffic distribution and observed the performance of the PBC. Simulation results show that the PBC exhibits good performance and presents a practical choice over both unbuffered and fully buffered crossbars.

5 Conclusions

High-speed routers are complex devices. They function at the heart of the tremendous growth and complexity of the Internet. We have discussed the design of scheduling algorithms for high speed switching fabrics. This thesis has studied the PBC switching architecture, exploring ways towards scalable, low cost, and robust switching system. So far, the scheduling algorithms known in literature have been either simple to implement but with poor performance or too complex but with good performance.

The PBC switch exhibits better performance but still being cost-comparable to unbuffered switches, as the performance results have shown. In this work, we eliminated the cost-performance tradeoff regarding the fully buffered crossbars by devising schedulers directly coupled to the underlying switch architecture. The latter resulted to a switching architecture that is strictly better than CICQ switches. We have addressed two of the most popular issues regarding the scheduling process in high-speed switches, namely, performance guarantees and MWM approximation. And, we have motivated the problem of simultaneously scheduling unicast and multicast traffic.

Our first significant contribution, mainly of theoretical interest, has been the mapping of the bufferless optimal MWM scheduling in a PBC switch. We used the fluid model theory, together with Lyapunov functions, in order to prove and come up with a scheduler and demonstrate how the switching architecture in question can achieve 100% throughput without any speedup.

Our second significant contribution has been the proposal of an adaptive frame scheduling algorithm that makes use of the circuit switching theory (AF-DROP-PR). The main idea that we exploit is to regressively allocate throughput to a flow-aware switch, in this way, we achieve the performance and fairness index of the ideal OQ switch, minimizing the memory requirements. To our knowledge, there has not been proposed any switching architecture that provides throughput guarantees in a more practical way. The pigeonhole principle was used to identify the conditions that would enable us to tell something predictable when scheduling in PBC switches.

Our third contribution has been the proposal of an implementable algorithm that schedules unicast and multicast traffic in parallel. Thanks to that, we were the first ones that studied the performance of a PBC switch under integrated traffic. Also, we were able to find out about an interesting tradeoff between the number of iterations a state-of-the-art integrated scheduler needs to perform in order to achieve an efficient matching between inputs and outputs and the number of internal buffers in a PBC needed to achieve the same or better performance with just one iteration.

We wish to make the last point. As system requirements increase, the capabilities of hardware may continue to lag. The underlying hardware can be slow, inefficient, unreliable, and perhaps even variable and probabilistic in its performance. Of course, some of the above are already true with regard to memory. And so, it will become neces-

sary to find architectural and algorithmic techniques to solve these problems. If we can discover techniques that are sufficient to emulate the large performance requirements of the system, then it is possible to build solutions that can use such imperfect underlying hardware.

Definitions and Traffic Models

A

A.1 Definitions

We assume that time is slotted into cell times. Let $A_{ij}(n)$ denote the cumulative number of arrivals to input i of cells destined to output j at time n. Let $A_i(n)$ denote the cumulative number of arrivals to input i. During each cell time, at most one cell can arrive at each input. λ_{ij} is the arrival rate of $A_{ij}(n)$. $D_{ij}(n)$ 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. $X_{ij}(n)$ 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:

$$X_{ij}(n+1) = X_{ij}(n) + A_{ij}(n) - D_{ij}(n) \tag{A.1}$$

Let $A(n)$ denote the vector of all arrivals $\{A_{ij}(n)\}$, $D(n)$ denote the vector of all departures $\{D_{ij}(n)\}$, 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:

$$X(n+1) = X(n) + A(n) - D(n) \tag{A.2}$$

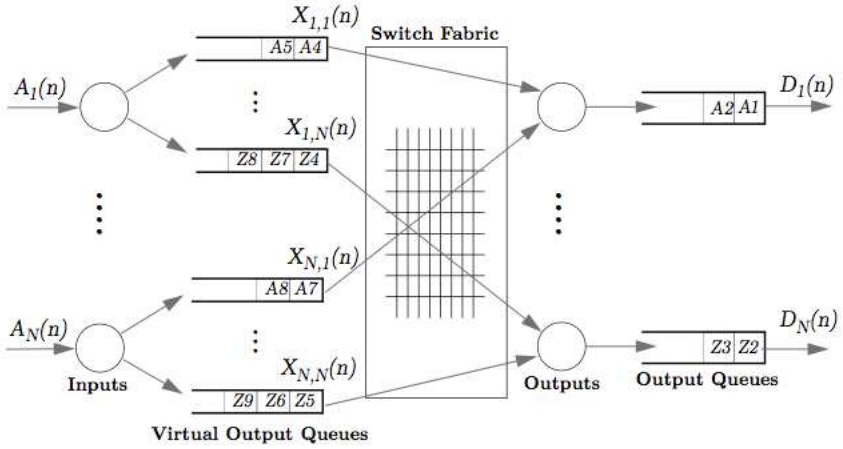


Figure A.1: Network traffic models [2].

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

- **i.i.d.:** Traffic is called independent and identically distributed (i.i.d.) 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.

- **100% Throughput:** A router is said to achieve 100% throughput if under any admissible iid traffic, for every $\epsilon > 0$, there exists $C > 0$ such that $\lim_{n\to\infty} Pr\{\sum_{ij} X_{ij}(n) > C\} < \epsilon$.

Performance Analysis of a PBC Switch

B

In this appendix, we'll describe an algorithm that implements a technique for detecting the performance bottleneck among the different router line cards. More specifically, giving the current internal buffer size configuration, the algorithm tries to identify the outputs where adding extra buffering space leads to the maximum improvement in performance. One way to guide this process would be to simulate the system implementing such a configuration and then profile the simulation results. Unfortunately, despite its flexibility, the simulation approach suffers from extremely long simulation times. Since we need to evaluate the system for every possible buffer configuration, the evaluation based on direct simulation is simply impossible to afford. In the following, we propose a analytical model which can be used to quickly analyze the current internal buffer size configuration and detect the performance bottlenecks; this is done by solving a series of nonlinear equations derived from queuing models.
We will try to identify the output port where adding extra buffering space leads to the maximum improvement in performance. We will do that by detecting which output has the highest blocking probability, the output port which owns the internal buffers that are responsible for the performance bottleneck in the current switch configuration should increase the size of its internal buffers.
To solve this problem analytically, we resort to the theory of finite queueing networks. The basic element in the model is a $M/D/1/K$ finite queue, consisting of one $M/M/1$ queue and one $M/D/1/K$ queue, as it will be explained later on.
The arrival rate λ_{ij} destined for $output_j$ caused by $input_i$ can be calculated using the following equation:

$$\lambda_{ij} = (cell\_injection\_rate_i) \times (probability\_of\_receiving\_grant_i) \times on_j$$

where on_j equals 1 if the cell from $input_i$ is destined to $output_j$, otherwise it is equal to 0.

The calculation of the service rate μ_j of the cells destined to $output_j$ is not trivial, as it depends not only on the switch's service delay, but also on probabilities of a cell being switched to $output_j$ and whether or not the internal buffers of the $output_j$ are full.

Once the value of the service rate per output port is determined, the probability $output_j$'s internal buffers to be in "full state" can be calculated easily using the finite $M/D/1/K$ queueing model:

$$c_j = 1 - \frac{b_{K-1}}{1 + \rho_j \times b_{K-1}} \tag{B.1}$$

where $\rho_j = \lambda_j/\mu_j$ and $b_n = \sum_{a=0}^{n} \frac{(-1)^a}{a!} \times (n-a)^a \times e^{(n-a)\times\rho} \times \rho^a$.
Cells destined to $output_j$ accept grant provided that $output_j$ has available space in its internal buffers. Thus, we can use the reciprocal of the blocking probability to approximate the service rate that can be provided upstream. More specifically, the effective service rate as observed by λ_j is approximated as $1/c_j$. Since the total arrival rate to $output_j$ is $\lambda_j = \sum_i \lambda_{ij}$, if we approximate the service provided downstream to $output_j$ as an $M/M/1$ queue, the expected number of cells to be delivered to the internal buffers of $output_j$ can then be calculated as:

$$E[N_{M/M/1;j}] = N_j = \frac{\lambda_j}{\frac{1}{c_j} - \lambda_j}$$

On the other hand, based on Little's formula:

$$N_j = \lambda_j \times w_j$$

So, the average time spent for entering $output_j$'s internal buffers can then be approximated as:

$$E[w_{M/M/1;j}] = w_j = \frac{1}{\frac{1}{c_j} - \lambda_j} \tag{B.2}$$

`On the other hand,` w_j `should also be the average waiting time observed by a cell in the` λ_j `flow if it is to be delivered to` $output_j$

The arrival rate of a flow λ_{ij} coming from $input_i$ and destined to $output_j$ is equal to $p_{ij} \times \lambda_{ij}$. If this flow should provide the same average cell latency, then its service rate $\bar{\mu}_j^i$ must satisfy the following equation, again based on Little's formula:

$$w_j = \frac{1}{\bar{\mu}_j^i - p_{ij} \times \lambda_{kj}} \tag{B.3}$$

Substituting w_j in Equation B.2 the RHS of Equation B.3, we can calculate the equivalent service rate of this flow by:

$$\bar{\mu}_j^i = \frac{1}{c_j} - \lambda_j + p_{ij} \times \lambda_{ij} \tag{B.4}$$

The average upstream service experienced by all input ports i from output port j can now be calculated by the following equation:

$$\bar{\mu}_j = \sum_i p_{ij} \times \bar{\mu}_j^i \tag{B.5}$$

The next step to further reduce the model is merging the two queues. Thus, the average queue length of the λ_j downstream flow can be approximated by a $M/D/1/K$ queue (due to the work–conservation principle), so:

$$E[Q_{M/D/1/K;j}] = K - \frac{\sum_{n=0}^{K-1} b_n}{1 + \rho_j \times b_{K-1}}$$

Then, the average queue length of λ_j should be equal to the sum of the queue that models the delay an input request has to wait to receive a grant (note; when $\rho \to 1$ this delay tends to be equal to N/B) and the queue that models the availability of the $output_j$'s internal buffers (upstream queue model):

$$E[Q] = \frac{\lambda_j}{\frac{1}{(\frac{N}{2\times B})} - \lambda_j} + \frac{\lambda_j}{\bar{\mu_j} - \lambda_j}$$

Combining the last two equations together, we have:

$$\mu_j = \frac{\lambda_j \times b_{K-1} \times (K - E[Q])}{\sum_{n=0}^{K-1} b_n - K + E[Q]} \tag{B.6}$$

At this point, we have described the relation between the $output_j$ service rate μ_j and the $output_j$ blocking probability c_j (by combining Equation (B.4)(B.5)(B.6)).

By performing similar derivations for all the outputs of the switch, we can finally build a series of equations which describe the overall switch's behavior. The initial buffer configuration is decided based on profiling the grant probability per input. The $fsolve$ utility from MATLAB or OCTAVE can be used as the non-linear solver of the system equations. The output port which has the highest blocking probability is selected as the bottleneck output port and the size of each internal buffers is increased. The above procedure is repeated until the desired performance is achieved.

Remarks; (1) We do assumptions about the aggregate arrival stream, more specifically, we assume that it consists of a large number of sources with none of them dominating the others. (2) The AMS queue allows us to analyze the effect of the burstiness of a source by varying λ.

Bibliography

[1] L. Mhamdi. PBC: A partially buffered crossbar packet switch. *Computers, IEEE Transactions on*, 58(11):1568 –1581, nov. 2009.

[2] Sundar Iyer. *Load Balancing and Parallelism for the Internet*. PhD thesis, Stanford University, 2008.

[3] Bin Liu H. Jonathan Chao. *High Performance Switches and Routers*. Wiley-IEEE Press, 2007.

[4] P. Gupta. *Algorithms for routing lookups and packet classification*. PhD thesis, Stanford University, 2000.

[5] Piet Van Mieghem. *Data Communications Networking*. Techne Press, 2006.

[6] Denzel W.E. Engbersen T. Herkersdorf A. Bux, W. and Ronald P. Luijten. Technologies and building blocks for fast packet forwarding. *IEEE Communications Magazine*, pages 70–77, 2001.

[7] http://www.qdrsram.com/.

[8] http://www.micron.com/products/dram/.

[9] Robert R. Schaller. Moore's law: past, present, and future. *IEEE Spectr.*, 34(6):52–59, 1997.

[10] Shang-Tse Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input/output-queued switch. *Selected Areas in Communications, IEEE Journal on*, 17(6):1030 –1039, jun 1999.

[11] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[12] http://www1.cs.columbia.edu/ evs/intro/stable/writeup.html.

[13] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. In *Decision and Control, 1990., Proceedings of the 29th IEEE Conference on*, pages 2130 –2132 vol.4, dec 1990.

[14] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *Communications, IEEE Transactions on*, 47(8):1260 –1267, aug 1999.

[15] J.G. Dai and B. Prabhakar. The throughput of data switches with and without speedup. volume 2, pages 556 –564 vol.2, 2000.

[16] M. Nabeshima. Performance evaluation of a combined input-and crosspoint-queued switch. *IEICE Transactions on Communications*, E83-B(3):737–741, March 2000.

[17] Francois Abel, Cyriel Minkenberg, Ronald P. Luijten, Mitchell Gusat, and Ilias Iliadis. A four-terabit packet switch supporting long round-trip times. *IEEE Micro*, 23(1):10–24, 2003.

[18] Vinodh Cuppu, Bruce Jacob, Brian Davis, and Trevor Mudge. A performance comparison of contemporary dram architectures. In *ISCA '99: Proceedings of the 26th annual international symposium on Computer architecture*, pages 222–233, Washington, DC, USA, 1999. IEEE Computer Society.

[19] Cyriel Minkenberg, Ronald P. Luijten, François Abel, Wolfgang Denzel, and Mitchell Gusat. Current issues in packet switch design. *SIGCOMM Comput. Commun. Rev.*, 33(1):119–124, 2003.

[20] S.-T. Chuang, S. Iyer, and N. McKeown. Practical algorithms for performance guarantees in buffered crossbars. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 981 – 991 vol. 2, march 2005.

[21] R.E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial Mathematics, 1987.

[22] B. Prabhakar, N. McKeown, and R. Ahuja. Multicast scheduling for input-queued switches. *Selected Areas in Communications, IEEE Journal on*, 15(5):855 –866, jun 1997.

[23] Cheng-Shang Chang, Wen-Jyh Chen, and Hsiang-Yi Huang. On service guarantees for input-buffered crossbar switches: a capacity decomposition approach by birkhoff and von neumann. In *Quality of Service, 1999. IWQoS '99. 1999 Seventh International Workshop on*, pages 79 –86, 1999.

[24] Y. Tamir and H.-C. Chi. Symmetric crossbar arbiters for vlsi communication switches. *Parallel and Distributed Systems, IEEE Transactions on*, 4(1):13 –27, jan 1993.

[25] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. High speed switch scheduling for local area networks. In *ASPLOS-V: Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, pages 98–110, New York, NY, USA, 1992. ACM.

[26] A. Mekkittikul and N. McKeown. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 792 –799 vol.2, mar-2 apr 1998.

[27] P. Giaccone, D. Shah, and B. Prabhakar. An implementable parallel scheduler for input-queued switches. *Micro, IEEE*, 22(1):19 –25, jan/feb 2002.

[28] N. McKeown. The islip scheduling algorithm for input-queued switches. *Networking, IEEE/ACM Transactions on*, 7(2):188 –201, apr 1999.

[29] R. Rojas-Cessa, E. Oki, Zhigang Jing, and H.J. Chao. Cixb-1: combined input-one-cell-crosspoint buffered switch. In *High Performance Switching and Routing, 2001 IEEE Workshop on*, pages 324 –329, 2001.

[30] T. Javidi, R. Magill, and T. Hrabik. A high-throughput scheduling algorithm for a buffered crossbar switch fabric. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 5, pages 1586 –1591 vol.5, 2001.

[31] D. Shah and M. Kopikare. Delay bounds for approximate maximum weight matching algorithms for input queued switches. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 1024 – 1031 vol.2, 2002.

[32] H.J. Kushner. *Stochastic Stability and Control*. Academic Press, Orlando, FL, 1967.

[33] Guy Fayolle. On random walks arising in queueing systems: ergodicity and transience via quadratic forms as lyapunov functions—part i. In *Proceedings of the workshop held at the Mathematical Sciences Institute Cornell University on Mathematical theory of queueing systems*, pages 167–184, Red Bank, NJ, USA, 1989. J. C. Baltzer AG, Science Publishers.

[34] Paolo Giaconne. *Queueing and scheduling algorithms for high performance routers*. PhD thesis, Politechnico Di Torino, 2002.

[35] http://en.wikipedia.org/wiki/EDRAM.

[36] Shang-Tse Chuang. *Providing Performance Guarantees in Crossbar-based routers*. PhD thesis, Stanford University, 2005.

[37] R. Rojas-Cessa. High-performance round-robin arbitration schemes for input-crosspoint buffered switches. In *High Performance Switching and Routing, 2004. HPSR. 2004 Workshop on*, pages 167 – 171, 2004.

[38] M. Katevenis N. Chrysos. Transient behavior of a buffered crossbar converging to weighted max-min fairness. Technical report, Inst. of Computer Science, FORTH, 2002.

[39] Sundar Iyer, Rui Zhang, and Nick McKeown. Routers with a single stage of buffering. *SIGCOMM Comput. Commun. Rev.*, 32(4):251–264, 2002.

[40] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report TR-301, DEC Research, September 1984.

[41] Lotfi Mhamdi, Christopher Kachris, and Stamatis Vassiliadis. A reconfigurable hardware based embedded scheduler for buffered crossbar switches. In *FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 143–149, New York, NY, USA, 2006. ACM.

[42] James Balfour and William J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 187–198, New York, NY, USA, 2006. ACM.

[43] I.V. Senin, L. Mhamdi, and K. Goossens. Efficient multicast support in buffered crossbars using networks on chip. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1 –7, 30 2009-dec. 4 2009.

[44] M. Andrews, S. Khanna, and K. Kumaran. Integrated scheduling of unicast and multicast traffic in an input-queued switch. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1144 –1151 vol.3, mar 1999.

[45] N. McKeown. A fast switched backplane for a gigabit switched router. *Business Commun. Rev.*, 27(12), 1997.

[46] L. Mhamdi. On the integration of unicast and multicast cell scheduling in buffered crossbar switches. *Parallel and Distributed Systems, IEEE Transactions on*, 20(6):818 –830, june 2009.

[47] N. McKeown. *Scheduling Algorithms for Input-Queued Cell Switches*. PhD thesis, University of California at Berkeley, 1995.

[48] L. Mhamdi and M. Hamdi. Scheduling multicast traffic in internally buffered crossbar switches. In *Communications, 2004 IEEE International Conference on*, volume 2, pages 1103 – 1107 Vol.2, june 2004.

[49] J.Y. Hui and T. Renner. Queueing analysis for multicast packet switching. *Communications, IEEE Transactions on*, 42(234):723 –731, feb/mar/apr 1994.

[50] M.A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri. Multicast traffic in input-queued switches: optimal scheduling and maximum throughput. *Networking, IEEE/ACM Transactions on*, 11(3):465 – 477, june 2003.

[51] B. Prabhakar, N. McKeown, and R. Ahuja. Multicast scheduling for input-queued switches. *Selected Areas in Communications, IEEE Journal on*, 15(5):855 –866, jun 1997.

[52] E. Schiattarella and C. Minkenberg. Fair integrated scheduling of unicast and multicast traffic in an input-queued switch. In *Communications, 2006. ICC '06. IEEE International Conference on*, volume 1, pages 287 –292, june 2006.

Curriculum Vitae

Nikolaos Skalis is $1.88m$ tall, outrageously handsome, Greek, and proud of it. In addition to being a hero, trendsetter, and leader of fashion, he is widely regarded as an expert in all aspects of electronics (at least by his mother).

After receiving his Engineering Diploma in electrical engineering in 2007 from Democritus Polytechnic, Greece, Nikos continued his studies in the Delft University of Technology in Embedded Systems. To cut a long story short, Nikos now finds himself be undetermined about what he is going to do next.

In his spare time (Ha!), Nikos is attending in as many music and art events as he can. On the off-chance that you 're still not impressed, Nikos was once referred to as a "networking expert" by someone famous, who wasn't prompted, coerced, or remunerated in any way!