12-2007

Fabric-on-a-Chip: Toward Consolidating Packet Switching Functions on Silicon

William B. Matthews
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by William B. Matthews entitled "Fabric-on-a-Chip: Toward Consolidating Packet Switching Functions on Silicon." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Itamar Elhanany, Major Professor

We have read this dissertation and recommend its acceptance:

Gregory Peterson, Donald W. Bouldin, Bradley Vander Zanden

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:
I am submitting herewith a dissertation written by William B. Matthews entitled "Fabric-on-a-Chip: Toward Consolidating Packet Switching Functions on Silicon". I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Itamar Elhanany
Major Professor

We have read this dissertation
and recommend its acceptance:

Gregory Peterson

Donald W. Bouldin

Bradley Vander Zanden

Accepted for the Council:

Carolyn R. Hodges
Vice Provost and
Dean of the Graduate School

Fabric-on-a-Chip: Toward Consolidating Packet Switching Functions on Silicon

A Dissertation

Presented for the Doctor of Philosophy Degree

The University of Tennessee, Knoxville

William Brad Matthews

December 2007

Dedication

This dissertation is dedicated to my wife for her love and support. Thank you.

Acknowledgments

I would like to thank my advisor, Dr. Itamar Elhanany, for the guidance and tremendous support both academically and professionally. I am especially grateful for his preserverance and insightful instruction throughout my time in the program. Thank you.

I would further like to thank Dr. Donald Bouldin, Dr. Gregory Peterson, and Dr. Bradley Vander Zanden, for serving on my Ph.D. committee. The time and input provided is greatly appreciated.

This work received considerable benefit from the input of Vahid Tabatabaee, who provided insight and assisted in verifing some of the proofs presented in this dissertation.

The majority of my time in the Ph.D. program was supported by the Bodenheimer Fellowship. I would like to thank Michael Crabtree and the remaining Bodenheimer Fellowship Endowment donors for their tremendous generosity. For this, I am especially grateful and appreciative.

Lastly, I would like thank my wife and parents for all your love and support. It has meant everything to me.

Abstract

The switching capacity of an Internet router is often dictated by the memory bandwidth required to buffer arriving packets. With the demand for greater capacity and improved service provisioning, inherent memory bandwidth limitations are encountered rendering input queued (IQ) switches and combined input and output queued (CIOQ) architectures more practical. Output-queued (OQ) switches, on the other hand, offer several highly desirable performance characteristics, including minimal average packet delay, controllable Quality of Service (QoS) provisioning and work-conservation under any admissible traffic conditions. However, the memory bandwidth requirements of such systems is $O(NR)$, where N denotes the number of ports and R the data rate of each port. Clearly, for high port densities and data rates, this constraint dramatically limits the scalability of the switch.

In an effort to retain the desirable attributes of output-queued switches, while significantly reducing the memory bandwidth requirements, distributed shared memory architectures, such as the parallel shared memory (PSM) switch/router, have recently received much attention. The principle advantage of the PSM architecture is derived from the use of slow-running memory units operating in parallel to distribute the memory bandwidth requirement. At the core of the PSM architecture is a memory management algorithm that determines, for each arriving packet, the memory unit in which it will be placed. However, to date, the computational complexity of this algorithm is $O(N)$, thereby limiting the scalability of PSM switches.

In an effort to overcome the scalability limitations, it is the goal of this dissertation to extend existing shared-memory architecture results while introducing the notion of Fabric on a Chip (FoC). In taking advantage of recent advancements in integrated circuit technologies, FoC aims to facilitate the consolidation of as many packet switching functions as possible on a single chip. Accordingly, this dissertation introduces a novel pipelined memory management algorithm, which plays a key role in the context of on-chip output-queued switch emulation. We discuss in detail the fundamental properties of the proposed scheme, along with hardware-based implementation results that illustrate its scalability and performance attributes.

To complement the main effort and further support the notion of FoC, we provide

performance analysis of output queued cell switches with heterogeneous traffic. The result is a flexible tool for obtaining bounds on the memory requirements in output queued switches under a wide range of traffic scenarios. Additionally, we present a reconfigurable high-speed hardware architecture for real-time generation of packets for the various traffic scenarios. The work presented in this thesis aims at providing pragmatic foundations for designing next-generation, high-performance Internet switches and routers.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent years have witnessed unprecedented advances in the design, verification formalism [1], and deployment of high-capacity, high-performance packet-switching fabrics. Such fabrics are commonly employed as the fundamental building blocks in data-networking platforms that span a wide variety of application spaces. The market segment for which a switch or router was designed predominantly dictates its capacity. Core (or backbone) Internet routers, for example, are able to support multiple terabits per second [2], while systems built for metropolitan area networks (MANs) typically carry hundreds of gigabits per second (Gbps). Local area networks, representing the lower end of the switch/router market, have a switching fabric that supports up to tens of Gbps. However, switching fabrics are not limited to Internet transport equipment. Storage area networks (SANs), for example, often necessitate large packet switching engines to allow vast amounts of data to traverse a fabric, whereby storage data segments (*i.e.* blocks) flow from storage devices to servers and users, and vice versa.

During the late 1990s, many believed that the growth in Internet traffic would increase at a rate that would require significant upgrades in switching infrastructure as often as every 18 months. However, despite the increasing growth in user traffic (approximated to double every 12 months)[3], the pragmatic requirements of backbone switches and routers are somewhat more modest. Nevertheless, the large number of components in switch fabrics, which drive such large systems, renders the latter highly complex to design, test and maintain. Thus, in an effort to alleviate some of the key difficulties in designing large

switching fabrics, the concept of Fabric on a Chip (FoC) is introduced. In view of realistic technological limitations, it should be noted that FoC solutions would not be designed for core/backbone routers. Rather, the target application space for such products would be where hundreds of Gbps and below are required, *e.g.* in MAN, high-end LAN, and SAN, among others.

Taking advantage of recent advances in integrated circuit technologies, the goal of FoC architectures is to enable the consolidation of as many core switching functions as possible on a single chip. By achieving a high level of integration, it is argued that much larger systems can be readily realized. Moreover, the resulting designs will consume significantly fewer resources than the traditional approach. Accordingly, this chapter focuses on the topic of on-chip output-queued router emulation. The memory management problem is introduced, to which solutions using a novel architecture and algorithms are offered and discussed in detail.

1.1 Benefits of the Fabric-on-a-Chip Approach

There are numerous benefits to considering the notion of consolidating switching fabric functions on a chip. The first is the ability to reduce system physical components. By reducing the number of chips in the system, we directly obtain a reduction in size and design complexity, resulting in simplified board layouts and mechanical considerations. Such improvements in the design process are far from negligible, because a corollary to simplified design is shorter design cycles. As discussed earlier, reduction of power requirements – a crucial pragmatic aspect of any switch/router – is obtained due, primarily, to the reduction in high-speed serial transceivers in use. The proposed approach replaces chip-to-chip communication with on-chip communication, which has considerably lower power-consumption characteristics.

When considering the design of systems with capacities of hundreds of Gbps and beyond, other engineering aspects play a key role in guaranteeing successful deployment. One such key issue is staying within a workable power budget; high-speed serial transceivers that enable the transmission and reception of data signals at rates of Gbps, require a great

deal of power. Practical maintenance constraints limit the amount of power that switches and routers may consume. In conventional switch/router designs, multiple high-bandwidth data signals originating from the input ports, arrive at the fabric and traverse it en route to their destination ports. Any reduction in the number of serializer/de-serializer (SerDes) circuits utilized by the various chips is guaranteed to directly reduce the overall system power consumption. Yet another element impacting power consumption is the amount of memory devices used.

A related advantage of FoC is higher reliability. It is generally acknowledged that lowering the number of (independent) components in any given system increases its reliability, since fewer components are prone to failures and thus need to be replaced. In view of recent technical standardization efforts pertaining to packet processing products, one may argue that FoC helps facilitate the rapid exploitation of standard interfaces to further support the interoperability between different semiconductor products used in a switch or router. Lastly, the notion of FoC is coherent with the recent trend toward modern System on a Chip (SoC), a trend that is gaining momentum due to the inherent advantages it presents, in particular with respect to cost reduction.

Figure 1-1 illustrates the various components of a traditional input-queued switch; these components have the potential to be integrated on a single chip as part of the FoC framework. Improvements in the fabrication of VLSI circuitry play a key role as enablers for FoC. Due to advances in packaging technology, it becomes plausible to consider that all data packets arrive at the FoC directly. This reduces the need for virtual output queueing [4] and some output buffers associated with standard router architectures. Due to the ability to embed multiple megabits of dual-port SRAM on a chip, packets can be efficiently stored and switched internally. We shall refer to packets as being of fixed size. This is generally true for all practical switch fabric designs, as external packets are typically segmented into fixed-size data units and reassembled as they exit the switch.

The crosspoint switches and scheduler, key components in input-queued switches, are avoided thereby substantially reducing chip count and power consumption. Correspondingly, much of the signaling and control information that typically spans multiple chips can be carried out on chip. Finally, the router management and monitoring functions can be

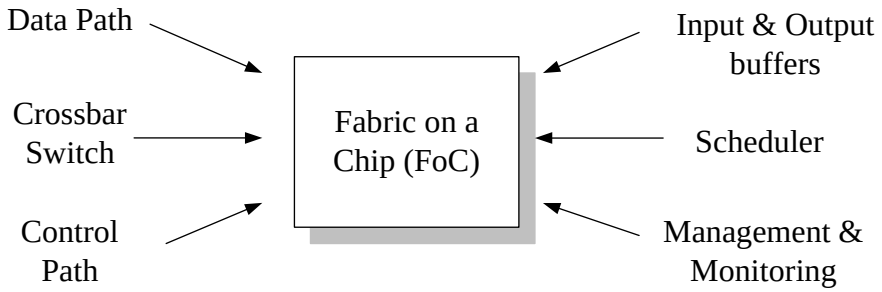

Figure 1-1: Potential functions to be consolidated as part of the FoC framework

centralized since all the information is available at a single location.

A packet switch can be viewed as a discrete time system connecting m inputs to n outputs. Its core function is the forwarding of packets arriving at input link i, $1 \leq i \leq m$, to the output link j, $1 \leq j \leq n$, provided by an identifier located in the header of each packet. It is commonly assumed that all links operate at the same transmission rate and that packets are of the same fixed size. The arrival process, $A_{i,j}(t)$ is a discrete time process where packet arrivals occur occur at port i for output port j in slotted time intervals corresponding to the transmission time of a single fixed size packet, or time slot**.** If we consider the output queued switch, depicted in Figure 1-2, it is clear that multiple packets, arriving during the same packet time, will likely be simultaneously destined for the same output link. The departure process, $D_j(t)$, is also a discrete time process such that either zero or one packet can depart during a given time slot. Clearly, the simultaneous arrival of more than one packet will result in a *conflict* at the output, commonly referred to as *output contention.* In a pure output queued switch, all packets arriving in the same time slot, destined for the same output port, j, are eligible to be transferred across the switch fabric into the same FIFO output queue, Q_j, within one time slot.

All of the packet switching architectures detailed in this dissertation attempt to switch packets from N input (ingress) to N output (egress) links, hence $m = n$. For the output queued switch fabric, it is evident that transferring N packets, with a line rate of R, to a single FIFO output queue results in an memory bandwidth requirement of O(NR). This is derived from the need to be able to accept (write) up to N arriving packets while simulta-

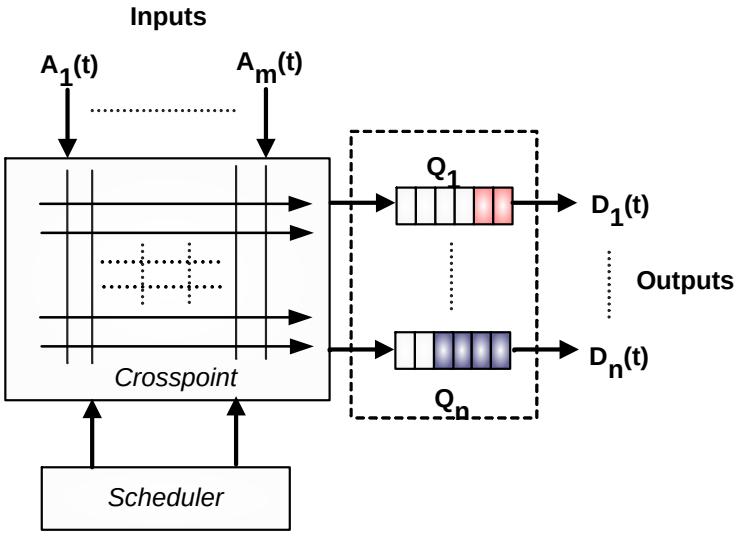

Figure 1-2: Components of an output queued (OQ) switch.

neously transmitting (reading) up to N departing packets. This requirement significantly limits the scalability of OQ switches with respect to their aggregate switching capacity.

Alternately, buffering of packets can be designated to occur at the input ports, thereby reducing the memory bandwidth requirement to match the line rate R. In the instance that FIFO queueing at each input is employed, only the first packet in each queue is eligible for transfer across the switch fabric. If the first cell at the front of the FIFO is blocked, other cells in the queue cannot be forwarded to an unused output. This phenomenon is commonly referred to as *head-of-line* (HOL) blocking and results in degraded switching throughput. A switch architecture is called *non-blocking* if it satisfies the requirement that an arriving packet destined for any output, to which no other packets are destined, can be forwarded immediately regardless of the destinations assigned to all other arriving packets.

In comparing input queued and output queued switch fabrics, it is clear that output queued switch fabrics require increased bandwidth for the interconnect and memory. However, they do not suffer from HOL blocking as arriving packets destined to one output link do not block packets destined to different outputs. A key advantage of output queued architectures is that the average delay, as experienced by arriving packets, is kept to a minimum. Moreover, output queued switches facilitate controllable Quality of Service (QoS)

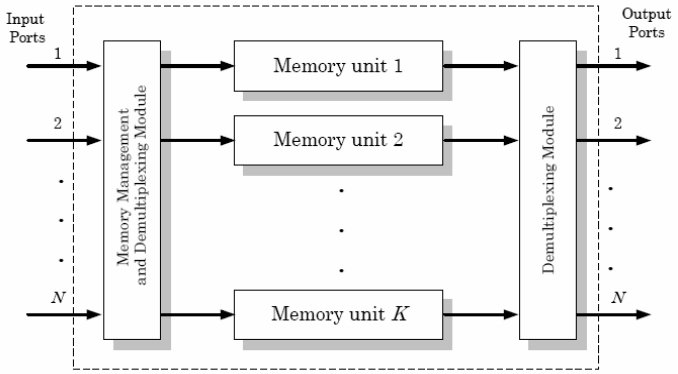

Figure 1-3: General structure governing the proposed parallel shared memory (PSM) switch architecture. Incoming packets are placed in a set of K $(>N)$ memory units

provisioning. There have been attempts to emulate the performance of an output queued switch by means of utilizing input queued architectures with an internal speedup of 2. In subsection 1.3.1, this approach and its inherent limitations are presented in detail.

1.2 The Parallel Shared Memory (PSM) Switch

In contrast to existing shared memory switch architectures, an alternative approach, termed the parallel shared memory (PSM) switch, has recently received much attention. The PSM switch offers the ability to retain the desirable attributes of output queued switches while significantly reducing their memory bandwidth requirements [5]. A PSM switch fabric, depicted in Figure 1-3, utilizes a pool of slow-running memory units operating in parallel. In order to properly operate, the PSM switch must have sufficient memory bandwidth. At the core of the architecture is a memory management algorithm that determines, for each arriving packet, the memory unit in which it will be placed. However, the complexity of such algorithms found to date is $O(N)$, where N denotes the number of switch ports, thereby inherently limiting the scalability of the scheme.

Initial work has indicated that, assuming each of the shared memory units can perform at most one packet-read or -write operation during each time slot, a sufficient number of memories needed to emulate a FCFS output queued switch is $K = 3N-1$ [5]. The latter can

be proven by employing constraint sets analysis (also known as the "pigeon hole" principle), summarized as follows. An arriving packet must always be placed in a memory unit that is currently not being read from by any output port. Since there are N output ports, this first condition dictates at least N memory units are available. In addition, no arriving packet may be placed in a memory unit that contains a packet with the same departure time. This results in additional $N-1$ memory units representing the $N-1$ packets having the same departure time as the arriving packet, that may have already been placed in the memory units. Should this condition not be satisfied, two packets will be required to simultaneously depart from a memory unit that can only produce one packet in each time slot.

The third and final condition states that all N arriving packets must be placed in different memory units (since each memory can only perform one write operation). By aggregating these three conditions, it is shown that at least $3N-1$ memory units must exist in order to guarantee FCFS output queueing emulation. Although this limit on the number of memories is sufficient, it has not been shown to be necessary. In fact, a tighter bound was recently found, suggesting that at least $2.25N$ memories are necessary [6]. Regardless of the precise minimal number of memories used, a key challenge relates to the practical realization of the memory management mechanism, *i.e.* the process that determines the memories in which arriving packet are placed. Observably, the above memory management algorithm requires N iterations to complete.

In [7][8] Prakash, Sharif, and Aziz proposed the Switch–Memory–Switch (SMS) architecture as an abstraction of the M-series Internet core routers from Juniper. The approach consists of statistically matching input ports to memories, based on an iterative algorithm that statistically converges in $O(logN)$ time. However, in this scheme, each iteration comprises multiple operations of selecting a single element from a binary vector. Although the nodes operate concurrently from an implementation perspective, these algorithms are $O(log^2 N)$ at best (assuming $O(logN)$ operations are needed for each binary iteration as stated above). Since timing is a critical issue, the computational complexity should directly reflect the intricacy of the digital circuitry involved, as opposed to the high-level algorithmic perspective.

1.3 Background

In this chapter, we summarize the current strategies employed in order to overcome the high memory bandwidth requirements imposed by output queued switches, and identify their key limitations. Moreover, we discuss the emulation of an output queued switch in the context of input queued and combined input and output queued (CIOQ) architectures, as commonly used in many of today's large scale routers. This is followed by a survey of existing shared memory architectures that serve as the basis for the work presented in this dissertation.

1.3.1 Emulation of Output Queued Switches

The memory bandwidth required to buffer arriving packets often dictates the switching capacity of a packet switching fabric. With increasing line rates and the demand for greater capacity, inherent memory bandwidth limitations imposed by output queued switches proved to be impractical for large scale implementation [9]. As a result, input queued switches became a more widely used framework for practical packet switching. Since packets are buffered directly as they arrive at their respective inputs, the memory bandwidth requirement is merely R, the data rate at each port. This is a sharp contrast to the $O(NR)$ memory bandwidth requirements of output queued switches, where N denotes the number of ports and R the data rate of each port. However, if the input buffer is FIFO and a packet at the head of an input queue is blocked, then all packets behind it are precluded from traversing the switch, even when the desired output link is idle. This problem is referred to as *head-of-line* (HOL) blocking. It has been shown that, given uniformly distributed i.i.d. traffic, HOL blocking limits the switch throughput to 58% of the aggregate bandwidth [10]. This problem can be eliminated altogether, if we allow each input to maintain a separate queue for each output. This approach is known as virtual output queueing (VOQ) [11][12][13]. A switch employing VOQs and a maximum weight matching algorithm can achieve the 100% throughput [14] provided by the output queued approach. However, the maximum weight matching algorithm is not practical to implement, as it has a lower-bound computational complexity of $O(N^{2.5})$.

An alternative approach is to retain the VOQs at the input, while also including queues at the output, and speedup the internal crossbar. This is yields a combined input and output queued (CIOQ) switch architecture. With an internal speedup of two or greater, a CIOQ switch can precisely emulate an output queued switch [15][16]. Hence, it is possible to obtain the desirable performance characteristics of an output queued switch. However, we contend that as line rates increase the scheduling algorithms required to emulate an OQ switch do not scale well, thus raising the clear need for alternative solutions.

1.3.2 Shared Memory Switch Architectures

The shared memory switch architecture utilizes a sharply different approach to queueing packets by providing only one stage of buffering. This buffering stage is placed between an interconnect located at the ingress and egress ports of the switch. This is in contrast to the CIOQ router which provides two buffering stages, one at the ingress and another at the egress, with the interconnect located in the middle. Shared memory architectures are generally distinguished by the techniques used to represent the single stage of buffering. One such distinction is the scheduling which falls into one of two classes: (1) routers which utilize a randomized switch scheme to achieve load balancing, or (2) deterministic routers which attempt to emulate conventional routers to provide delay guarantees for packets [5]. The memory management algorithms presented in this dissertation are deterministic, given they distribute packets in a structured manner across all memories located in the center stage of the switch.

Single Buffered Routers

A common attribute in single buffered (SB) routers is the use of a crossbar at the switch input to provide load balancing at the ingress port of the switch fabric [17]. Consider an input queued switch utilizing VOQs, the crossbar allows incoming packets to be evenly distributed in an effort to reduce delay [18] and present a uniform traffic distribution to the buffering stage. In the work presented by Chang [19][20] and later Keslassy [21], they exploit the uniformity of the traffic matrix to apply a Birkhoff-von Neumann capacity decomposition approach in order to achieve an online switch complexity of $O(1)$ and more

efficient use of the available buffer space. A drawback of this approach is that missequencing of packets is common. To address this problem, the use of a coordination buffer [22] is required.

Parallel Shared Memory Routers

Each of the various switch fabric architectures mentioned above assume that packet buffers operate at a rate, R, that is *at least* equivalent to the line rate. As the capacity of commercial routers increases by 2.2 times every 18 months, the random access time of electronic memory has increased by only 1.1 times over the same time frame [5]. Parallel shared memory (PSM) routers were proposed to distribute the capacity across a pool of K memories, located in a centralized location, operating below the line rate. Obviously, the pool of K memories must provide sufficient bandwidth such that all packets can be placed in a single time slot. If we assume that each of the shared memory units can perform at most one packet-read or -write operation during each time slot, then we would like to determine a sufficient number of memories required to emulate a FCFS OQ switch. This can be determined through the application of constraint sets analysis (also known as the "pigeon hole" principle), summarized in Section 1.2, to be $3N - 1$. Although this limit on the number of memories is sufficient, recent results suggest that $2.25N$ memories are necessary for exact emulation of a FIFO output queued switch [6].

Distributed Shared Memory Routers

The distributed shared memory (DSM) switch represents a slight variation on the PSM theme, whereby the pool of memories is no longer required to be located in a centralized location. Instead, the memories are located in linecards that are interconnected by a crossbar [5]. In such a configuration, arriving packets are immediately switched to a line card, using the crossbar, for storage until they are scheduled to depart. When the scheduled time arrives, the packet is read from the line card and switched, by means of a crosspoint switch, to the correct output. It is important to note that every packet must traverse the crossbar twice before reaching its destination. This results in a crossbar operating rate of $2R$. Moreover, the DSM switch requires a crossbar operating rate of $6R$ and a total mem-

ory bandwidth of $3NR$ to deterministically emulate a FCFS output queued router. The inefficiency of this architecture with regards to crossbar size and memory utilization limits its scalability.

1.4 Traffic Models for Performance Evaluation

Performance evaluation of packet switching architectures requires the use of traffic models to emulate the target environment of the switch in order to establish relevant metrics for comparison. In general, the two principal criteria that affect switch performance are the arrival process and the destination distribution. The arrival process defines the time correlation and/or dependence of incoming packets and is typically represented by two classes of traffic: bursty and non-bursty. The destination distribution corresponds to the probability that an arriving cell is destined to each of the output ports. The most common and straightforward case is the uniform distribution, whereby each packet has an identical likelihood of being destined to each output. Throughout this dissertation, we employ both bursty and non-bursty traffic arrival patterns in the course of our performance analysis. The following subsections describe various traffic models implemented in our performance simulation environment.

1.4.1 Bernoulli Arrivals

The arrival process consists of packets arriving in succession with some associated inter-packet delay. The simplest and most commonly deployed arrival process is Bernoulli i.i.d., which is a memoryless process generating an arrival with a given probability regardless of the history of arrivals. In each time slot, a packet is generated with probability p, and no packet is generated with probability $1 - p$, which means the inter-arrival time distribution is a Bernoulli distribution with a parameter p. Clearly, for the Bernoulli arrival process, the mean rate of the arrival process, or the average input offered load, denoted by λ, is equal to p.

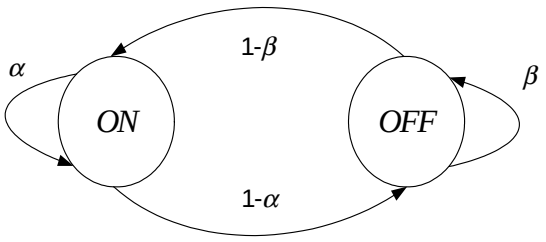

Figure 1-4: Two-state Markov-modulated arrival process.

1.4.2 ON/OFF Model

It has been extensively shown in the literature that typical network traffic tends to be correlated or "bursty" [23]. This is principally due to the diversity of modern networking applications and the aggregation of network traffic. The bursty traffic model employed in this dissertation is based on a two-state Markov chain, see Figure 1-4.

The two-state Markov chain generates arrivals according to an alternating ON/OFF source. For the two-state Markov modulated model presented in Figure 1-4, arrivals occur when the process is in the ON state, while no packets are generated while the process is in the OFF state. As a result, a stream of correlated bursts and silent periods, both geometrically distributed in length, is produced. The load offered by the two-state Markov modulated process is given by

$$\lambda = \frac{1-\beta}{2-(\alpha+\beta)}, \tag{1.1}$$

where α and β denote the probability of remaining in the ON and OFF states respectively. Furthermore, we represent the mean burst size, denoted by MBS, as

$$MBS = \sum_{i=1}^{\infty} i\alpha^{i-1}(1-\alpha) = \frac{1}{1-\alpha}. \tag{1.2}$$

From equations 1.1 and 1.2, we obtain the following relationship between the average input offered load and the mean burst size,

$$\lambda = \frac{MBS(1-\beta)}{MBS(1-\beta)+1} \Longrightarrow \lambda_{\max} = \frac{MBS}{MBS+1} \tag{1.3}$$

Since at least a single OFF state separates two consecutive bursts, the maximal arrival rate is bounded by $MBS/(1+MBS)$.

1.4.3 Destination Distribution

The destination distribution determines the destination output for incoming packets. Generally speaking, a destination distribution is characterized by a set of probabilities, p_{ij}, denoting the likelihood that an incoming packet at input i is destined to output j, and satisfying

$$\sum_{j=1}^{N} p_{ij} = 1.$$

Let λ_i denote the average input offered load for input port i, λ_j denote the average output offered load for output port j, and λ_{ij} the average number of packets arriving at input i and destined to output j. Then, resulting relationship follows

$$\lambda_{ij} = \lambda_i p_{ij} \tag{1.4}$$

$$\lambda_j = \sum_{i=1}^{N} \lambda_{ij} = \sum_{i=1}^{N} \lambda_i p_{ij}, \tag{1.5}$$

and the average output offered load is given by

$$\mu = \frac{1}{N} \sum_{j=1}^{N} \lambda_j. \tag{1.6}$$

Throughout this dissertation, we will consider only admissible traffic, whereby

$$\sum_{i} \lambda_{i,j} < 1, \sum_{j} \lambda_{i,j} < 1.$$

Uniform Destination Distribution

The most commonly deployed destination distribution is the uniform destination distribution, For this distribution, each packet is destined for a given output with identical probabilities such that $p_{ij} = \frac{1}{N}$, for all i and j. Clearly, equations 1.4, 1.5, and 1.6, from

the preceding section yield

$$\lambda_{ij} = \frac{\lambda_i}{N}, \quad \lambda_j = \frac{1}{N}\sum_{i=1}^{N}\lambda_i$$

such that the average output offered load is given by

$$\mu = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{N}\sum_{i=1}^{N}\lambda = \frac{1}{N}\sum_{i=1}^{N}\lambda_i.$$

The Zipf Destination Distribution

In practice, packet streams are not distributed uniformly across the destinations. Instead, traffic tends to be focused on preferred, or popular, destinations. As means of evaluating the performance of the proposed architectures, we employ a non-uniform destination distribution model that follows Zipf's law [24][25][26]. The Zipf law states that the frequency of occurrence of some events, as a function of the rank (m) which is determined by the above frequency of occurrence, is a power-law function, i.e. $P_k \approx 1/k^m$. It has been shown that many natural and human phenomena, such as Web access statistics, company size and biomolecular sequences, all obey the Zipf law with the order being close to 1 [23]. The probability that an arriving cell is heading to destination k was thus modeled by

$$Zipf_m(k) = \lambda_m = k^{-m}\left(\sum_{j=0,1,..N} j^{-m}\right).$$

While $m = 0$ corresponds to uniform distribution, as m increases the distribution becomes more biased towards preferred destinations. Clearly,

$$\sum_{j=1}^{N} p_{ij} = \sum_{j=1}^{N}\frac{j^{-r}}{\sum_{k=0}^{N}k^{-r}} = \frac{\sum_{j=1}^{N}j^{-r}}{\sum_{k=0}^{N}k^{-r}} = 1$$

$$\lambda_{ij} = \frac{j^{-r}\lambda_i}{\sum_{k=0}^{N}k^{-r}}, \ \lambda_j = \frac{j^{-r}\sum_{i=1}^{N}\lambda_i}{\sum_{k=0}^{N}k^{-r}}$$

$$\mu = \frac{1}{N}\sum_{j=1}^{N}\lambda_j = \frac{1}{N}\frac{\sum_{j=1}^{N} j^{-r}\sum_{i=1}^{N}\lambda_i}{\sum_{k=0}^{N} k^{-r}} = \frac{1}{N}\sum_{i=1}^{N}\lambda_i$$

1.5 Motivation

It is the overall goal of this work to propose practical designs for high capacity packet switches that:

- Emulate the desirable performance attributes of an output queued switch
- Provide a scalable solution that offers quality-of-service (QoS) guarantees and support for multicast traffic
- Facilitate the consolidation of as many packet switching functions as possible on a single chip

In the context of emulating an output queued switch, we would like to overcome the high memory bandwidth requirements imposed by traditional shared memory architectures, yet remain practical so as to yield a path towards practical hardware implementation. In particular, this dissertation advocates a parallel shared memory approach whereby packets are distributed across a finite number of slow-running memories. A core challenge pertains to the realization of a memory management algorithm with a computational complexity of $O(\log N)$, and can utilize straightforward multiplexers to switch data segments from one location to another, as opposed to necessitating a crosspoint switch. Acceptable costs, according to the paradigm fostered here, include fixed latency and a reasonable increase in the number of memory units employed.

To illustrate the practicality of the FoC approach, we propose a novel pipelined memory management algorithm that is capable of providing the same QoS attributes found in output queued switches. We demonstrate the proposed memory management algorithm by means of hardware implementation and provide metrics related to its performance and resource requirements.

1.6 Dissertation Outline

In Chapter 1, we introduced the notion of Fabric on a Chip, the motivation for this approach, and the strong desire to emulate an output-queued switch in the proposed framework. Additionally, the primary features and limitations of existing packet switching architectures, with respect to their ability to emulate the desirable attributes of an output queued switch, were described.

In Chapter 2, a novel performance analysis for output queued cell switches with general independent heterogeneous traffic is introduced. In particular, we present an arrival process in which input ports generate bursty streams that are non-uniformly distributed across the outputs, and derive an approximation for related queue size distributions. This effort outlines a methodology for obtaining bounds on the behavior and expected performance characteristics of output queued switches under a wide range of correlated traffic scenarios.

Chapter 3 introduces two novel memory management algorithms for parallel shared memory (PSM) switches. Each scheme proposed utilizes a pipelined hardware architecture to achieve the throughput gain required. The algorithms exploit parallelism of the packet placement process, thereby gaining execution speed at the expense of a fixed latency. In Chapter 4, we present enhancements to provide QoS guarantees, multicast traffic support, and offer load balancing. This chapter concludes with a brief comparison of the proposed architectures to current switch fabric solutions.

In Chapter 5, we provide implementation results relating to the hardware implementation of the proposed memory management algorithms. Additionally, we introduce a reconfigurable high-speed hardware architecture for heterogeneous mutlimodal packet generation in an effort to generate real-time stimulus for verification of packet switching architectures. Chapter 6 provides a summary of the contributions made thus far and outlines the plan for work to be completed prior to the defense of the dissertation.

Chapter 2

Analysis of Output Queued Cell Switches with Heterogeneous Traffic

Output queued switches have been extensively studied in the literature. To a large extent, they represent the theoretical limit on the performance that can be achieved in any space-division switching fabric [10]. Consequently, performance analysis of input queued switches is commonly carried out in comparison to that of an output queued router [10][27][28]. Pragmatic output queued switching fabrics, such as those based on shared memory architectures [29], have been deployed in switches and routers. However, the majority of the studies performed on these systems considers traffic that obeys a Bernoulli (uncorrelated) process. Moreover, in most cases uniform destination distribution is assumed such that all input ports offer the same load intensity to all output ports.

Recall from Section 1.1, that arriving cells in an output queued router architecture traverse the switching fabric directly to their designated outputs, without being queued or delayed in any way at the ingress (input) stage. Such a scheme requires that a dedicated link, be it logical or physical, exist between each input port and each output port. In other words, N^2 such links are needed for an $N \times N$ switch. A key advantage of output queued switches is that of minimal latency and controllable QoS provisioning, both a result of the

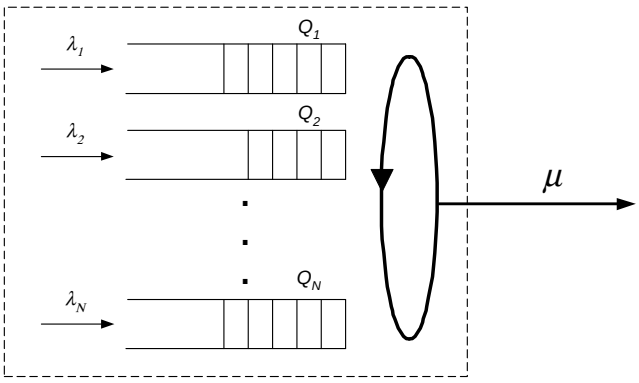

Figure 2-1: A basic logical model of the queueing architecture employed at an egress port of an output queued switch.

fact that arriving cells progress towards their destination without any impediment at the ingress. In practical implementations, a shared-memory architecture is employed, whereby multiple arriving cells are stored in a single physical memory unit. If a single memory unit is utilized at each output port, an $O(NR)$ memory bandwidth requirement results, where R denotes the cell arrival rate. Figure 2-1 depicts a typical model for queueing architectures in output queued switches, whereby each port generates cells at a different mean arrival rate.

In this chapter, analysis for output queued switches is presented with non-uniformly distributed bursty arrivals that are generated using a multitude of ON/OFF arrival processes. Random arbitration is applied between the queues such that non-empty queues compete equally for service during each time slot. A random arbitration scheme is considered primarily since it represents, from a hardware implementation perspective, a simple, scalable approach to arbitrating between multiple queues. More sophisticated arbitration schemes, such as those considering information regarding the queues' states, are expected to at least match, if not exceed, the performance of random arbitration. In this respect, random arbitration provides a lower bound on the attainable performance of the generic switch architecture considered. Based on the per-queue probability generating functions of the interarrival times distribution, it is shown that accurate depiction of the queues' behavior can be obtained.

2.1 Queuing Model with Bursty Arrivals

2.1.1 Notation and Formulation

Packets may vary in size as they arrive at the switch ports. In typical switching platforms, a segmentation module partitions packets into fixed-size cells that are later reassembled at the egress modules prior to departing the switch. Processing fixed-size data units has proven both practical and easier to study. To that end, all data units traversing the switch fabric are assumed to be of fixed size. We consider a discrete-time queueing system with N queues of infinite buffer capacity and a single server, in which all events occur at fixed time slot intervals. Within each time slot, at most N arrivals may occur, originating from the N inputs. Consequently, at most a single departure occurs servicing one of the non-empty queues at each time slot.

We model the service distribution as a memoryless process for which there is a constant probability of service, during each time slot. The aggregate service rate is, respectfully, defined as

$$\mu = \sum_{k=1}^{N} \mu_k \tag{2.1}$$

In order for the system to be stable, we require that the service rate, μ, exceed the aggregate steady-state rate of arrivals such that

$$\sum_{j=1}^{N} \lambda_j < \mu \tag{2.2}$$

For a stable system, we can state that for each queue, the probability of arrival, must converge to, or even equal, the departure rate such that

$$\lambda_k = \mu_k \left(1 - \pi_0^{(k)}\right) \tag{2.3}$$

where μ_k denotes the probability that queue k is serviced given that the queue is non-empty, and $\pi_0^{(k)}$ is the probability that the queue is empty.

2.1.2 ON/OFF Arrivals with Geometric Service Times

It has been shown in the literature [30] that in a GI/Geo/1 discrete-time queueing system (independent arrivals times and geometrically distributed service times), if $f_n\,(n \geq 1)$ is the interarrival time distribution, with a p.g.f. $F\,(z) = \sum_{n=1}^{\infty} f_n z^n$, and queue service times are geometrically distributed with parameter μ_k, then the stationary queue size distribution,$\pi_m = (1-\gamma)\,\gamma^m \;\; m \geq 0$, as viewed by an arriving cell will always be in the form

$$\pi_m = \begin{cases} 1-\xi & m = 0 \\ \xi\,(1-\gamma)\,\gamma^m & m \geq 1 \end{cases} \tag{2.4}$$

where γ is a unique root of the equation that lies in the region $(0, 1)$ and ξ is constant. The latter is, by definition, independent of arrivals. Hence utilizing (2.4) to derive ξ yields the first moment

$$E\,[Q] = \sum_{m=1}^{\infty} m\pi_m = \frac{\xi}{(1-\gamma)} = \frac{\lambda}{\mu\,(1-\lambda)} \tag{2.5}$$

which provides us with the mean queue occupancy. Employing Little's results [30][31], the mean latency is given by

$$E\,[W] = \frac{1}{\mu\,(1-\gamma)} \tag{2.6}$$

A late arrival model is considered, for reasons of convenience, such that within a time slot boundary a departure will always precede an arrival event. We observe the queue size at instances following the arrival phase, such that time slot boundaries are delimited by the observation instances. Consider a discrete-time, two-state Markov chain generating arrivals modeled by an ON/OFF source which alternates between the ON and OFF states. While in the ON state, a single cell is generated (per time slot). Let the parameters p and q denote the probabilities that the Markov chain remains in states ON and OFF, respectively. An arrival is generated for each time slot that the Markov chain spends in the ON state. The result is a stream of correlated arrivals and silent periods, both of which are geometrically distributed in duration.

It can easily be shown that the parameters p and q are interchangeable with the mean arrival rate, $\lambda = (1-q)\,/\,(2-q-p)$, and mean burst size, $B = 1/(1-p)$. Consequently, the offered load is identical to the steady-state portion of the time the chain spends in state

ON. Recalling the notation f_n for the interarrival times distribution, the probability of two consecutive arrivals occurring is identical to the probability that following an arrival the Markov chain remains in state ON, i.e. $f_1 = p$. Similarly, f_2 is the probability that following an arrival, the chain transitions to the OFF state and then returns to the ON state. For $n > 2$, it is apparent that following a transition from the ON state to the OFF state, there are $n - 2$ time slots during which the chain remains in the OFF state before returning to the ON state. Accordingly, we obtain the following general expression for f_n:

$$f_n = \begin{cases} p & n = 1 \\ (1-p)q^{n-2}(1-q) & n > 1 \end{cases} \tag{2.7}$$

The corresponding p.g.f. is

$$F(z) = pz + (1-p)(1-q)\frac{z^2}{1-qz} \tag{2.8}$$

Next we solve the equation $z = F(z\mu + (1-\mu))$ to find that the root in the region $(0, 1)$ is

$$\gamma = \frac{(1-\mu)}{\mu}\left\lfloor \frac{1}{\mu(1-p-q)+q} - 1 \right\rfloor \tag{2.9}$$

Examining the condition $\lambda < 1$, which must be satisfied for stability, yields the anticipated inequality

$$\mu > \frac{1-q}{2-p-q} = \lambda \tag{2.10}$$

2.2 Output Queued Switch Model with Bursty Arrivals

In the investigated system, random arbitration is employed, suggesting that during each time slot for which a service event occurs, one of the non-empty queues is randomly selected for transmission in an unbiased manner. The latter implies that the service discipline to each queue is also memoryless since during each time slot no information regarding previous service cycles is considered. As such, we will exploit the results for the GI/Geo/1 queueing system to derive approximate behavioral analysis of the individual output queues.

We focus our analysis on queue k observing that three conditions must be met during

each time slot for the queue to be serviced: (1) service must be granted to the port (output link not congested), (2) the queue must be non-empty and, finally, (3) the queue must prevail when equally contending against the other non-empty queues. While the first two conditions are rather straightforward, the third condition requires some elaboration. Assuming that queue k is non-empty ($Q_k > 0$), it has an equal probability of being selected for transmission as any other non-empty queue. The mean number of non-empty queues, excluding queue k, can be approximated by $\sum_{j \neq k} \left(1 - \pi_0^{(j)}\right)$ where $\pi_0^{(j)}$ denotes the stationary probability that queue j is empty, provided that queue k is non-empty. To obtain the mean size of the contending set, we then add one to the mean number of non-empty queues. By multiplying the expressions for the three conditions stated above, we find the probability of departure from queue k to be

$$\mu_k \cdot \left(1 - \pi_0^{(k)}\right) = \frac{\mu\left(1 - \pi_0^{(k)}\right)}{\sum_{j \neq k}\left(1 - \pi_0^{(j)}\right) + 1} = \frac{\mu\left(1 - \pi_0^{(k)}\right)}{N - \sum_{j \neq k} \pi_0^{(j)}} \tag{2.11}$$

Since the arrival rate should converge to, or even equal, the departure rate, we equate (2.11) to the rate of arrivals for each queue, yielding

$$\lambda_k = \frac{\mu\left(1 - \pi_0^{(k)}\right)}{N - \sum_{j \neq k} \pi_0^{(j)}} \tag{2.12}$$

The latter holds when we assume that for large values of N the conditional probability of each queue being empty, given the size of other queues, converges to the unconditional probability of that queue being empty. Hence, we have N linear equations for the N variables $\pi_0^{(j)}$ $(j = 1, 2, ..., N)$. Solving for $\pi_0^{(j)}$, we directly obtain μ_k, the probability of service to each of the queues.

We next turn our attention to the case where each input produces a stream of bursty arrivals modeled by a unique ON/OFF process, with respective parameters p_k and q_k pertaining to traffic originating at input (source) k. In view of the fact that queue sizes are geometrically distributed, based on (2.9) the respective parameters of these distributions

are

$$\gamma_k = \frac{(1-\mu_k)}{\mu_k}\frac{(1-\mu)}{\mu}\left\lfloor \frac{1}{\mu_k(1-p_k-q_k)+q_k} - 1 \right\rfloor$$

We note that, as expected, γ_k is a function of both the arrival model parameters and the rate at which the output queues are serviced. Fundamental performance metrics, such as the mean cell latency, are directly derived for each of the queues as shown in (2.6).

2.3 Simulated Performance of an Output Queued Cell Switch

Our simulations pertain to a scenario where traffic is both bursty and non-uniformly distributed between the inputs. Arrivals are generated by an ON/OFF model which is independently operated for each input. As means of validating the analytical deductions with simulation results, we employ a non-linear destination distribution model named Zipf's law [32][33]. The Zipf law states that the frequency of occurrence of some events, as a function of the rank (m) where the rank is determined by the above frequency of occurrence, is a power-law function: $P_k \sim \frac{1}{k^m}$. Accordingly, the probability that an arriving cell is heading to destination k is given by

$$\lambda_k^{(m)} = k^{-m}\left(\sum_{j=1}^{N} j^{-m}\right)^{-1} \tag{2.13}$$

While $m = 0$ corresponds to a uniform distribution, as m increases the distribution becomes more biased towards preferred destinations. There has been recent evidence that the Zipf model accurately portrays web caching and access statistics, in particular when the parameter m is close to unity. Recent studies have illustrated the presence of Zipf law characteristics in Internet traffic patterns [34].

Previous work [10] has shown that the mean queue size in an output queued router with First-in-First-Out (FIFO) arbitration and traffic arriving uncorrelated and uniformly distributed, is

$$E[Q]_{FIFO_{BERNOULLI}} = \frac{\lambda^2(N-1)}{2(1-\lambda)N}, \tag{2.14}$$

where N denotes the number of ports in the switch. Uniformly distributed ON/OFF arrivals arriving at an output queued router employing FIFO arbitration yield the following mean

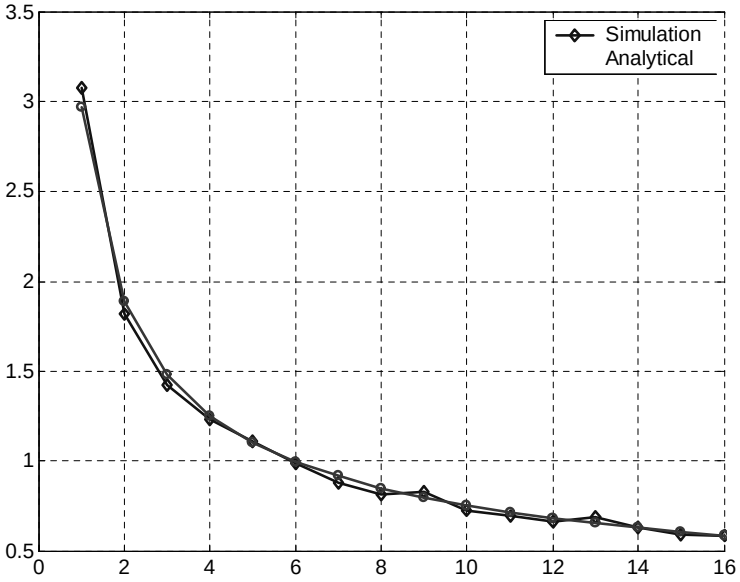

Figure 2-2: Mean queue occupancy for each of the output queues in a 16-port switch where $\lambda = 0.8$ and $\mu = 1$. Arriving traffic is distributed according to the Zipf$_{k=0.5}$ with mean burst sizes of 6 cells.

queue size

$$E[Q]_{FIFO_{ON-OFF}} = \frac{\lambda B (N-1)}{(1-\lambda) N}, \tag{2.15}$$

where B is the mean burst length. Analysis that provides foundations for the above can be found in [35]. By dividing each of the above mean queue sizes by the normalized offered loads, we obtain the corresponding mean cell latencies. We shall refer to these assertions when comparing the latency of the FIFO discipline to that of random arbitration.

Figure 2-2 illustrates the mean queue occupancy for each of the output queues in a 16-port switch, where $\lambda = 0.8$ and $\lambda = 1$. Arriving traffic is distributed between the queues according to a Zipf$_{m=0.5}$ distribution with a fixed mean burst size of 6 cells. As can be observed, the latency for each queue is well correlated with its share of the offered load. The two curves correspond to the simulated model and analytical approximation, as described in section 2.2. The simulation results clearly validate the accuracy of the proposed analytical inference. For alll simulation results presented in this section, the duration in

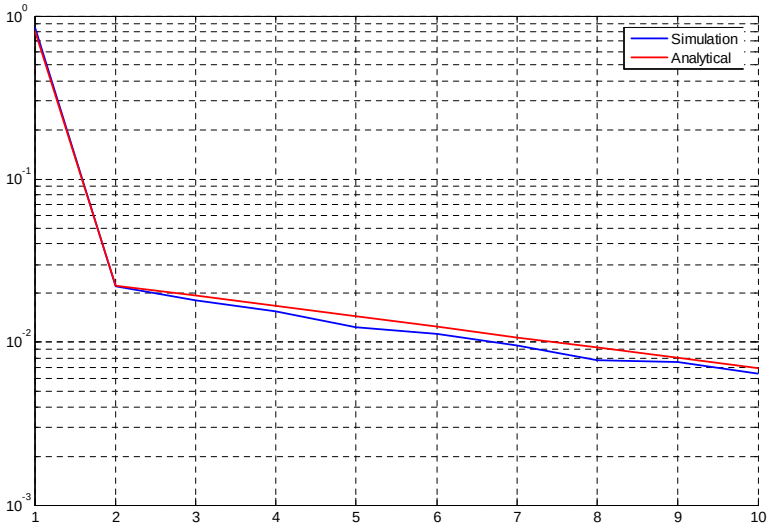

Figure 2-3: Queue size distribution for a 16-port switch with normalized offered load of 0.75, mean burst size of 8 cells and a probability of service of 0.9.

each case was 100,000 time slots to ensure a consistent steady state response.

In Figure 2-3, the queue size distribution is shown, for a 16-port switch that is introduced with uniformly distributed bursty traffic and an aggregate normalized offered load of 0.75. The mean burst size is 8 cells, while the probability of service (μ) is 0.9. The latter can reflect, for example, on a system that has a 10% congestion (no-service) time.

In Figure 2-4, the mean queue latency is presented as a function of the offered load for both random and FIFO arbitration. Traffic is assumed to be governed by the ON/OFF model with a mean burst size of 8 cells. Results for the case of random arbitration are shown for different probability of service values, the impact of which is clearly observable in particular as the offered load approaches the service rate. Also shown are the mean latency attributes of FIFO arbitration, which in many cases constitute the theoretical lower limit on the delay through a space division switch. For the case of $\mu = 1$, FIFO and random arbitration yield identical results, suggesting that the presented methodology for obtaining the latency under generic traffic conditions provides valuable approximation to a pure FIFO system, as would be expected.

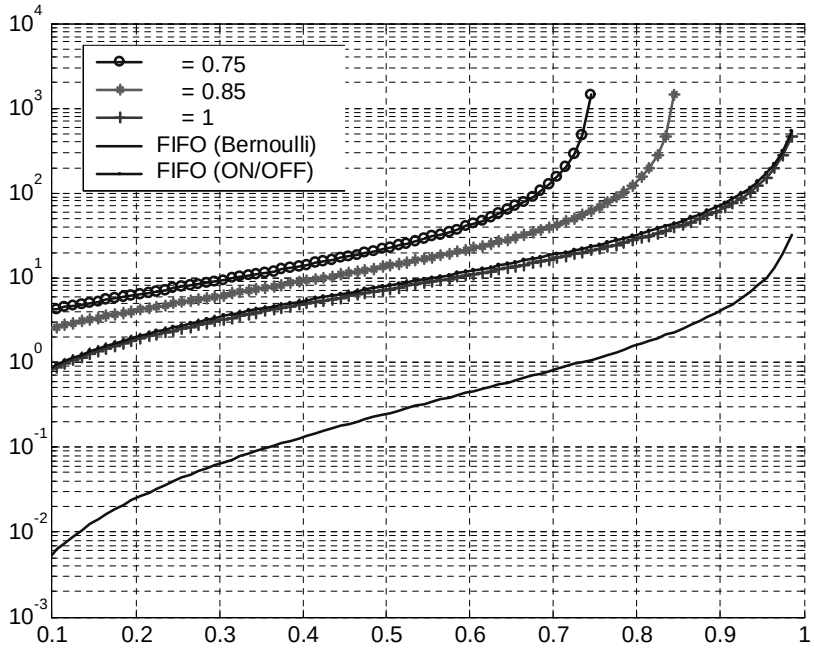

Figure 2-4: Mean queueing latency as a function of the offered load for random and FIFO arbitration schemes.

Chapter 3

Scalable Packet Placement Algorithms for PSM Switches

In this chapter, we present two novel pipelined memory management architectures which play a key role in the context of on-chip output-queued switch emulation. We discuss in detail the fundamental properties associated with each of the proposed schemes, along with techniques for accelerating each architecture to achieve greater port density and further the consolidation of packing switching functions on a single chip.

3.1 PSM Switch Architectures

This section focuses on the proposed pipelined memory management architecture for PSM switches, as generally illustrated in Figure 1-3. The basic assumption in output queued switching is that an arriving packet is assigned a departure time. The latter reflects on the number of time slots that the packet is to remain within the switch prior to departing. Consequently, the first step is to calculate the departure time for each packet, prior to the insertion of packets into the memory management subsystem. This process is defined by the output scheduling algorithm employed, and is generally very fast. The most straightforward scheduler is the first-come-first-serve (FCFS), in which packets are assigned departure times in accordance with their arrival order. To provide delay and rate guarantees, more sophisticated schedulers [36] can be incorporated, as reflected by the departure time assignments.

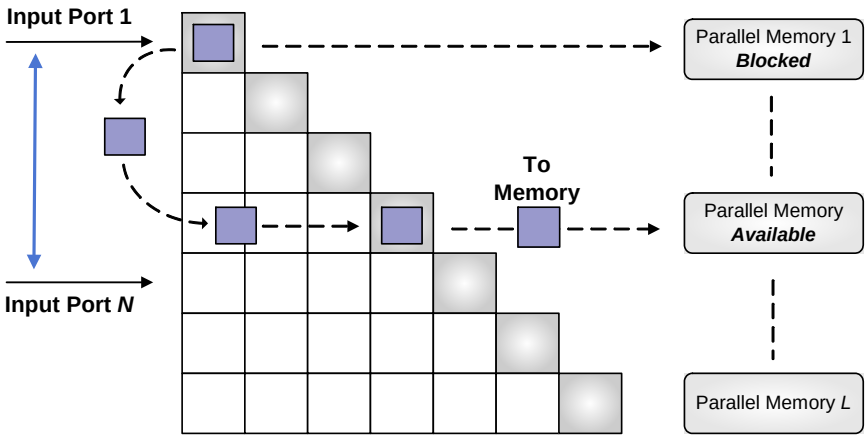

Figure 3-1: Memory-management pipeline architecture

The main contribution here lies in the fact that the placement algorithm distributes the packet-placement process, thereby gaining execution speed at the cost of a fixed latency.

3.2 Row-associative memory management algorithm

The first router architecture we will propose is the row-associative memory management algorithm. For this router architecture, the memory-management algorithm is implemented using a multi-stage pipeline architecture, as depicted in Figure 3-1. The pipeline architecture consists of $\frac{L(L+1)}{2}$ cell buffering units arranged in a triangular structure. Each row is associated, or coupled, with one of the parallel shared memory units. Hence, the architecture requires L parallel shared memories. Incoming packets arriving at input port i are initially inserted into row i. The underlying mechanism is that at every time slot, packets are horizontally shifted one step to the right, with the exception being cells located on the diagonal of the structure. The diagonal cells, in this scheme, have the ability to move vertically to another row in the same column. A cell moves vertically if any of the following two conditions are met: (1) the memory associated with the row in which it is currently located already contains a packet with the same departure time; (2) there is another cell ahead of it in the same row with the same departure time. Therefore, vertical moves are used as means of resolving memory placement contentions. The goal of the placement

scheme is that once a packet reaches the last column of the pipeline, it is guaranteed to be located in a row which is associated with a memory that does not contain any packets with the same departure time.

3.2.1 Memory Management Algorithm

In this section, we provide some complexity analysis results pertaining to the memory requirements of the row associative memory management algorithm. In the proposed model, rows of the pipeline are arranged in $\mathcal{B}$ sequential blocks, whereby there is one row per input port (for total of N rows) in the first block. As such, every input port writes its packets into one row. A cell in block r can only jump vertically to a cell in block $r+1$. In order to illustrate the underlying memory-management principal, we shall refer to the following example.

Example 1 *Consider the simple scenario depicted in Figure 3-2. The state of the pipeline for four consecutive time slots (i.e. t, $t+1$, $t+4$, $t+5$) is shown. At time t, there are two packets in the second row and three packets in the third row, all with the same departure time D. Two packets are located at diagonal positions, and since there are two packets ahead of them with the same departure time, they shift down to a row in the second block (row 6) and then move one position to the right, as shown in the pipeline diagram for time $t+1$. Note that the packets were not permitted to move to row 5 since their position is already occupied with two other packet with departure times X and Y. At time $t+4$, the second packet with departure time D in row 6 reaches the diagonal element, and, since there is another packet ahead of it with same departure time, it moves vertically to a row in the third block (row 7), followed by a shift one step to the right, as shown in the $t+5$ diagram. In this example, the pipeline consisted of three blocks of rows with 4, 2 and 1 rows in each, respectively. We emphasize that the notion of blocks is used only for illustration purposes. As will later be discussed, partitioning the structure into these blocks facilitates the complexity analysis as well as memory requirements of the architecture.*

Lemma 1 *There should be at least $2N$-r rows in block r, for $r \in [2, 3, \ldots \mathcal{B}]$.*

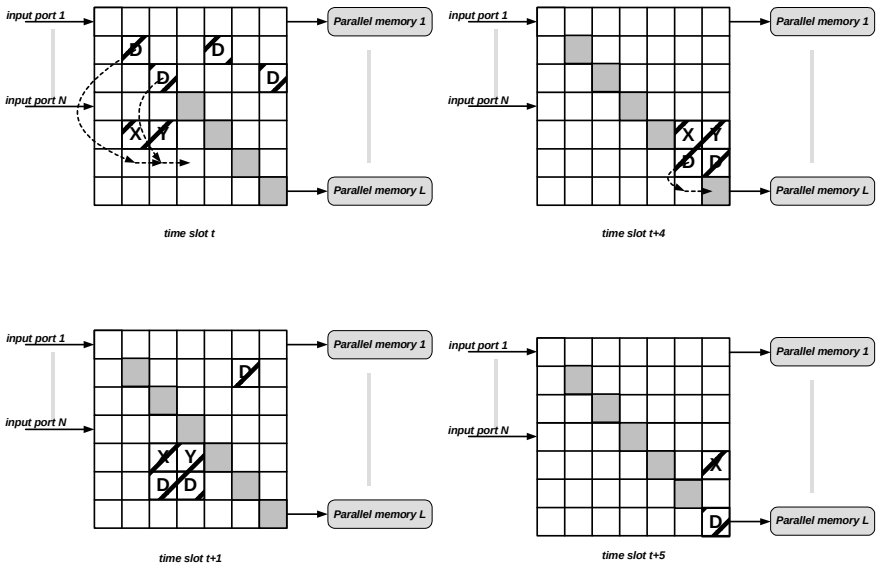

Figure 3-2: Example illustrating the proposed memory-management algorithm for a 4-port switch. The state of the pipeline structure is depicted for 4 consecutive time slots.

Proof. Consider a cell that is shifting vertically from block r to block $r+1$. It will find at most $N-1$ rows blocked by other packets, since there have been at most $N-1$ other packets that may have arrived at the same time as this packet and may have shifted down in prior time slots. Moreover, it will find at most $N-r-1$ other rows having packets with the same departure time. Note that there could be at most $N-1$ other packets in the system with the same departure time; however, r of them have already been accounted for having caused this packet to jump to block $r+1$. Therefore, there are at most $2N-r-2$ rows (or locations) that this packet cannot move to in block $r+1$. Using the pigeon-hole principle, we conclude that $2N-r$ is a sufficient number of rows for block $r \in [2, 3, \ldots \mathcal{B}]$.

■

From lemma 1, for a switch with N ports and $\mathcal{B}$ blocks, the total number of rows (parallel memories) can be expressed as:

$$\begin{aligned} L(N) &= N + (2N-2) + (2N-3) + ... + (2N-\mathcal{B}) \\ &= N + 2N(\mathcal{B}-1) - (\mathcal{B}+2)(\mathcal{B}-1)/2 \\ &= (2\mathcal{B}-1)N - (\mathcal{B}+2)(\mathcal{B}-1)/2 \end{aligned} \tag{3.1}$$

In order to compute number of the rows, we have to find $\mathcal{B}(N)$, which directly reflects the maximum number of vertical shifts a packet can perform prior to being successfully assigned to a memory. We shall refer to Example 1. and derive an upper limit on the number of conflicting packets with the same departure time in the fourth block, i.e. after three vertical shifts have occurred. We would like to find an upper limit on the number of conflicting packets after three shifts and use that recursively to obtain the maximum number of jumps, and hence blocks, that are sufficient to guarantee that each arriving packet will be successfully assigned a memory.

Lemma 2 *The maximum number of packets with the same departure time in the fourth block is* $\left(\sqrt{N}\text{-}1\right)$

Proof. Suppose that there are P_1 packets with the same departure time in the first block. Throughout the proof, we shall refer to this set of packets having the same departure time. Note that $P_1 \leq N$, since there cannot be more than N packets with the same departure time in the system. Let us assume that these packets are located in $R_1 (R_1 \leq N)$ rows of the first block. Therefore, the number of packets that move vertically to the second block will be $P_2 = P_1\text{-}R_1$. Next, we compute the number of rows in the second block that contain one of these packets. There are $N_2 = N_1\text{-}R_1$ packets that have moved to the second block, and the maximum number of packets that can shift simultaneously to the same row is R_1. Hence,

$$R_2 \geq \left\lfloor \frac{P_1\text{-}R_1}{R_1} \right\rfloor \tag{3.2}$$

Therefore, the maximum number of packets with the same departure time that can move to the third block is given by

$$P_3 = P_2\text{-}R_2 \leq P_1\text{-}R_1\text{-}\left\lfloor \frac{P_1\text{-}R_1}{R_1} \right\rfloor \tag{3.3}$$

If N_1-M_1 is divisible by M_1, then

$$P_3 \leq P_1(1\text{-}\frac{1}{R_1})\text{-}R_1\text{+}1 \tag{3.4}$$

otherwise, since $N_4 \leq N_3$-1, we have

$$\begin{aligned} P_3 &\leq P_1(1\text{-}\frac{1}{R_1})\text{-}R_1\text{+}2 \\ P_4 &\leq P_1(1\text{-}\frac{1}{R_1})\text{-}R_1\text{+}1 \end{aligned} \tag{3.5}$$

The maximum value of the expression in 3.5 is reached when $R_1 = \sqrt{P_1}$. Substituting $P_1 = N$, yields the inequality

$$P_4 \leq \left(\sqrt{N}\text{-}1\right)^2 \tag{3.6}$$

■

Note that if N is a complete square we have,

$$P_3 \leq \left(\sqrt{N}\text{-}1\right)^2 \tag{3.7}$$

In the following corollary, we exploit these results to determine the order of the memory blocks.

Corollary 1 *A sufficient number of parallel memory blocks required for an NxN switch, employing the proposed architecture, is $O(\sqrt{N})$*

Proof. Equation 3.6 shows that for an N-port switch, the maximum number of conflicting packets with the same departure time in the fourth block is $\left(\sqrt{N}-1\right)^2$. Let $\mathcal{B}(N)$

represent the number of stages required for an N-port switch. We thus have,

$$\begin{aligned} \mathcal{B}(N) &= \mathcal{B}(N\text{-}2\sqrt{N}+1)+3 \\ \mathcal{B}(1) &= 1 \end{aligned} \tag{3.8}$$

from which we conclude that $\mathcal{B}(N) = O(\sqrt{N})$. ■

Theorem 1 *For an N-port switch, where $N \leq k^2$ $k \in \{1, 2, \ldots.\}$, the number of memories is*

$$L(N) \leq 4k^3\text{-}5k^2+k+1 \tag{3.9}$$

with equality if $N = k^2$.

Proof. We prove the equality for $N = k^2$, suggesting that the general case trivially follows. We first show by induction that the number of required row blocks are $\mathcal{B}(k^2) = 2k$-1. For k=1, the result is trivial. We assume that the result holds for k and infer it for k+1. For $N = (k+1)^2$, using lemma 2 and (3.2) (given that N is a complete square), we have

$$\mathcal{B}((k+1)^2) = \mathcal{B}(k^2) + 2 = 2k - 1 + 2 = 2(k+1) - 1 \tag{3.10}$$

Then, we utilize the relationship proven for L in (3.1) to obtain

$$\begin{aligned} L(k^2) &= (2\mathcal{B} - 1)N - (\mathcal{B} + 2)(\mathcal{B} - 1)/2 \\ &= (4k - 3)k^2 - (2k + 1)(2k - 2)/2 \\ &= 4k^3 - 5k^2 + k + 1 \end{aligned} \tag{3.11}$$

■

The above theorem states that the number of parallel memories required in this architecture is $O(N^{1.5})$. Given that the minimum number of memories is $O(N)$, the increased memory requirement observed is the price paid in order make the assignment problem parallel and feasible from an implementation perspective. For example, with $N = 16$ (i.e. k=4), the number of parallel memory elements is 177, arranged in 7 blocks. To illustrate a case whereby this memory requirement is reached, we refer to the following example.

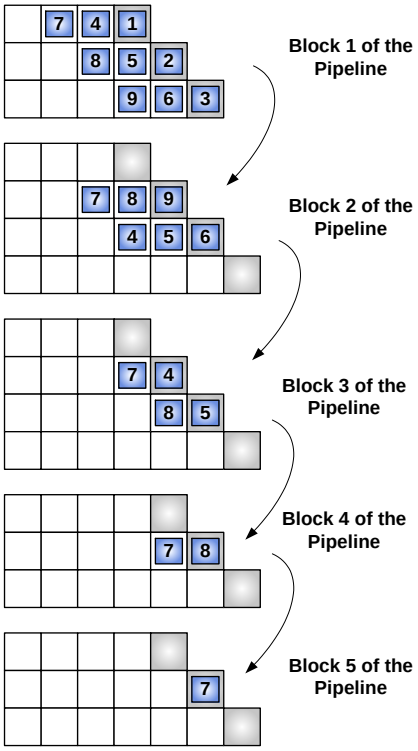

Figure 3-3: Adversarial scenario demonstrating the sufficiency bound on the number of memory units for a 9-port switch.

Example 2 *We refer to the following adversarial scenario pertaining to a 9-port switch, which is easily extendable to larger switch sizes when N is a complete square. Consider the settings illustrated in Figure 3-3. There are nine packets with the same departure time residing in the first block of memory. The first three packets are scheduled, while the others are shifted vertically to the second block. Note that packets 4, 5 and 6 reach the diagonal together and hence move down simultaneously. As shown, they may end up in the same row. A similar pattern of behavior is observed for packets 7, 8 and 9. Packets 6 and 9 are scheduled in the second block, while packets 5, 8 and 4, and 7 move to the third block. Once again packets 5 and 8 move simultaneously to the same row and packets 4 and 7 to a different row. Clearly, after two moves we end up with a set of four conflicting packets, i.e. from $k = 3$ we reach a condition with $k = 2$. Similarly, we need two more shifts to reach a point with only one packet in a block.*

3.2.2 Accelerating the Memory Management Algorithm

Accelerated Packet Switch Architecture

In this section, we extend the memory management algorithm presented in Section 3.2.1 by introducing computation and memory speedup components such that a more efficient high-speed packet placement algorithm is attained, yielding higher system scalability. In doing so, we maintain an identical pipeline architecture consisting of $\frac{L(L+1)}{2}$ cell buffering units arranged in a triangular structure, where L denotes the number of parallel memory units. By introducing the notion of speedup, s, we now require that the pipeline operate s times faster than the line rate. An immediate benefit of operating the pipeline at a higher rate is reduced system latency. Moreover, if the incoming packets, from the set of N input ports, are presented to the pipeline in groups of $\frac{N}{s}$, the number of conflicts from packets with the same arrival time is reduced from N to $\frac{N}{s}$.

The introduction of speedup, in the context of reducing arrival conflicts, implies that incoming packets from input port i be initially inserted into row $i \bmod \left(\frac{N}{s}\right)$. The underlying mechanism remains the same in that at every time slot, packets are horizontally shifted one step to the right, with the exception of the diagonal cells. A packet residing in a diagonal cell is either shifted (moved) vertically to another row in the same column or placed in the memory associated with the row in which it resides. Vertical packet shifts occur if the memory associated with the row in which the packet resides contains another packet with the same departure time. If a vertical shift is to be performed, the diagonal cell must select a row in its column that satisfies the conditions outlined by the "pigeon-hole principle". Applying these constraints, vertical moves provide a mechanism for resolving memory placement contentions. The goal of the scheme remains that once a packet reaches a diagonal cell in the pipeline it has exclusive access to the memory located in its row. If the current row memory is occupied, an attempt is made to place the packet in a row for which there are no existing conflicts

Placement decisions along the diagonal are made concurrently and independently to maximize the processing speed of the system. As selections are independently made for each packet, it is possible for packets along the diagonal to simultaneously select the same

row. In the single packet placement scheme presented in Section 3.2.1, there exists only one memory location in a row for any given departure time. To reduce the number of conflicts associated with packets simultaneously selecting the same row, the number of memory locations in a row for a given departure time can be increased to $m > 1$. As multiple packet placements to a single memory are now allowed, we must guarantee that m packets can be read from memory during a single packet time. One might speculate that the pipeline speedup, s, and the number of placements allowed, m, which is effectively the memory read rate, must be equal. This is generally not required, since it might be necessary to operate the pipeline at a slower rate as dictated by potentially faster on-chip SRAM resources. In this case, it is still prudent to offer additional placement locations in order to minimize conflict.

In provisioning m packet placement locations for each memory, it would appear that the reduction in row memories is merely an inconsequential outcome of increasing the memory depth. As packets shift vertically from block b to $b+1$, the block size, in terms of physical rows, decreases as s and m increase. This suggests that a vertical movement bypasses fewer potentially acceptable rows with each subsequent placement. In subsequent sections, we provide analysis that derives optimal values for these parameters. In order to illustrate the underlying memory-management principal, we refer to the following example.

Memory Requirements for the Accelerated Packet Placement Algorithm

In this section, we obtain an upper bound on the number of memories sufficient for the pipelined memory management architecture, given a speedup factor, s. In constructing the proof as we did in the previous section, we view the pipeline rows as arranged in $\mathcal{B}$ sequential blocks. Speedup is introduced into the system through the partition of the N arriving packets into s distinct segments. Packets that arrive to any of the N ports, at time t, are presented to the first block which consists of $\frac{N}{s}$ rows. Arriving packets are then multiplexed and written to one of the $\frac{N}{s}$ rows in this first block. Once placed in a row, a packet can only be written to one of m memory locations for a given departure time, or shift vertically to another row in block $b+1$.

Lemma 3 *There should be at least* $\left(\frac{s+m}{sm}\right) N - b$ *rows in block b, for* $b \in [2, 3, ...\mathcal{B}]$.

Proof. Consider a packet moving from block b to $b+1$. For a system with speedup s, it will find at most $\frac{N}{s}-1$ packets having the same arrival time. Furthermore, there are at most $N-bm-1$ packets with the same arrival time, since at least, bm packets with the same arrival time are served in the first b blocks. Therefore, in block $b+1$, there are at most $\frac{N}{s}-1$ rows occupied with packets with the same arrival time. Since up to m packets with the same departure time can be served with one memory, we need $\frac{N-bm}{m}$ additional rows for packets with the same departure time. Hence, we need $(\frac{N}{s}-1)+(\frac{N-bm}{m})$ rows for block $b+1$, or $(\frac{s+m}{sm})N-b$ rows for block $b \in [2,3,..,\mathcal{B}]$. ■

For a switch with N ports and P blocks, the total number of rows (parallel memories) can be expressed as:

$$\begin{aligned} L(N) &= \frac{N}{s}+\left(\left(\frac{s+m}{sm}\right)N-2\right)+ \\ &\quad \left(\left(\frac{s+m}{sm}\right)N-3\right)+.. \\ &\quad +\left(\left(\frac{s+m}{sm}\right)N-\mathcal{B}\right) \\ &= \frac{N}{s}+N\left(\frac{s+m}{sm}\right)(\mathcal{B}-1)- \\ &\quad (\mathcal{B}+2)(\mathcal{B}-1)/2 \\ &= N\frac{((s+m)(\mathcal{B}-1)+m)}{sm}- \\ &\quad (\mathcal{B}+2)(\mathcal{B}-1)/2 \end{aligned} \tag{3.12}$$

To compute the total number of rows (or memory units), we must determine the maximum number of $\mathcal{B}(N)$ blocks, or vertical shifts, required to successfully assign all packets to memory.

Lemma 4 *The maximum number of packets with the same departure time in the fourth block is* $P_4 \leq P_1\left(1-\frac{m}{R_1}\right)-mR_1+m^2$.

Proof. Suppose there are P_1 packets with the same departure time in the first block. Recall that there can be no more than $\frac{N}{s}$ packets with the same arrival time in the first block and no more than N packets with the same departure time in the system, such that $P_1 \leq N$. Packets only move vertically from the first block if a given packet resides in a row

that contains m other packets with the same departure time. Let us state that there are P_1 packets residing in R_1 $\left(R_1 \leq \frac{N}{s}\right)$ rows of the first block, then the number of packets that propagate vertically to the second block must equal the number of conflicting packets given by

$$P_2 = P_1 - mR_1.$$

Decisions regarding which row destination for a given packet are made independently such that packets with the same departure time can shift simultaneously to the same row. Note that a maximum of R_1 packets can shift simultaneously such that the resulting number of rows with conflicts in the second block is given by

$$R_2 \geq \left\lfloor \frac{P_1 - mR_1}{R_1} \right\rfloor \tag{3.13}$$

The value of R_2 represents the number of unique rows that received packets, with the same departure time, from the first block. Applying these same principles, we can further state the maximum number of packets with the same departure time that can shift to the third block block is given by

$$P_3 = P_2 - mR_2 \leq P_1 - mR_1 - m\left(\left\lfloor \frac{P_1 - mR_1}{R_1} \right\rfloor\right) \tag{3.14}$$

If $P_1 - mR_1$ is divisible by R_1, then

$$P_3 \leq P_1\left(1 - \frac{m}{R_1}\right) - mR_1 + m^2 \tag{3.15}$$

otherwise, since $P_4 \leq P_3 - 1$, we have

$$\begin{aligned} P_3 &\leq P_1\left(1 - \frac{m}{R_1}\right) - mR_1 + m^2 + 1 \\ P_4 &\leq P_1\left(1 - \frac{m}{R_1}\right) - mR_1 + m^2 \end{aligned} \tag{3.16}$$

■

The maximum value of 3.16 is obtained when $R_1 = \sqrt{P_1}$. Substituting $P_1 = N$, yields the following inequality

$$P_4 \leq \left(\sqrt{N} - m\right)^2 \tag{3.17}$$

Note that if N is a complete square we have,

$$P_3 \leq \left(\sqrt{N} - m\right)^2 \tag{3.18}$$

Corollary 2 *A sufficient number of parallel memory blocks required for an NxN switch, employing the proposed architecture, is* $O(\sqrt{N})$.

Proof. Equation 3.17 shows that for an N-port switch, the maximum number of conflicting packets with the same departure time in the fourth block is $\left(\sqrt{N} - 1\right)^2$. Let $\mathcal{B}(N)$ represent the number of stages required for an N-port switch. We can thus express the total number of stages required using the recursive relationship,

$$\begin{aligned} \mathcal{B}(1) &= 1 \\ &\vdots \\ \mathcal{B}(N) &= \mathcal{B}(N\text{-}2\sqrt{N}\text{+}1)\text{+}3 \end{aligned} \tag{3.19}$$

from which we conclude that $\mathcal{B}(N) = O(\sqrt{N})$. ■

Theorem 2 *For an N-port switch, where* $N \leq k^2$ $k \in \{1, 2, \ldots.\}$ *and* $s = m$*, the number of memories is*

$$L(N) \leq \frac{4k^3}{m^2} + (\frac{3}{m} - \frac{6}{m^2})k^2 - (\frac{5}{m} - \frac{4}{m^2})k - (2 - \frac{5}{m} + \frac{2}{m^2}) \tag{3.20}$$

with equality if $N = k^2$

Proof. We prove the equality for $N = k^2$, suggesting that the general case trivially follows. We first show by strong induction that the number of required row blocks are

$$\mathcal{B}(k^2) \leq \frac{2k}{m} + (2 - \frac{2}{m}) \tag{3.21}$$

For $k = 1$, the result is trivial. In order to prove it for $k \leq m$, it is sufficient to show $B(m^2) = 2$. To that end, for $k = m$, notice that number of rows in the first block is,

$$R(1) = N/s = k^2/s = m^2/m = m.$$

Therefore, maximum number of packets that can move simultaneously to the same row in the second block is m (one packet from each row in the first block). Since each memory can serve up to m packets with same departure time, all packets in the second block rows can be scheduled and there is no need to have third block rows.

So far, we have proved the result for $k = 1, \cdots, m$. Next we use, the strong induction step to prove it for $k > m$. We assume it is true for $1, \cdots, k$ $k \geq m$ and prove it for $k + 1$. For $N = (k + 1)^2$, using lemma 4 and (3.18) (given that N is a complete square), we have

$$\begin{aligned} \mathcal{B}((k+1)^2) &\leq \mathcal{B}((k+1-m)^2) + 2 \\ &\leq \frac{2(k+1-m)}{m} + 2 - \frac{2}{m} + 2 \\ &= \frac{2(k+1)}{m} + (2 - \frac{2}{m}) \end{aligned} \tag{3.22}$$

Now, we substitute (3.22) and $s = m$ in (3.12) to obtain,

$$\begin{aligned} L(k^2) &= k^2 \frac{2(\mathcal{B}-1)}{m} - \frac{(\mathcal{B}+2)(\mathcal{B}-1)}{2} \\ &\leq k^2 \frac{2(2k/m - 2/m + 1) + 1}{m} - \\ &\frac{(\frac{2k}{m} + 4 - \frac{2}{m})(\frac{2k}{m} + 1 - \frac{2}{m})}{2} \\ &= \frac{4k^3}{m^2} + (\frac{3}{m} - \frac{4}{m^2})k^2 - \frac{2k^2}{m^2} - \\ &(5 - \frac{4}{m})\frac{k}{m} - (2 - \frac{5}{m} + \frac{2}{m^2}) \\ &= \frac{4k^3}{m^2} + (\frac{3}{m} - \frac{6}{m^2})k^2 - (\frac{5}{m} - \frac{4}{m^2})k - (2 - \frac{5}{m} + \frac{2}{m^2}) \end{aligned} \tag{3.23}$$

■

In comparison to (3.11), the above theorem states that for a speedup of 4, with $s = m$, a 16-port switch fabric can be realized with 58 memory units, as opposed to the 177 parallel memory units required without the inclusion of speedup. Further, the number of required memory units for a switch fabric with $N = 64$ ports is a quite practical 144 memory units. It is apparent that the packet placement algorithm yields a substantial reduction in the overall memory requirements, thus paving the way for the implementation of large scale FoC based platforms.

3.3 Column-Associative Packet Placement Algorithm

In this section, we present an alternative memory management algorithm that employs a column-associative memory management algorithm. The row-associative memory management algorithm, presented in the previous section, determined placements by allowing each cell located on the diagonal to determine whether it could be placed in the associated row memory. If the row memory was occupied, the cell would then select a row, from a set of $2N-1$ potential candidates, in which to place the conflicted cell. Essentially, the placement algorithm allowed the cell move vertically until an available memory was found. With the column-associative memory management algorithm, we no longer limit placement decision considerations to one cell. Instead, we employ a pipeline memory management algorithm, whereby all cells arriving at time t are coupled with a single memory each time slot. In doing so, we allow multiple cells to be considered for placement at once while reducing placement contention. This provides a significant reduction in the memory requirement, while simultaneously reducing packet delay. In subsequent sections, we will describe the proposed architecture in detail then, in Chapter 4, enhacements that provide multicast support, QoS guarantees, and load balancing capabilities are presented.

3.3.1 Column-Associative Packet Placement Algorithm

As we stated in our discussion of the row-associative packet placement architecture, packets arrive at each ingress port where they are segmented into fixed size cells for efficient placement in memory. Prior to the presentation of a cell to the switch fabric, the departure time

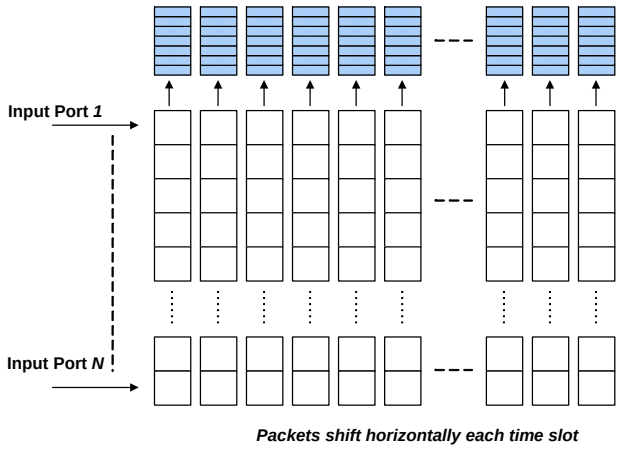

Figure 3-4: Packets are selected for placement into each column memory based on their departure time.

must once more be determined in order to establish the appropriate departure sequence for a given set of arriving cells. We initially consider a first-come-first-serve (FCFS) scheduler whereby packets are assigned departure times in accordance with their arrival order. In later sections, we will provide support for more sophisticated schedulers whose departure time assignments are intended to provide delay and rate (QoS) guarantees.

The proposed memory management algorithm, illustrated in Figure 3-4, consists of $M \times N$ cell buffering units arranged in a rectangular structure, whereby cells enter at the leftmost column and are guaranteed placement upon reaching the final column. Each column in this structure is associated with a single parallel memory unit resulting in a total of M parallel memories. Packets arriving at ingress port i are segmented into k byte cells prior to insertion into row i. Consequently, cells located in the same column will contain a matching arrival time when this scheme is utilized. At the end of each cell time, all cells located in column i will be transferred to column $i + 1$, providing every cell with the opportunity to be placed in each of the M available parallel memories.

The cell placement unit, located in each column, determines whether a cell, located in its column, can be placed in the corresponding column memory. A cell will be placed in the associated column memory if the following conditions are met: (1) the memory associated with the column does not already contain another cell with an identical departure time; (2)

the cell is selected for placement over the potential $N-1$ other cells with the same arrival time. As a cell progresses through each subsequent stage, memory placement contentions are reduced. In this scheme, once a cell is shifted to the last column of the pipeline, it is guaranteed to be placed in a memory that does not contain a cell with a matching departure time.

In the proposed architecture, rows can be viewed as simple shift registers, whereby cells are shifted one stage to the right at each time step. At each stage of the pipeline, a single cell assignment is attempted per column. The motivation for this approach is to reduce the complexity of the placement algorithm by isolating memory assignments, thus minimizing memory contention.

3.3.2 Basic Analysis of Resource Requirements

Theorem 3 *A total of* $2N-1$ *column dual-port memories is sufficient for an* N*-port switch utilizing the proposed packet placement algorithm.*

Proof. Consider an arriving cell, C_1. It will find at most $N-1$ other cells with an identical arrival time competing for the same memory. Furthermore, at most $N-1$ other cells with the same departure time could have been placed in unique memories prior to the arrival of C_1. Thus, at least $(N-1)+(N-1)+1 = 2N-1$ memories are required to guarantee the placement of cell C_1. ■

The cell placement unit, located in each column, maintains a mapping of memory locations to corresponding to departure times, that have been reserved by cells successfully placed in the memory. This mapping, which is effectively a binary mask, shall be referred to as the column's *occupation vector*. In addition, each column maintains a mapping that specifies pre-allocated (or reserved) departure times of cells that have yet to be placed in a column memory. This mapping is referred to as the column's *vacancy vector*. The placement element in the pipeline performs a bitwise xor operation over the occupation and vacancy vectors to determine if there is a cell with a departure time available for placement in the memory. If a cell is selected for placement, it will be extracted from the pipeline and written the corresponding column memory. Cells not selected for placement are subsequently forwarded to the next stage of the pipeline.

Example 3 *Consider the simple scenario depicted in Figure 3-5 illustrating the state of the pipeline for three consecutive time slots (i.e. t, t+1, t+2). At time t, three cells are inserted into the first column of the pipeline with departure times* $\{1, 2, 1\}$ *respectively. Assuming that all memories are initially empty, the placement engine will simply select the first cell in the column for placement. Thus, the cell in row* 1 *with departure time* 1 *is placed in the memory associated with column* 1*. The remaining cells* $\{2, 1\}$ *will shift right to column 2 at time* $t + 2$*, while new cells, with departure times* $\{1, 3, 2\}$*, are inserted into the first column. The placement element at column 2 will now select the cell with departure time 2, leaving the only remaining cell,* $\{1\}$*, to shift to column 3. In column 1, a cell with departure time 1 has already been placed in the memory associated with column 1. Thus, the placement element in column one must select the cell at row 2, with departure time 3, for placement. At time* $t + 2$*, additional cells, with departure times* $\{2, 4, 3\}$ *are introduced into the pipeline at column 1. As the cell with departure time 2 has yet to be selected, it is next selected for placement. Additionally, columns 2 and 3 both select cells with departure time of 1.*

It can be observed from this example, that the column-based memory management algorithm clearly provides an effective mechanism for resolution of memory contention.

Scalability is achieved as decisions are disassociated from the number of ports in the system. Increased port densities do not directly infer an increase in the time required to make a placement decision. Instead, placement decisions are determined only by the number of departure times offered by the switch fabric in the form of the occupation and vacancy vectors. In the context of FoC, these mapping vectors are bound only by the depth of an individual column memory.

The coupling of placement decisions with memory depth is not unbounded. In all practical switching systems, once a buffer approaches (or is close to approaching) its limit, flow-control signaling should be provided to the traffic sources, indicating the need to either slow down or temporarily stop the flow of packets to a particular destination. Such a mechanism is always required since instantaneous data congestion may occur at any router or switch. In fact, even if the traffic is said to be statistically admissible, implying that no input or output is oversubscribed, it may still be the case that for short periods of time a given output port is oversubscribed. To address such scenarios, and in an effort to reduce

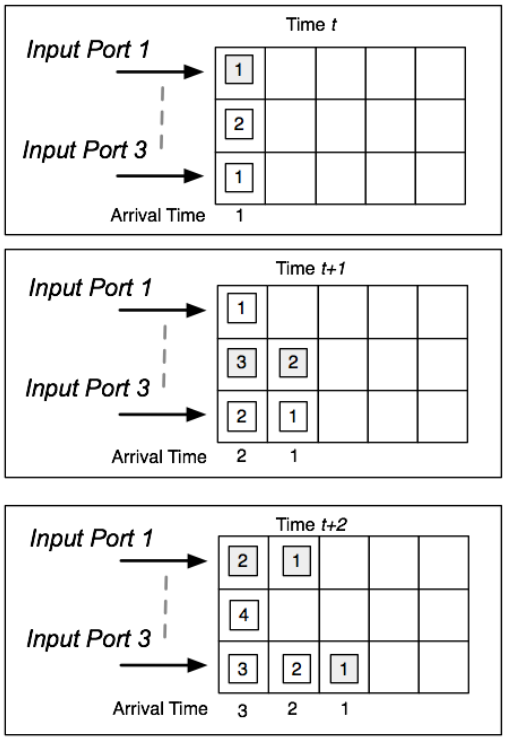

Figure 3-5: Example illustrating the proposed memory management algorithm for a 3-port switch. The state of the pipeline structure is depicted for 3 consecutive time slots.

the probability of packet loss, the linecards typically host large memory spaces. As such, each parallel memory unit contained in the FoC is relatively shallow with its depth governed by the system response to backpressure and the number of memory units employed.

3.3.3 Incorporating Speedup

To achieve a further reduction in the number of required memories, it is clear that we must limit memory contention in the packet placement process. One source of contention is a blocked placement that occurs as a result of offering only one memory location per departure time for each memory unit. This constraint can be alleviated by provisioning multiple cell placements for a single column memory, allowing a single memory to host $1 < m \leq N$ cells with identical departure times. As a result, a cell C, with departure time d, will consider a memory unavailable for placement only if it contains m cells, having arrived prior to C, that have a matching departure time.

Theorem 4 $\left(\frac{m+1}{m}\right) N - 1$ *memories are guaranteed to be sufficient for an* $N-$*port switch, where each memory can hold* m *cells with the same departure time.*

Proof. Consider the arrival of cell C_1 to the first column of the switch. It will find at most $N-1$ other packets with the same arrival time competing for the same memory. Moreover, there are at most $N-1$ other packets with the same departure time that may have been placed in a memory prior to the arrival of C_1. As cells are presented to memories sequentially, each offering multiple departure times, the $N-1$ cells with matching departure times must be contained in one of $\frac{N}{m}-1$ memories. Therefore, a total of $\left(\frac{N}{m}-1\right)+(N-1)+1 = \left(\frac{m+1}{m}\right) N - 1$ memories are required to ensure placement of C_1. ■

It should be noted that provisioning multiple cell placements for each column memory does not impact the time required to establish a memory's availability. The primary trade-off in this instance resides in the increased density of each memory unit and the requirement that packets must be read from memory at a rate m prior to being forwarded to the appropriate egress ports. Having made this observation, we introduce a placement (or computation) speedup of s, which assumes the pipeline operates at a rate s times faster than the line rate. One advantage of operating the pipeline at an increased rate is reduced

latency. Moreover, cell arrivals from a set of N inputs can be presented to the switch fabric in groups consisting of $\frac{N}{s}$ cells. This further provides a reduction in the number of conflicts associated with packets possessing the same arrival time from N to $\frac{N}{s}$.

Theorem 5 *A total of* $\left(\frac{s+1}{s}\right) N - 1$ *memories is sufficient for an* N*-port switch with a placement speedup of* s*.*

Proof. For a placement speedup of s, an arriving cell, C_1, will find at most $\frac{N}{s} - 1$ other packets with the same arrival time competing for the same memory. Furthermore, at most $N - 1$ other packets with the same departure time may have been placed in unique memories prior to the arrival of C_1. Therefore, at total of $\left(\frac{s+1}{s}\right) N - 1$ memories are required to ensure placement of cell C_1. ■

Given that columns are interlocked with memories, the incorporation of placement speedup infers a memory write speedup equal to s. However, offering multiple cell placements does not necessarily require an increase in memory write speed. Furthermore, the multiple cell placements does not require a pipeline speedup either. Thus, we can utilize both multiple cell placements and placement speedup concurrently to gain an even greater reduction in memory requirements. In view of the above assertions, it should be evident that the sufficient number of memories in this case is $k = \left(\frac{s+m}{sm}\right) N - 1$. Moreover, we achieve a reduction in the size of the cell buffering structure such that $k \times \frac{N}{s}$ cell buffering units are now required.

Chapter 4

Enhancements to the Switch Fabric on a Chip Architecture

This chapter extends the column-associative memory management algorithm, presented in section 3.3, to provide support for additional features such as quality of service (QoS) guarantees, load balancing and multicast traffic. In discussing multicast traffic first, we recognize that multicasting provides an effective mechanism for conserving network bandwidth by enabling the efficient transmission of a single stream of data to multiple destinations. This has become increasingly important with the growing demand for multimedia applications and has lead to considerable attention in the area of packet scheduling algorithms, as indicated by [37][38][39][40]. In concert with this increased demand for multicast traffic, there exists a concurrent need for service guarantees to support multimedia applications such as VoIP, IPTV, and video on demand (VOD). The basic framework for these multimedia applications require that content be sent to multiple recipients simultaneously while meeting stringent performance requirements (low latency, minimal jitter, fairness, etc.). For continued proliferation of these applications, the underlying infrastructure must provide efficient mechanisms for storage and forwarding of packets while offering quality of service (QoS) guarantees. Indubitably, this presents a significant challenge as we attempt to optimize performance and resource requirements, by incorporating features such as load balancing, while increasing the complexity in supporting multicast services and quality of service guar-

antees. In this chapter, we present architectural enhancements that enable the inclusion of these features. In the subsequent sections, we provide a detailed discussion of the proposed feature enhancements, while also outlining metrics that establish their viability.

4.1 Memory Provisioning for Multicast Traffic

The memory requirement for the proposed architecture has, to this point, been defined considering unicast traffic only. Given that each unicast cell can be destined for at most one egress port, the analysis is considerably easier since there can be at most one departure time assignment associated with each cell. With multicast traffic, each cell can be destined to multiple egress ports. The exact number of copies is expressed in terms of the maximum fanout q, $q \in \{1, .., N\}$, which denotes the number of egress ports for which a cell, C, can be destined. Consequently, a total of q departure time assignments must be made for C.

One straightforward approach for processing multicast cells, known as *copy multicast* [41], is to simply replicate the cell at the ingress, thus creating q unicast copies which are each delivered to the fabric and switched individually. Copy multicast has obvious drawbacks in that it requires a fabric speedup of q at the ingress, while also potentially requiring q additional memory locations to store the copied cells in a column memory. This is derived from the fact that each ingress port can receive a multicast packet destined for up to q egress ports.

To more efficiently store cells, an alternate approach, termed *fanout multicasting*, can be employed. In fanout multicasting, a multicast cell is transmitted to the switch fabric only once and avoids replication. Instead, only one copy of the cells is stored in memory and is read by the fabric multiple times for delivery to each of the q destinations. From an effective memory utilization perspective, this is an optimal solution. However, achieving this goal is a significant challenge given the difficulty of the placement process.

Lemma 5 $(q+1)(N-1)+1$ *memories is sufficient for a PSM switch, utilizing the column-based packed placement algorithm, with fanout multicasting.*

Consider fanout multicasting in a PSM switch, we observe an arriving multicast cell, C_q, will contain α, $\alpha \leq q$, unique departure times, which suggests there will also be α

potential departure time conflicts. For the column-based placement process using fanout multicasting, we recognize that a departure time conflict with any of the α unique departure times results in a placement failure. For each of the α unique departure times associated with the arriving cell, there are at most $(N-1)$ other cells with the same departure time that may have been placed in unique memories prior to the arrival of C_q. Ignoring arrival time constraints for the moment, clearly the placement of the multicast cell C_q requires $q\,(N-1)+1$ memories to resolve departure time contention when fanout multicasting is employed.

A linear reduction in this memory requirement can be achieved with the inclusion of memory speedup, m. If we allow multiple cells, having the same departure time, to be written to the same memory, we reduce the departure time constraint such that

$$k = \left(\left\lceil \frac{q}{m} \right\rceil + 1 \right) (N-1) + 1 \tag{4.1}$$

memories are now sufficient to ensure placement of a multicast cell into memory. However, this approach has limited scalability given the speedup needed to minimize the number of required memories cannot be easily attained.

The formulation of an optimal policy for placing cells into shared memories should incorporate the beneficial aspects of both copy and fanout multicasting. For copy multicasting, we seek to minimize the number of replications required to place a packet without increasing the number of column memories required. To accomplish this goal, we utilize the positive aspects of fanout multicasting by bounding the number of replications that can occur. This can be achieved by allowing the association of a finite number of departure times, b, to occur for some number of replicated cells. The approach restricts the maximum number of duplicate packets, or replications, to be $\left\lceil \frac{q}{b} \right\rceil$. From Lemma 5, we recognize the memory sufficiency condition can now be represented by

$$k = (b+1)\,(N-1) + 1. \tag{4.2}$$

Further optimization of the multicast placement algorithm can be attained with the introduction of both memory and computation speedup.

Offering a memory speedup, m, provides a mechanism for reducing the departure time constraint associated with allowing multiple departure times to be contained in one cell. Additionally, the incorporation of computation speedup, s, enables cells to be introduced in groups of $\frac{N}{s}$ which reduces the arrival time constraint. As a result, the introduction of both memory and placement speedup yields a simultaneous reduction in both the arrival and departure time constraints.

Theorem 6 $\left\lceil \frac{sb+m}{sm} \right\rceil N - b$ *memories are sufficient for an* N*-port multicast PSM switch utilizing the proposed packet placement algorithm.*

Proof. For the N-port multicast switch with computation speedup, there are at most b departure times associated with a given multicast cell, C_m. Additionally, there can be at most m cells located in a given column memory with departure times that match those contained in C_m. As placement opportunities occur sequentially, C_m can be denied placement into at most $b\left(\frac{N}{m}-1\right)$ memories due to a departure time constraint violation. Furthermore, an arriving multicast cell, C_m, will find at most $\frac{N}{s}-1$ other cells with the same arrival time. Therefore, a total of $b\left(\frac{N}{m}-1\right)+\left(\frac{N}{s}-1\right)+1 = \left\lceil \frac{sb+m}{sm} \right\rceil N - b$ memories are required to ensure placement of cell C_m. ■

Bounding the number of replications that can occur for each multicast cell facilitates an optimal trade-off between placement complexity, required memories, and memory depth. For fewer cell replications, the replication bound, b, can be increased which results in reduction of the required memory depth. While this yields fewer duplicate cells, the number of unique memory instances will increase given the added departure time conflicts. This can be minimized if we tightly couple the replication bound with computation and memory speedup. For $s = m$, the memory sufficiency condition becomes $\left\lceil \frac{b+1}{s} \right\rceil N - b$. In doing so, we are able to limit the number of unique memory instances by only increasing the replication bound, b, in parallel with the speedup, s. Table 4.1 outlines the number of memories needed for various port densities, and m and s values.

Table 4.1: Number of memories required for the proposed PSM multicast switch employing the column-associative memory management algorithm

Switch Ports (N)	Speedup (s)	Replication Bound (b)	Maximum Replica-tions	Memory Units
16	1	1	16	31
16	2	2	8	30
16	4	4	4	28
32	1	1	32	63
32	2	2	16	62
32	4	4	8	60
64	1	1	64	127
64	2	2	32	126
64	4	4	16	124

4.2 Providing QoS Guarantees

In a network of output-queued or shared memory switches, it has been demonstrated that a scheduler capable of providing weighted fairness among flows, or delay guarantees, at all queueing points in the network can yield end-to-end guarantees for well-behaved flows [42]. The FCFS scheduler considered thus far is incapable of providing such fairness, or delay guarantees, as the scheme is biased toward flows that transmit at high rates.

Implicit in our goal of controlling delay is an underlying need to modify the relative departure times of packets residing in memory. This can be accomplished if we allow the departure time assignment policy to follow any *push-in first-out* (PIFO) [43] queueing discipline (WFQ, GPS, strict priority, etc.). In a PIFO queueing discipline, the relative transmission order of cells remains fixed once placed in the queue. Arriving cells may be inserted into the queue at any point, but always depart from the head of line. The departure time assigned to an arriving cell is performed prior to insertion. The scheduling policy can dictate cells within a packet be given sequential departure times, thus enabling performance guarantees for both packets and cells.

Consider the single egress queue employing a PIFO queueing discipline, containing cells destined for output a only. In this example, we denote the cell destined for output a, with departure time 2, as a_2, departure time 3 as a_3, and so forth. Initially, three cells are destined to leave output a in order, as depicted in Figure 4-1(a). Now consider the insertion

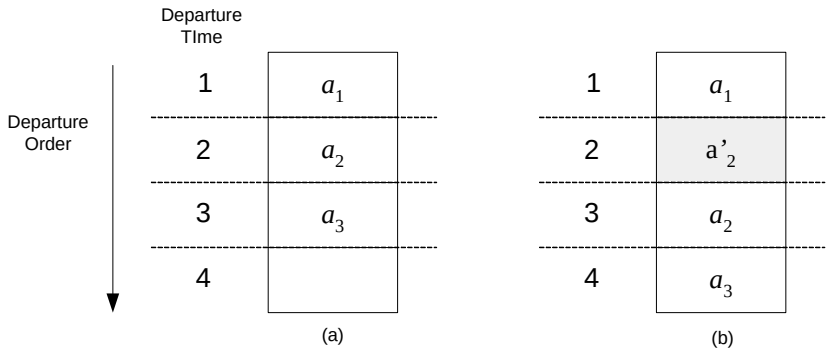

Figure 4-1: (a) Initial ordering of PIFO queue prior to insertion of cell $\left(a_2^{'}\right)$ (b) Ordering of PIFO queue after cell $\left(a_2^{'}\right)$ has been 'pushed-in'

cell of $a_2^{'}$ into the PIFO queue. In this instance, $a_2^{'}$ is scheduled to depart prior to cell a_2. Thus, it will be 'pushed in' just after the cell a_1. The insertion of $a_2^{'}$ will obviously delay the departure of cells a_2 and a_3, as depicted in Figure 4-1(b), by one cell time. This is the desired outcome, as all cells will depart in the same relative order.

The introduction of PIFO queues does, however, introduce contention not evident in the preceding example. In that instance, all cells were destined for a single output a and stored in the same PIFO queue. This is convenient for the purpose of highlighting the operation of a PIFO queue. However, this neglects potential conflict associated with cells destined for an output other than a in a system consisting of k PIFO queues. Allowing cells to be placed into memories regardless of their departure times introduces a conflict, whereby a cell destined for output a could be pushed into a PIFO queue such that it modifies the relative order of cells destined for the other $N-1$ egress ports.

To highlight the difficulty in managing k PIFO queues, we consider the insertion of multiple cells, into a subset of k memories, to demonstrate conflicts that can can occur. In Figure 4-2(a), we have three PIFO queues containing packets destined to ports a, b, and c. The cells $\{a_1, b_1, c_1\}$ will depart together in the first time slot, then followed by cells $\{a_2, b_2, c_2\}$ in the next time slot, and so forth. First, we consider the insertion of cell $a_2^{'}$, depicted in Figure 4-2(b), into the first PIFO queue, just prior to cell a_2. As observed in the previous example, the relative order of departures of cells destined to port a remains

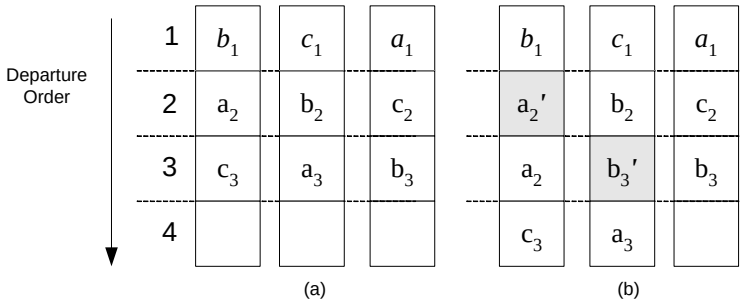

Figure 4-2: (a) Initial ordering of PIFO queue prior to insertion of cell $a_2^{'}$ (b) Ordering of PIFO queue after cell $a_2^{'}$ has been 'pushed-in'

unchanged. However, the relative order of cells destined for port c does change, as c_3 will now depart one time slot later than its scheduled departure time. Further, consider the insertion of $b_3^{'}$ into the second PIFO queue. We again face a similar problem given cell a_3 will also depart one time slot after its scheduled departure time. Further, cells destined for the same output, $b_3^{'}$ and b_3, will now depart at the same time from separate memories. As multiple cells can be delivered simultaneously to the same egress port, an output buffer, operating at $O\left(NR\right)$, is now required to eliminate output contention at each egress port. This approach is impractical as it reintroduces the high bandwidth requirement found in OQ switches, while also eliminating their desirable flow isolation attributes.

To resolve the memory contention present when multiple PIFO queues are employed, we first seek to avoid conflicts for cells destined to different output ports. To do so, we utilize the conflict-free permutation approach introduced in [5], whereby the relative departure order is modified to avoid read conflicts. With this approach, we adjust the read process such that each output, in a router consisting of N outputs and k shared memories, is allowed to read at most one cell from each of the k memories. If we consider a set of k memories, whereby each memory is denoted by m_i , $i \in k$, then cells destined for output a would be read from memories using the read sequence $\{m_1, m_2, m_3, ..., m_k\}$. As N cells will need to depart every time slot to obtain maximum throughput, we define a read window, w, that consists of N cells (one cell destined for each egress port), such that N memories will be simultaneously read. With this approach, the read sequence for additional outputs will

need to be shifted in order to perform simultaneous reads for each output. Consider cells destined for output b, the read cycle must start from m_2 such that its read sequence does not conflict with sequence used by output a. Thus, for output b, the read sequence would be $\{m_2, m_3, .., m_k, m_1\}$. Assuming there is a cell in every memory, destined for each output, then the application of this approach will achieve maximum throughput given each output will receive k cells in an interval consisting of k cell times.

As we have elected to use an alternate read process, we have introduced an additional source of contention. This contention is derived from the fact that at most one cell can be read, during a single read cycle consisting of k cell times, from each of the k memories for the individual outputs. For instance, if β cells, $2 < \beta \leq k$, are destined for the same output and happen to be placed in the same memory, then only one cell will depart in the scheduled interval of k departure times. This results in the delay of $\beta - 1$ cells, such that these cells will no longer depart during their assigned slot. This implies additional placement constraints are needed to ensure each of the k cells, scheduled to depart in a given read cycle, are placed in unique memories.

If we reconsider the scenario described above employing the alternate read process, we must first define a read cycle, ς, to consist of 3 cell times. Prior to the insertion of $a_2^{'}$, ς consists of the cells $\{a_1, a_2, a_3\}$. As such, each cell must be located in separate memories to depart in the correct order. After the insertion of $a_2^{'}$, ς now contains the cells $\left\{a_1, a_2^{'}, a_2\right\}$ which all must be placed in unique memories. Further, if we considered the insertion of a cell $a_1^{'}$, clearly ς would then contain the cells $\left\{a_1^{'}, a_1, a_2^{'}\right\}$, where the three cells are yet again required to be located in unique memories. At this point, we have inserted two cells and now require five unique memories to store each of the cells. If we allow incoming cells to be "pushed in" to memories without restriction, the number of unique memory instances required will clearly increase without bounds.

In outlining the placement constraints, we recall the read process is structured such that k cells are delivered to each of the N outputs in a read cycle consisting of k cell times. The exact number of shared memories, k, necessary to support this read operation must be $k \geq N$, since each output must read at least one packet from each memory without contention. For $k = N$, an arriving cell C_1 must not be "pushed in" to one of the $N - 1$

memories containing cells destined to depart in the $N-1$ time slots prior to C_1. This is apparent given that each of the N cells scheduled to depart in the k cell times, all destined for the same output, must be located in separate memories. While this ensures C_1 will be depart in the correct interval of k cell times, its insertion can still produce conflicts with cells scheduled to depart after C_1. This is obvious as the relative departure order of cells following C_1 is altered resulting in the scenario presented above. To resolve this issue, we restrict C_1 from being placed in the $N-1$ memories containing cells scheduled to depart just after it, that are also destined for the same output port. These two constraints can then be combined to state an arriving cell, C_1, should not be placed into the $N-1$ memories containing cells, destined for the same output port, that are scheduled to depart either before, or after, C_1. From these constraints, we observe that $k = 2\,(N-1)+1 = 2N-1$ memories is sufficient to ensure no placement conflicts will occur.

The result we have just presented, however, neglects the arrival time conflicts that are a result of the column-associative packet placement algorithm. If we consider an arriving cell can be blocked by at most $N-1$ cells having the same arrival time, we recognize the memory sufficiency condition increases such that, $k = 2\,(N-1)+(N-1)+1 = 3N-2$ memories are now sufficient to ensure placement occurs without conflict. Further reduction of the memory requirement can be achieved by incorporating speedup as discussed in Section 3.3.3. Since PIFO queues are employed, we do not need to offer multiple departure times for each memory, instead we utilize both memory write- and read-speedup, s, to provide a reduction in resource requirements. Cells will again be introduced to the pipeline in groups of $\frac{N}{s}$ and written to memory with speedup s. Additionally, we will now read cells from memory in groups of $\frac{N}{s}$ to achieve a simultaneous reduction in pipeline depth and memory requirements. Incorporating the speedup gains, we observe the memory requirement decreases such that, $k = 2\left(\frac{N}{s}-1\right)+\left(\frac{N}{s}-1\right)+1 = 3\left(\frac{N}{s}\right)-1$ is now sufficient.

In reading a complete window of cells, the order in which cells arrive at a given output is not guaranteed. The read process only ensures that all k cells scheduled to depart, in a period of k cells times, will be delivered to each output in a given interval. Since the order is not guaranteed, we recognize a small reordering buffer containing at least $k-1$ cells is required to hold cells associated with a given read cycle. To determine the total

delay observed by an arriving cell, we must include the delay of the pipeline, k cell times, and the delay associated with the reordering buffer, $k-1$ cell times. Aggregating the delay associated with both the placement process and the reorder buffer, we note that a parallel shared memory router, utilizing the column-associative packet placement algorithm, can emulate an output queued router within $2k-1$ time slots

4.3 Load Balancing

Our discussion thus far has assumed that cells are inserted into the memory management pipeline at the leftmost column and are shifted right at each time slot. As cells propagate through the pipeline, column placement units select a single cell for storage in the associated column memory. The progression of cells through each subsequent stage of the pipeline results in placements that tend to be concentrated near columns closest to their insertion point. This is intuitive considering that for every cell in a given pipeline column, there must be a corresponding cell in that column's memory with a matching departure time. As a group of arriving cells, always located in the same column, advances in the pipeline, the likelihood that a conflict exists for all cells in the column decreases significantly. In addition, we note that after $N-1$ successive occurrences where no packets are placed, at least one packet is guaranteed to be placed in the successive N columns.

To illustrate the clustering of cell placements around the insertion point, we present simulation results, shown in Figure 4-3, for a 64-port switch with 127 memories. In highlighting the clustering effect, we employ a FIFO queueing scheme. Since the clustering of cells is the result of a horizontal progression of cell placements (not the queuing discipline), the results are equally relevant for PIFO queueing schemes. In the simulation results presented, cell arrivals are uniformly distributed across the destinations and obey a Bernoulli i.i.d. process with a normalized traffic load of one. As evident from the queue distribution presented, cell placements are densely clustered around the insertion point and drop off considerably after N placement opportunities.

From an implementation perspective, we would prefer the distribution of cells across the shared pool of memories be uniformly distributed. This enables dynamic queue management

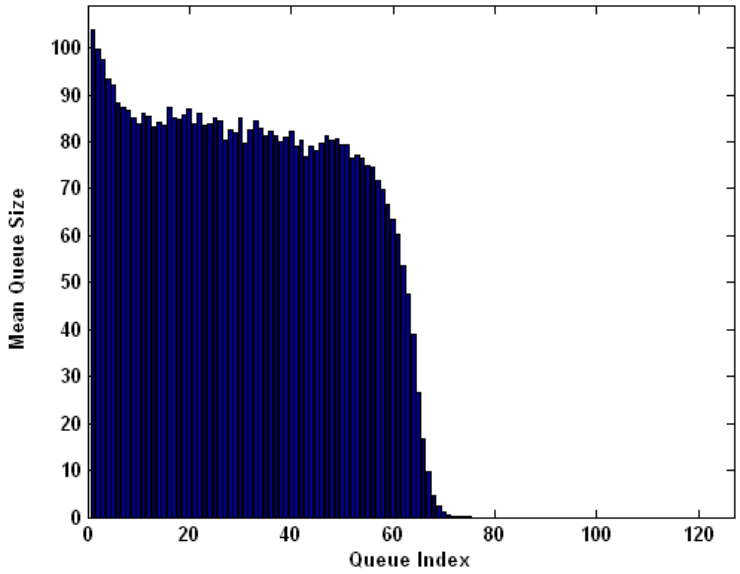

Figure 4-3: Queue size distribution for a PSM system without load balancing.

algorithms and thresholds to be more efficiently implemented. In order to more effectively distribute cell placements, the elements at which cells are inserted into the pipeline must vary for each port to avoid clustering. To accomplish this goal, a rotation technique, presented in Figure 4-4, is employed, whereby cells arriving at port i are inserted into the column that precedes the insertion point used in the previous time slot. For example, if at time t a cell was inserted into column φ, then all cells inserted at time $t+1$ must be placed in column $\varphi - 1$. If a cell is inserted into the rightmost column and has yet to be placed in memory, it is shifted to the leftmost column, representing a circular shift, to avoid cell loss. As cells advance in a clockwise manner and insertion points occur counter-clockwise, then we must ensure that a cell will not meet an insertion point within $2N - 1$ time slots to avoid cell loss.

Theorem 7 *Uniform distribution of cells across the memories, employing a FIFO queuing discipline, can be achieved with* $4N - 2$ *memory units.*

Proof. For a cell C_1 inserted at column φ_1, there must be $2N - 1$ columns prior to the insertion of another cell. Thus, an additional $2N - 1$ columns are required to rotate the

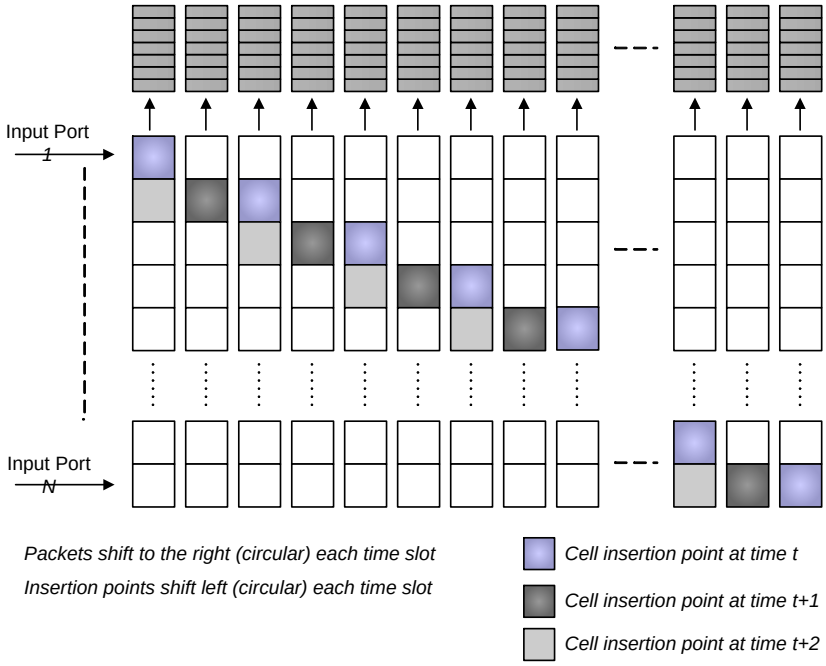

Figure 4-4: Rotation insertion scheme to support load balancing.

insertion points to ensure uniform distribution across the memories. Aggregating the two terms, we conclude $4N - 2$ memories are sufficient to achieve a balanced cell distribution.

■

In Figure 4-5, a 32-port switch with 126 memories is simulated assuming maximal load. It is observed that a rather uniform memory occupancy distribution results, with fewer cells stored in each memory.

4.4 Comparison of Switching Architectures

In evaluating switch architectures, considerations are most often given to performance, scalability, and technical viability. Packet delay and throughput are critical components in establishing switching fabric performance, whereas scalability and technical viability are considered in terms of the resource utilization for increasing port densities. This section will outline, with respect to the stated metrics, the benefits of the row-associative and column-associative packet placement algorithms as compared to existing switching architectures

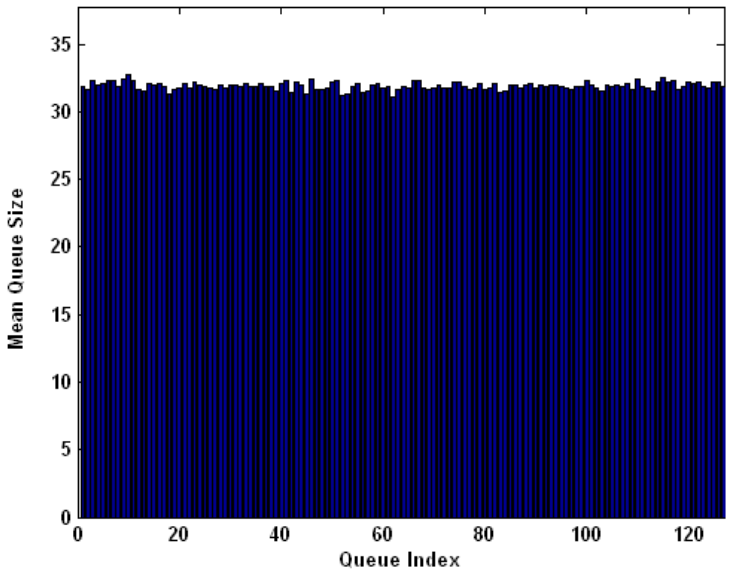

Figure 4-5: Queue size distribution with load balancing.

(IQ, OQ, PPS, etc.). Our goal is to highlight the scalability of the proposed architectures with regards to the memory bandwidth requirement, as well as discuss practical implementation considerations.

Several alternative architectures were presented in Section 1.3 to provide context for the PSM architectures outlined in this dissertation. In Table 4.2, we provide a summary of the bandwidth and memory resource requirements associated with each architcture. Perhaps the most straightforward architecture is the shared memory switch, whereby all incoming packets are placed into a single, shared buffer during each time slot. Additionally, all outgoing packets must be read from the same shared buffer in one time slot resulting in a memory bandwidth requirement of $2NR$. Output queued (OQ) switches were proposed to reduce this memory requirement by buffering incoming packets at their destination. For an OQ switch having N ports, there can be N packets destined for a single output queue. This results in each queue having a memory bandwidth requirement of $(N+1)\,R$. While the shared memory and OQ switch architectures provide 100% throughput and offer ideal delay properties, both impractical for increasing line rates and port densities given their

Table 4.2: Comparison of various routing architectures.

Type	**Total Memories**	**Memory Bandwidth Required**	**Total BW Required**	**Comments**
Shared Memory	N	$2NR$	$2NR$	Offers ideal delay and throughput metrics.
Output Queued	N	$(N+1)R$	$N(N+1)R$	Offers ideal delay and throughput metrics.
Input Queued	N	$2R$	$2NR$	Maximum throughput 58% without VOQs. Achieves 100% throughput with MWM.
CIOQ	$2N$	$2R$	$2NR$	Emulates FCFS OQ switch with speedup of 2.
Load Balanced Router	N	$2R$	$2NR$	Mis-sequencing present. Offers 100% throughput
	$2N$	$2R$	$2NR$	Utilizes coordination buffer to emulate OQ switch and provide QoS guarantees.
Distributed Shared Memory	N	$4R$	$4NR$	Emulates FCFS OQ switch.
	N	$4R$	$4NR$	Provides QoS guarantees.
Row-associative PSM	$O\left(N^{1.5}\right)$	$2sR$	$2sNR$	Emulates FCFS OQ switch.
Column-associative PSM	$\left(\frac{s+m}{sm}\right)N-1$	$2sR$	$2sNR$	Emulates FCFS OQ switch
	$3\left(\frac{N}{s}\right)-1$	$2sR$	$2sNR$	Provides QoS guarantees with delay of $2k-1$ time slots
	$\left\lceil\frac{b+1}{s}\right\rceil N-b$	$2sR$	$2sNR$	Supports multicast traffic with bound, b, on the number of copies.
	$4N-2$	$2sNR$	$2sNR$	Provides load balancing

respective memory bandwidth requirement.

In comparison to the shared memory and OQ switch architectures, both the row-associative and column-associative packet placement approaches offer similar throughput and delay properties. The shared memory and OQ switch architectures do offer better performance when QoS guarantees are offered. Recalling our discussion of the column-associative packet placement architecture, in order to support QoS guarantees a reordering buffer was required, which results in a delay of $2k - 1$ cell times. However, both the row-associative and column-associative architectures only require a memory bandwidth requirement of $2sR$, which is considerably more practical as line rates and port densities increase. Moreover, the multicast approach utilized by the column-associative PSM architecture provides considerably better memory utilization compared to the OQ architecture. For OQ architectures, each multicast cell must be replicated and stored at each egress destination upon arrival. This results in a sizable increase in the memory requirement relative to the proposed PSM architectures.

Input queued (IQ) routers were introduced to minimize the high bandwidth of OQ switches by buffering arriving cells at their inputs. As indicated in Chapter 1.3, an IQ switch that employs a FIFO queueing discipline has a maximum throughput of 58% due to Head-of-Line (HOL) blocking. In the HOL blocking scenario, a blocked packet at the head of an input queue prevents all packets that follow it from traversing the switch, even when the desired output link is idle. Aside from the HOL blocking issue, the IQ switch does possess a pragmatic memory bandwidth requirement of $2R$. To eliminate the HOL blocking issue, an alternate approach, termed virtual output queueing, was proposed whereby each input maintains a separate queue for each output. This approach allows the switch to achieve 100% throughput using a maximal weight matching (MWM) algorithm. However, the MWM algorithm is not practical from an implementation perspective, as it has a computational complexity of $O\left(N^{2.5}\right)$.

A hybrid approach that utilizes both input and output queueing, referred to as combined input output queueing (CIOQ), can emulate the delay and throughput properties found in the shared memory and OQ switches with a speedup of two. While CIOQ does address throughput and delay issues associated with IQ switches, the complexity of the

scheduling algorithms remain a problem as they do not scale well. The advantage of employing the proposed PSM architectures is the scalability of the placement process. For the column-associative packet placement algorithm, the placement process scales linearly, while maintaining the same memory bandwidth requirements present in IQ and CIOQ. In terms of resources, the number of physical memories required for IQ (N) and CIOQ ($2N$) architectures remains fixed regardless of the features supported. Conversely, the proposed column-associative PSM architecture requires additional resources if QoS provisioning ($3\left(\frac{N}{s}\right)-1$) or multicast support ($\left\lceil\frac{b+1}{s}\right\rceil N-b$) are required. While speedup does impact the scalabiliyt, given decisions must now be made faster, it does provide a significant reduction in the physical memory requirement not present in either the IQ or CIOQ architectures.

Similar architectures have been proposed that employ a central stage of buffering to distribute to the memory bandwidth requirement. One such architecture is the load balanced router which attempts to exploit the uniformity of the traffic matrix to apply a Birkhoff-von Neumann capacity decomposition approach to achieve $O\left(1\right)$ placement complexity. This approach can achieve 100% throughput and offers achievable memory bandwidth requirements of $2R$. However, the load balanced router has a significant drawback in that mis-sequencing of packets is common. This issue can be resolved by incorporating a coordination buffer at the egress to resequence packets. The use of a coordination buffer require a crossbar, operating at $2NR$, to be located between the shared memory and egress buffering. The load balanced router, compared to the proposed PSM architectures, offers equivalent throughput and delay guarantees. Howevever, the proposed PSM architectures do not require the construction and decomposition of the traffic matrix which results in a less complex decision process.

The final approach to be discussed is the distributed shared memory (DSM) architecture. This architecture utilizes a shared pool of memories which are located in linecards that are interconnected by a crossbar. Packets arriving to a DSM switch are immediately switched to a line card, using the crossbar, and placed in memory. Packets remain in memory until their scheduled departure time arrives. At that point, the packet is read from the line card and switched, by means of a crossbar, to the correct output. As packets must traverse the crossbar upon arrival and departure, the resulting crossbar operating rate is

at least $2R$. The initial placement into memory does not ensure a packet is placed into the line card from which it will depart. As a result, the DSM switch architecture requires a crossbar operating rate of $4R$ and a total memory bandwidth of $3NR$ to emulate the FCFS OQ switch performance. By comparison, the column-associative PSM architecture, for all implementations, requires a memory bandwidth of $2sR$. To provide QoS guarantees, the crossbar operating rate and memory bandwidth requirement must increase to $4R$ and $4NR$, respectively. Clearly, the lack of an intelligent placement process yields an inefficient architecture with regards to the crossbar operating rate and memory bandwidth requirements. Neither limitation is present in the column-associative and row-associative packet placement architectures proposed in this dissertation.

Chapter 5

Physical Realization of Packet Placement Algorithm

In this chapter, we provide a detailed discussion on the different implementation aspects pertaining to the proposed pipelined PSM architectures. In particular, we focus on the critical design path, make observations regarding scalability, and ramifications associated with incorporating speedup into each design. Moreover, we present a modular architecture for traffic generation that is scalable with respect to the number of packet generation modules and switch port densities supported. We conclude this chapter by discussing the implementation results, in the context of simulation accuracy, resource consumption and the scalability of the memory management core and traffic generator.

5.1 Switch Fabric Design Flow

5.1.1 Row-Associative Packet Placement Architecture

In Section 3.1, the row-associative memory-management algorithm for PSM switches employing a pipeline architecture was introduced. In the proposed router architecture, the memory-management algorithm is implemented using a multi-stage pipeline architecture that distributes the packet-placement process, thereby exchanging fixed latency for increased execution speed. The proposed pipeline architecture consists of $\frac{L(L+1)}{2}$ cell buffering

units arranged in a square structure. Each row is associated with one of the parallel shared memory units. Hence, the architecture requires L parallel shared memories. Incoming packets from input port i are initially inserted into row i. The underlying mechanism is that at every time slot, packets are horizontally shifted one step to the right, with the exception of the diagonal cells of the structure. The diagonal cells have the ability to move vertically to another row of the same column. A cell moves vertically if any of the following two conditions are met: (a) the memory associated with the row in which it is currently located already contains a packet with the same departure time; (2) there is another cell ahead of it in the same row with the same departure time. Therefore, vertical moves are used as a means of resolving memory placement contentions. The goal of the scheme is that once a packet reaches the last column of the pipeline, it is guaranteed to be located in a row, having an associated row memory, that does not contain any packets with the same departure time.

It should be evident that the critical path in the design is the memory assignment process that takes place at the diagonal elements, or decision cells, of the pipeline structure. Let us review the assignment process. Each decision cell, residing on row r_i ($i \in [1, 2, \ldots L]$) must determine whether a packet, p_i, can be placed at the memory associated with row i. The memory is considered available for packet placement if it does not already contain a packet with the same departure time d_i as the packet residing in the decision cell. If the memory is occupied with a packet with the same departure time, then the packet is shifted to another row.

The following three criteria govern the movement of packets to a new row r_n:

1. The memory located on the selected row (r_n) cannot contain a packet departing at time d_i.

2. No other packets on row r_n, ahead of the packet moved and which have yet to reach a diagonal location, can have a departure time d_i.

3. Row r_n must not contain one of the $N-1$ possible packets that have the same arrival time as the shifted packet. In other words, the position in the new row should be unoccupied.

In light of these placement rules, an availability vector of length k is created to indicate locations available for the placement of a packet at time d_i. In a distributed shared memory switch, each shared memory will contain km cells representing k consecutive departure times. A decision cell need not know the precise number of locations available for a given departure time, only that at the number is less than m. Thus, the size of the availability vector presented to the decision cell is simply k. In order to perform the appropriate placement selection for packet p_i with departure time d_i, the column of bits pertaining to position d in the availability map is examined. In accordance with lemma 3, we can state that since a packet always shifts into the next block, the maximal number of rows that is to be examined by each diagonal cell is $(\frac{s+m}{sm})N - 2$. This inherently bounds the critical path of the design, as it defines an upper bound on the search space that must be considered at each step of the memory management process.

While the availability map provides information pertaining to the memories contents, it does not provide any information regarding the location of the $\frac{N}{s} - 1$ packets that potentially arrived during the same time slot as p_i. Having stated that a vector of length $\left(\frac{s+m}{sm}\right) N - 2$ is sufficient to locate an available row, an occupancy vector must be created to determine the locations of $\frac{N}{s} - 1$ packets within the $\left(\frac{s+m}{sm}\right) N - 2$ subset, reflecting potential adequate rows. Hence, the bitwise-OR of the occupancy vector and availability vector provides a single binary decision vector representing viable rows for packet placement. The decision vector defines rows which (1) are not associated with memories that contain a packet with departure time d_i, and (2) do not contain a packet with the same arrival time as p_i.

The memory-management algorithm is based on selecting an available memory from one of $\left(\frac{s+m}{sm}\right) N - 2$ memories, such that all N packets are placed at the end of a at most $O(N^{1.5})$ phases. To accomplish this, we first determine whether the packet in the decision cell is blocked by a packet with the same departure time as p_i, as indicated by the availability vector. If p_i is blocked, the decision vector is then used to to select an available row from one of $\left(\frac{s+m}{sm}\right) N - 2$ rows. The result of this decision process is that p_i is either written to the memory associated with the row in which it resides or placed in a row that is free of conflict. The decisions made at each location are represented by the flowchart in Figure 5-1.

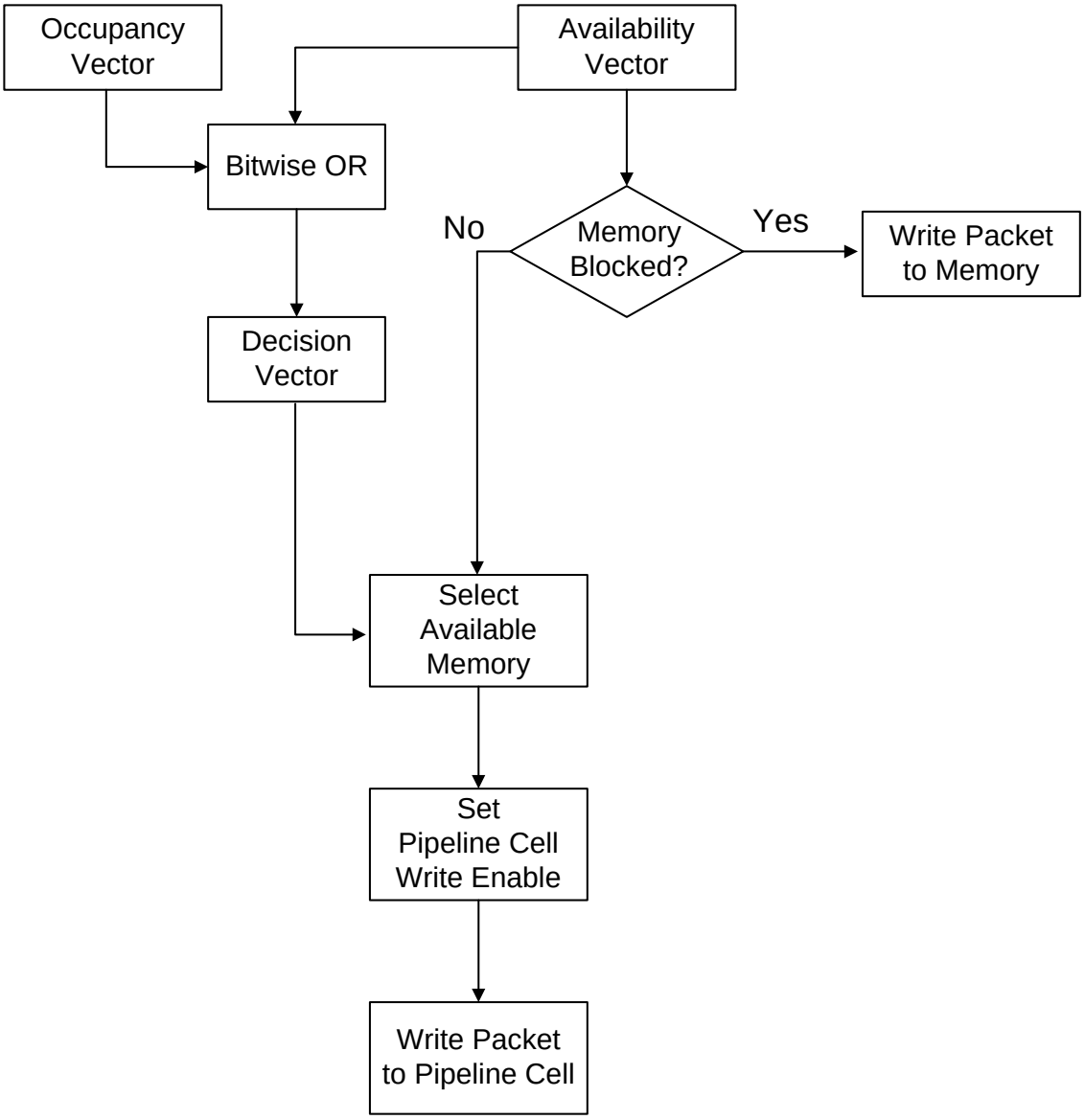

Figure 5-1: Outline of the process performed at the decision cells.

While placement is certainly the more difficult task, some consideration must be given to retrieving packets from a shared memory structure. One practical solution is to present a single packet bus to each shared memory unit. As packets are read from memory, each output port will have at most one packet destined for it at any given time. Correspondingly, we can present a shared packet bus of size $N \times packet\_size$ to each memory unit such that bus slice j, having width corresponding to a packet size, represents a packet destined for output j. Each shared memory unit determines the destination (j) of the packet it is transmitting to an egress port and only drives the j^{th} bus slice of the packet bus. As a result, each egress port will only be responsible for transmitting the packet located at the j^{th} bus slice, corresponding to the port enumeration of the packet bus. Implementation of this scheme requires the need for an $N - to - 1$ multiplexer located in each memory cell.

5.1.2 Column-Associative Packet Placement Architecture

Critical Path Analysis

In Section 3.3, a column-associative memory-management algorithm for PSM switches was introduced as an alternative to the row-associative approach discussed above. For the column-associative memory management, a multi-stage packet placement architecture was again employed to distribute the memory bandwidth requirement. In this instance, the proposed pipeline architecture consisted of $N \times (2N - 1)$ cell buffering units, where N is the number of ports, arranged in a rectangular configuration. Rather than couple rows with memories, we instead chose to couple columns with memories to provide an equal placement opportunity for all packets, having the same arrival time, to be placed in an associated column memory. In this arrangement, the architecture required $2N - 1$ shared memories. Subsequent enhancements, in the form of QoS, multicast, and load balancing, to the column associative memory management algorithm required an increase in number of memories and, consequently, number of cell buffering units required. The associated memory and cell buffering requirement for each enhancement is provided in Table 5.1.

The placement process receives incoming cells from input port i, then performs an initial insertion into row i of column 1. For each placement, the underlying mechanism is that, at every time slot, cells are horizontally shifted one step to the right, with the exception being

Table 5.1: Memory and cell buffering requirement for enhancements associated with the column-associative memory management algorithm.

Architectural Enhancement	**Memory Requirement**	**Cell Buffering Requirement**
Multicast	$\lceil (sb+m) \ / \ sm \rceil N - b$	$N \times \lceil (sb+m) \ / \ sm \rceil N - b$
QoS	$3\,(N \ / \ s) - 2$	$N \times 3\,(N \ / \ s) - 2$
Load Balancing	$4N - 2$	$N \times 4N - 2$

cells selected for placement. In the most basic case, a cell is selected for placement if there are no other cells, located in the associated column memory with a matching departure time. Employing multicast requires no other cells, located in the selected memory, contain matching departure time for each departure time associated with the multicast cell to be placed. For QoS cells, we simply require there not be $N-1$ cells having a departure time either before or after that of the cell selected for placement. Regardless of the enhancements, the goal of the scheme is that once a cell reaches the last column of the pipeline, it is guaranteed to be located in a column, having an associated memory, that is devoid of placement conflicts.

For the column-associative memory management architecture, clearly the critical path in the design is again the placement decision process that must be made by the placement primitives located in each column. Each primitive must determine whether a cell, residing in column $C_i\,(i \in [1, 2, ..., k])$, can be placed in the associated column memory. In the instance where the queueing disciple is PIFO, cells located in each column must be inspected to determine if they conflict with those previously placed in the associated column memory. If the cell is determined to be a viable candidate for placement, it is extracted from the pipeline and written into the column memory. Otherwise, the cell will advance to the next column where its placement viability will be re-evaluated for the next column memory.

To determine the feasibility of placing a cell in a column memory, each memory maintains a binary occupation vector, of length k denoting the depth of the memory, to indicate departure times that are currently reserved. To determine which cells in a given column are available for placement, a requests vector is created as cells are inserted into the pipeline. This vector, also of length k, provides a bitmap corresponding to the departure time for each of the cells located in a column. For unicast traffic, a single bit is set to indicate the

requested location. In our discussion of multicast traffic, we utilized the notion of a bound, b, which limited the number of replications, or placement requests, that could be made by a single cell. Consequently, a requests vector for a multicast cell can have at most b bits set to indicate the requested placement location.

The requests and occupation vectors are constructed and maintained such that we can obtain the set of viable cells (referred to as the candidates vector). If we denote the requests vector by α, and the occupation vector as ϕ, then each cell, c_i, represents a single bit in the candidates vector, Θ, which is constructed by performing the following bitwise operations

$$\Theta\left(i\right) = ((\neg\left(\alpha \ \oplus \ \phi\right) \ \& \ \alpha) == \alpha), \tag{5.1}$$

where $\oplus$ and $\neg$ denote the bitwise xor and bitwise not operations, respectively. The bitwise xor of the requests vector and the occupation vector determines whether the requested locations, in the case of multicast traffic, are available for c_i. The result is then masked with, and later compared to, the requests vector to produce a single bit which indicates the cells viability for a given memory. The candidates vector allows a priority encoder to be used such that a cell can quickly be selected for placement into the corresponding column memory. The cumulative result of this decision process is that c_i is either written to the memory associated with the column in which it resides or shifted to the next stage of the pipeline.

Given that the xor operation can be achieved at high speed, it becomes apparent that the priority encoder is the predominately time-consuming function. The complexity of a priority encoder is generally acknowledged to be $O(log\frac{N}{s})$, where N denotes the number of elements at its input, and s represents the placement speedup. It is noted that this complexity pertains to 5 -bit-level operations (rather than more complex arithmetic abstraction), clearly suggesting that the method is very efficient from a computational standpoint.

Alternative Pipeline Structure

The column-based memory management algorithm discussed thus far consists of $M \times N$ cell buffering units arranged in a rectangular structure, where N is the number of switch

ports and M represents the number of parallel memories. Recall from the implementation discussion above, cells are introduced at the leftmost column and are guaranteed placement upon reaching the final column. Once a placement decision is made, the cell is written directly to memory. As a result, there can be up to M write operations that occur in parallel. While the architecture will support M concurrent write operations, the system only needs to support an aggregate of N write operations per time slot.

The use of $M \times N$ cell buffering units implies a $c \times M \times N$ register requirement, where c is the cell size. If we consider a 32 port switch employing 63 memories, this would require approximately 1 Mbit of registers to implement a pipeline using 64 byte cells. To reduce this requirement, we observe that at most N cells are presented to the fabric during a single time slot. In addition, each of these N cells are guaranteed placement upon reaching the final column. Therefore, it is feasible to simply indicate which of the M memories a cell should be written to as it passes through the pipeline. Once the N potential cells arrive at the final column, they can all be written to memory simultaneously. Memory contention is avoided as the pipelined placement algorithm dictates that all cells, with the same arrival time, contain placement assignments to unique memories.

Given that cell placements can be delayed until the final column, it is no longer necessary to propagate each cell through the pipeline. Rather, we can extract the departure time from each arriving cell, then enqueue the cell in a delay buffer of depth M. Noting that only the departure time is required by a cell placement unit, we can simply forward the extracted departure time to the pipelined memory management algorithm. As the departure time for each cell propagates through the pipeline, each column placement unit will attach the index of the associated column memory to a selected departure time. Once all departure times have reached the final column, the corresponding cell will be read from the delay buffer and placed in the memory indicated by the attached index.

Propagating only the departure time and a memory index through the pipeline provides considerable reduction in the register requirement. The placement pipeline no longer requires $c \times M \times N$ registers to store cells as they are being placed. Instead, each cell buffering unit is only required to store the departure time, DT, and the index of the memory for which the cell has been marked for placement. Therefore, the pipeline needs to be of

size $(\lceil \log_2 (DT) \rceil + \lceil \log_2 (M) \rceil) \times M \times N$, which is considerably smaller than a full 64-byte cell. Considering, for example, a 64 port switch employing 127 memories and 256 departure times, we can now reduce to the register requirement to be approximately 28 Kbits, considerably less than the 1 Mbit required when cells are propagated. This reduction does have an associated cost in that we now require N multiplexors to place each cell in one of the potential M memory destinations.

5.2 Packet Generation

Traffic modeling plays a key role in the study of packet switching systems, such as Internet routers. As line rates increase towards tens of gigabits per second, the duration of individual packets decreases, rendering real-time traffic generation a fundamental engineering challenge. In evaluation of these systems, it is critical to reproduce traffic conditions that approximate the target environment. Additionally, the ability to generate traffic flows that establish the limitations of a given algorithm or architecture is highly desirable. To address these issues, we present a reconfigurable high-speed hardware architecture for heterogeneous multimodal packet generation.

The traffic generator we propose, presented in Figure 5-2, provides a modular architecture that is scalable with respect to the number of packet generation modules and switching port densities supported. There are five major components: a random number generator, arrival process generator, source-destination lookup, packet formatter, and a departure-time calculation module. Each module, with the exception of the departure time calculation module, can be replicated N times and is capable of generating traffic from one of k flows. The scalability of the departure time calculation module resides in the fact that it can accept destination addresses from N ports and will provide N departure times according to a FCFS scheduling algorithm.

Establishing the arrival process generation, we have stated that support for both the Bernoulli and two-state Markov modulated arrival processes is required. In the case of the Bernoulli process, packets are uncorrelated and possess no temporal dependency properties. Thus, each packet can be viewed as having a unique source-destination pair from cycle to

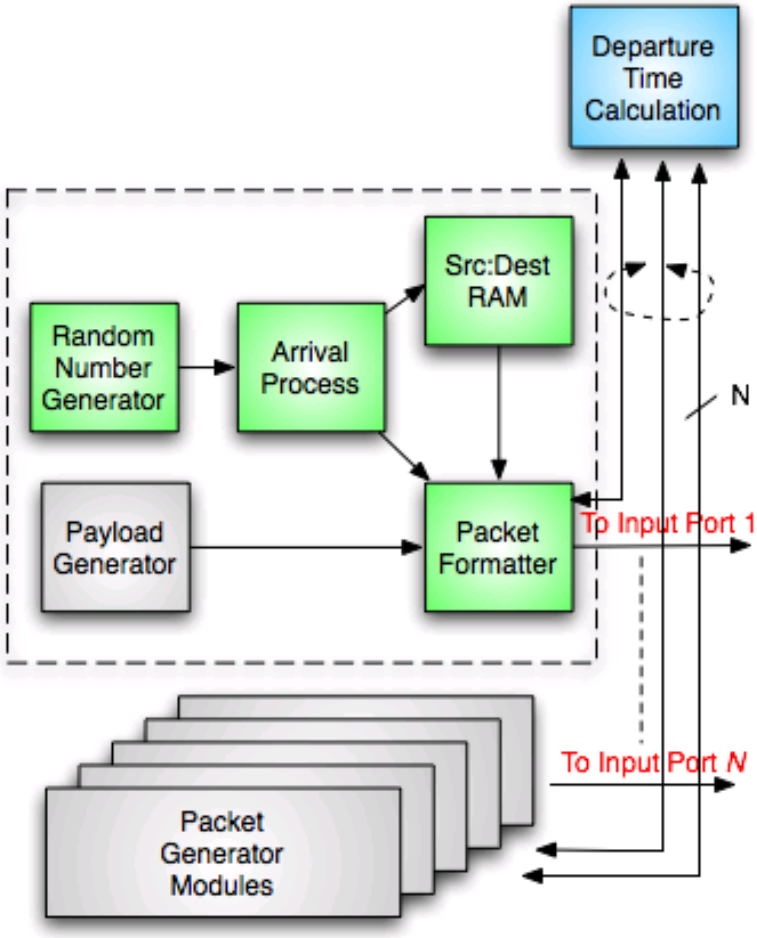

Figure 5-2: Packet Generator Architecture

cycle. For this reason, the Bernoulli arrival process module is designed to compare a random arrival process value, r_a, produced by the random number generator, to the user-configurable per-flow probability, p_k, in order to determine whether a packet should be transmitted on a per-cycle basis. The value of p_k represents the probability that a packet will be transmitted when flow k is selected. This set of k probabilities is stored in the Source-Destination RAM, with the index denoting the flow selected, using the random flow value, r_f, that is again produced by the random number generator. By dividing the n-bit number space of r_f into k distinct regions, the flow index is then determined using comparators to determine the k^{th} region in which r_f resides. Upon calculation of both r_a and r_f, the packet formatter is instructed to send a packet when the following evaluation is true: $r_a \leq p_k$.

The two state Markov-modulated arrival process is simply an extension of the Bernoulli arrival process architecture. The per-flow probability is again selected using the random flow value, r_f, by dividing its n-bit number space into k distinct regions to determine the k^{th} region in which r_f resides. However, the flow index is now used to retrieve two flow probabilities, p_k and q_k. The value of p_k represents the probability that the Markov chain remains in the ON and continues to transmit packets. Whereas, the value of q_k represents the probability that the Markov chain remains in the OFF state and does not transmit a packet. These transition probabilities are compared to the random arrival process value, r_a, on each cycle to determine whether to perform a state transition. The bursty nature of the two-state Markov modulated process can be maintained by storing new flow probability values only when the Markov chain transitions to the OFF state.

Prior to transmitting that a packet to the packet formatter, the arrival process module must also produce a source-destination pair prior to packet construction. The approach to generating a source-destination pair is to store the source-destination pairs in a RAM, of depth k, then perform the selection in conjunction with flow probabilities. The entire solution can be realized with three RAMs located in the source-destination RAM module. Two RAMs contain the arrival probabilities, $p_k$and q_k, while the third RAM contains only the destination address for the packet. Since the source address is the same for all k flows in a single packet generator module, it can be represented by a single register that is programmed during the initialization.

Once the target destination for packets presented at each port has been determined, the associated departure time can be calculated prior to transmission to the switch fabric. The packet generation architecture we present supports systems implementing first-come-first-serve (FCFS)-based scheduling. Packets arriving at a single port can have monotonically increasing departure times based on the arrival order. To calculate the departure times for a system with N ports, one must be able to compute departure times for the adversarial case where all N packets are destined for the same output port. The computation of departure times in a serial order would require that each computation complete in $\frac{50}{N}$ ns. Given that packet switching architectures consisting of 64 ports or more are common, this would require that each departure time be calculated in less than one nanosecond. Clearly, this is not practical. Instead, departure times must be calculated in parallel at the cost of silicon area. In Figure 5-3, the implementation of the departure time calculation fabric is presented. Since departure times monotonically increase as packets arrive at a given output, a register file containing the current departure times for each output is stored in the base departure time register file. To compute the departure time at each port, the encoded destination address presented to the fabric input is first split into an N-bit vectored request where a single active bit represents the packet destination. Once the N-bit vector has been generated for each destination address, the base departure time offset is computed by summing the q^{th} bit, where q identifies each output port, of each N-bit vector. The result of the summation represents the number of packets arriving at output port q. This base departure time offset is then added to the base departure time at the end of each clock cycle. This computation contains the critical path in the design due to the requirement of N $log_2(N)$-to-N multiplexors, whose output is immediately fed to a set of N adders that sum N bits. This sum is then added to the base departure time register prior to being registered.

Now that the method for calculating the base departure times has been established, the method for calculating per-packet departure times must now be stated. The departure time for each packet is a function of the destination address for every other packet arriving at that time cycle and the base departure time for each packets destination. Since we have the base departure time for each destination, all that is needed is to assign an offset for

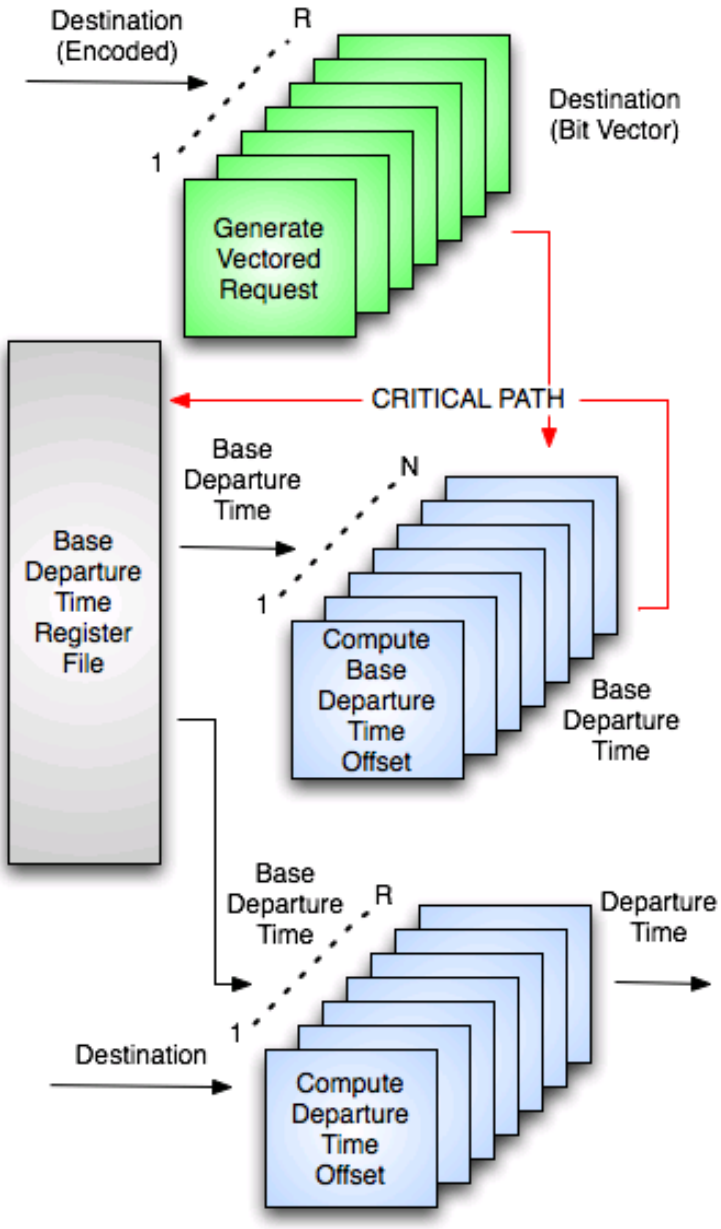

Figure 5-3: Departure Time Calculation Architecture

each packet relative to other packets arriving during the same time slot. The offset for each packet is calculated relative to the input port at which it was received. This effectively establishes a priority to each input port in terms of calculating its departure time offset. The first input port automatically has the highest priority. For each successive port, a comparator is needed to determine if a higher priority input has the destination. The result of each boolean comparison is added to the packets departure time offset. From this, we can establish that a packet arriving at input port 1 is automatically given a departure time of 1. Hence, input port 2 will require one comparator and possess a departure time of either 1 or 2. For input port *N*, a total of *N-1* comparators is required to produce a departure time than can vary from 1 to *N*. Once the departure time offset is calculated, it is added to the base departure time offset register, specified by the destination address, to determine the per-packet departure time. The selection of this base departure time offset register requires N of N-to-1 multiplexors because each packet can possess one of N destinations.

Upon calculating the departure time, its output is forwarded to the packet formatter which is responsible for synchronizing the departure time, destination and packet transmission information. The departure time, as well as the packet validity, is considered test information. For this reason, this information is embedded in the payload, by the packet formatter, for evaluation and performance analysis at the output of the switch. After formatting has completed, the packet is presented to the ingress interface of the switch fabric.

5.3 Hardware Implementation Results

5.3.1 Packet Generator Module

FPGA Implementation

The design of the packet generator architecture has been coded in Verilog HDL and synthesized using the Synplicity SynplifyPro synthesis tool targeting a Xilinx Virtex-4 XC4VLX80-11-FF1148 FPGA device. This implementation consists of 64-ports and supports 128 flows. Packets are produced at the output of the packet generator every 50 *ns*.

In order to maximize the devices available for implementation, we decided to reduce the logic resources required to compute departure times by limiting the computations to $N/2$,

Table 5.2: FPGA Implentation Cost and Expected Slack

Module	**LUTs**	**Slack (ps)**
Departure Time Calculation	24674	+0.657
Packet Generator	19214	42.83
Complete Design	43793	+0.27

or 32, departure times in a single cycle. By clocking the departure time calculation fabric at least $2x$ faster than the rest of the system, 64 departure times could be computed in the required 50 ns period. However, synchronization of the two clock domains requires an additional two clock cycles. For this reason, the departure time computation fabric operates at 12.5 ns, or $4x$ faster than the rest of the system.

The full implementation of the 64-port packet generator required 43,793 Lookup-Tables (LUTs), or 61% of the LUTs available in a Xilinx Virtex-4 XC4VLX80 device. Targeting a 20 MHz operation frequency, synthesis was able to meet timing with 27 ps slack. Recall three RAMs, two for probabilities and one for the source-destination pairs, are required by each generator module, resulting in a $3N$ RAM architectural requirement. Accordingly, this implementation consumed 192 of the 200 RAMs made available by the device.

Lookup-Table (LUT) resource consumption, presented in Table 5.2, reflects the implementation cost of each module. The departure time computation fabric fits comfortably in this device suggesting that the actual computation fabric could be designed to operate at the same clock rate as the rest of the system.

Simulation Observations

Correct operation of our packet generator architecture was verified using stimulus written in SystemC, and co-simulated with the Verilog HDL implementation, using the Synopsys VCS-MX simulator. Programming the individual packet generators, as well as simultaneously starting all packet generators after configuration, was performed using this stimulus. Configuration of the device is performed using a single shared bus with a per-generator address decoder that is located at the top-level of the packet generator implementation. The user has the ability to set values for each of the $2k$ flow probabilities, the k source-destination pairs, seeds for the two random number generators (LFSRs), and the arrival

Table 5.3: Offered Load Variance

Simulated Samples	Expected Offered Load	Actual Offered Load
1500	0.8	0.806832
200,000	0.8	0.800771
1×10^7	0.8	0.799683

process type for each packet generator. Hence, the duration of the configuration process for the entire system consists of $(2k + 3)N$ clock cycles.

Once the device is configured, each packet generator is enabled to generate packets concurrently. To verify the correct operation, the load offered by each packet generator was measured to determine the variance between the expected offered load and the offered load produced via simulation. Table 5.3 presents the simulation results for the 64-port packet generator configured to produce an offered load of 0.8.Results presented in this instance were for a single instance, but remain consistent across mutliple iterations. The variance between the expected and offered load decreases as the simulation length indicating that a real-time implementation would yield results that closely approximates the desired offered load.

5.3.2 FoC Implementation Results

Row-Associative Packet Placement Architecture

To establish the viability of the FoC architecture, employing the row-associative memory management approach, the algorithm was implemented in hardware targeting an Altera Stratix II EP28S60 FPGA device. The implementation consisted of eight ports, each operating at 10 Gbps, representing a switch with an aggregate capacity of 80 Gbps. The maximum departure time, k, was set to 64. Further, the system was designed with a placement decision speedup (s) of four, requiring packet placement decisions to be performed in approximately 12.5 ns. Additionally, there were four unique locations for each departure time in each row memory, i.e. $m = 4$. The prototype system, with speedup and multiple packet placement, utilized eight physical memories consuming a total of 26.624 kb, including logic mapped to memory. This assumed that only packet headers are processed (as payload is irrelevant to the decision making process). However, if 64-byte payload is assumed,

Table 5.4: Number of memories in the row-associative PSM switch

Switch Ports (N)	**Speedup** (s)	**Memory Units**
8	2	19
8	4	5
16	2	58
16	4	18
32	4	51
64	4	144

the aggregate on-chip memory requirements increased to 1.05 Mbit. While eight physical memories were implemented, principally for symmetry and test purposes, no more than five memories were actually required (as stated in Table 5.4).

The design required 17,383 adaptive look-up tables (ALUTs), or 35% of the ALUTs available on the target device. Proper evaluation of the switch was established by attaching a packet generator, implemented using an Altera Cyclone EP1C6Q-240C8 device, to apply both Bernoulli i.i.d. as well as bursty traffic to the PSM switch fabric. In varying the traffic load and patterns over a wide range of possible scenarios, the viability of the proposed algorithm in a real-time environment was established. The overall latency contributed by the architecture with respect to a pure output-queued router was 100 ns (8 stages of 12.5 ns each).

Column-Associative Packet Placement Architecture

The technical feasibility of the column-associative packet placement architecture was established by evaluating both the critical path and area requirement for a single column. As placement decisions are independent, the increase in delay and area requirements as columns are added is approximately linear with respect to port density. To obtain the relevant metrics, a switch fabric implementation consisting of 32 ports, each operating at 10 Gbps, having an aggregate capacity of 320 Gbps, was developed. As before, the system was designed with a placement decision speedup (s) of four, requiring packet placement decisions to be performed in approximately 12.5 ns. Additionally, there were four unique locations for each departure time in each column memory, i.e. $m = 4$. Utilizing an Altera Stratix II EP28S60 FPGA device, the delay associated with the decision process (criti-

cal path) was obtained and determined to be 10.844 ns. For the area requirement, the single column implementation of the placement process yielded an adaptive look-up table (ALUT) requirement of 1,279. From this, we can conclude that a full system consisting of $\left(\frac{s+m}{sm}\right) N - 1$ columns, in this instance 15 columns, would necessitate a minimum ALUT requirement of 19,185. One last metric of note is the overall latency contributed by the architecture, with respect to a pure output-queued router. This was found to be 187.5 ns, which is derived from having 15 stages, each requiring 12.5 ns.

For the implementation presented above, we defined the maximum departure time, k, to be 64 and the fixed cell size to be 64 bytes. Additionally, we can establish the 32 port system requires requires 15 physical memories since $s = m = 4$. Given multiple departure times are offered per column memory, *i.e.* $m = 4$, there must be 256 cell locations per memory. This results in a total memory requirement of 1.97 Mbit. Clearly, this memory requirement is practical given the current level of memory provisioning present in current VLSI offerings. In fact, the relatively small resource requirements of the column-associative packet placement architecture yields a solution that allows further consolidation of additional switching functions.

Chapter 6

Conclusions

6.1 Summary of Dissertation Contributions

6.1.1 Performance Analysis of Output Queued Cell Switches

In Chapter 2, a novel performance analysis of output queued cell switches with general independent heterogeneous traffic was presented. Random arbitration was employed whereby non-empty queues compete equally for service within each switching interval. In particular, the case of bursty two-state Markov-modulated arrivals in which input ports generate bursty streams that are non-uniformly distributed was studied. Under the assumption of a memoryless server, the probability generating function of the interarrival process is utilized to derive an approximation for the queue size distribution. The methodology established provides a flexible tool in obtaining bounds on the behavior and expected performance characteristics of output queued switches under a wide range of correlated traffic scenarios. The validity of the analytical inference was established via simulation results.

6.1.2 Scalable Packet Placement Algorithms for PSM Switches

In Chapter 3, we described two distinct pipelined memory management algorithms for PSM switches. The first was a row-associative memory management algorithm which employed a triangular pipelined structure to place incoming packets into a pool of shared memories. Each row was coupled with a corresponding memory, such that placements were determined

by allowing each cell, located on the diagonal of the triangular structure, to determine whether it could be placed in the associated row memory. If the row memory was occupied, the cell would then select a row, from a set of $2N-1$ potential candidates, in which to place the conflicted cell. Essentially, the placement algorithm allowed the cell move vertically until an available memory was found. We then proceeded to introduce a column-associative memory management algorithm, whereby a rectangular pipeline structure was employed such that each column was coupled with a single memory. Individual cell placements were no longer constrained to the row in which they were located. Instead, all cells arriving at time t were coupled with a column memory for placement consideration during each time slot. At the end of each time slot, the group of cells, having the same arrival time, would then advance to the next column for further consideration. Upon reaching the last column in the pipeline, all cells were guaranteed placment.

Each of the proposed schemes exploit time-space trade-offs to offer reduced computational complexity of the memory management process, thereby gaining scalability at the cost of fixed latency. In each instance, we provided analysis that outlined the memory and delay bounds, while also highlighting mechanisms, such as memory speedup, that can further reduce these bounds. In Chapter 4, we extended the column-associative PSM architecture to offer QoS guarantees, support for multicast traffic, and provide load balancing capabilities. We concluded with a comparison spanning the performance, implementation, and scalability attributes, between the proposed PSM solutions and existing switch fabric architectures.

6.1.3 Hardware Realization of a Fabric-on-Chip Architecture

In the first section of Chapter 5, we provide a detailed discussion pertaining to the different implementation aspects of the proposed memory management algorithms. In particular, we focus on the critical design path, scalability, and the effect of speedup in reducing the memory requirements for the parallel shared memory switch. At the conclusion of Chapter 5, implementation performance attributes, such as resource consumption and scalability prospects for each packet placement algorithm were discussed.

6.1.4 High-Speed Reconfigurable Architecture for Heterogeneous Multimodal Packet Traffic Generation

The remainder of Chapter 5 was dedicated to an issue complementing the implementation the FoC, which is the design of a reconfigurable high-speed hardware architecture for heterogeneous mutlimodal packet generation. The proposed architecture supports flow aggregation as well as provides a modular architecture that is scalable with respect to the number of packet generation modules and switching port densities supported. Support for both Bernoulli and two-state Markov modulated arrival processes was provided. Furthermore, FPGA implementation and simulation results were discussed to emphasize the viability of this architecture.

6.2 Relevant Publications

The following is a list of publications pertaining to contributions made thus far, as described in this dissertation:

- **B. Matthews**, I. Elhanany, "Multicast and QoS Provisioning in Parallel Shared Memory Switches," *in review for IEEE Transactions on Computers.*
- Xike Li, I. Elhanany, **B. Matthews**, "A Scalable Frame-based Multi-Crosspoint Packet Switching Architecture," *in review for IEEE Transactions on Computers.*
- **B. Matthews**, I. Elhanany, "A Scalable Memory-Efficient Architecture for Parallel Shared Memory Switches," *2007 IEEE Workshop on High Performance Switching and Routing (HPSR),* May, 2007
- **B. Matthews**, I. Elhanany, V. Tabatabaee, "Accelerated Packet Placement Architecture for Parallel Shared Memory Routers", *Proceedings of IFIP Networking 2007,* May, 2007
- **B. Matthews**, I. Elhanany, "Switch Fabric on a Reconfigurable Chip using an Accelerated Packet Placement Architecture," *15th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA),* Monterey, CA., February, 2007.

- I. Elhanany, V. Tabatabaee, **B. Matthews**, "Fabric on a Chip: A Memory Management Perspective," *High-Performance Packet Switching Architectures, Springer-Verlag*, London, September 2006.

- **B. Matthews**, I. Elhanany, "On the Performance of Output-Queued Cell Switches with Non-Uniformly Distributed Bursty Arrivals," *IEE Proceedings on Communications*, Vol. 153, No. 2, pp. 201-204, April, 2006.

- **B. Matthews**, I. Elhanany, V. Tabatabaee, "Fabric on a Chip: Toward Consolidating Packet Switching Functions on Silicon," *Proc. 2006 IEEE International Conference on Communications (ICC)*, Istanbul, Turkey, June, 2006.

- **B. Matthews**, I. Elhanany, "A High-Speed Reconfigurable Architecture for Heterogeneous Multimodal Packet Traffic Generation," *Proc. IEEE 48th Midwest Symposium on Circuits & Systems (MWSCAS 2005)*, Cincinnati, Ohio, August, 2005.

6.3 Additional Publications

The following is a list of publications pertaining to contributions made in areas outside the focus of this dissertation:

- **B. Matthews**, I. Elhanany, "Hardware Architecture for High-Speed Real-Time Dynamic Programming," *IET Proceedings on Computers and Digital Techniques,* November 2007

- **B. Matthews**, I. Elhanany, "Scalable Hardware Architecture for Real-Time Dynamic Programming Applications", *2006 IEEE Symposium on Field Programmable Custom Computing Machines (FCCM),* Napa, CA., April 2006

Bibliography

Bibliography

[1] V. T. R. N. A. P. I Elhanany, D. Chiou, "The network processing forum switch fabric benchmark specifications: an overview," *IEEE Network Magazine*, vol. 19, no. 2, pp. 4–9, 2005.

[2] Cisco systems crs1 multishelf router. Available at: www.cisco.com.

[3] L. G. Roberts, "Beyond moore's law: Internet growth trends," *Computer*, vol. 33, no. 1, pp. 117–119, Jan. 2000.

[4] Y. Tamir and G. Frazier, "Higher performance multiqueue buffers for vlsi communication switches," in *15th Annual Symposium on Computer Architecture*, 1988, pp. 343–354.

[5] R. Z. S. Iyer and N. Mckeown, "Routers with a single of buffering," *ACM Computer Communication Review (SIGCOMM'02)*, pp. 251–264, 2002.

[6] H. Liu and D. Mosk-Aoyama, "Memory management algorithms for dsm switches," *Stanford Technical Paper*, July 2004.

[7] A. P. A. Aziz and V. Ramachandra, "A near optimal scheduler for switch-memory-switch routers," in *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, July 2003, pp. 343–352.

[8] A. A. A. Prakash and V. Ramachandra, "Randomized parallel schedulers for switch-memory-switch routers: Analysis and numerical studies." IEEE INFOCOM 2004, 2004.

[9] S. Iyer, A. Awadallah, and N. McKeown, "Analysis of a packet switch with memories running at slower than the line rate," in *INFOCOM (2)*, 2000, pp. 529–537. [Online]. Available: citeseer.ist.psu.edu/iyer00analysis.html

[10] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *Communications, IEEE Transactions on [legacy, pre - 1988]*, vol. 35, no. 12, pp. 1347–1356, Dec. 1987.

[11] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-speed switch scheduling for localarea networks," 1993. [Online]. Available: citeseer.ist.psu.edu/anderson93high.html

[12] M. Karol, K. Eng, and H. Obara, "Improving the performance of input-queued ATM packet switches," in *INFOCOM '92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, Florence, Italy, May 1992, pp. 110–115.

[13] S. Majumder and R. Gangopadhyay, "Optimum architecture for input queuing ATM switches," *Electronics Letters*, vol. 27, no. 7, pp. 555–557, Mar. 1991.

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

[15] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input output queued switch," in *INFOCOM (3)*, 1999, pp. 1169–1178. [Online]. Available: citeseer.ist.psu.edu/article/chuang98matching.html

[16] B. Prabhakar and N. McKeown, "On the speedup required for combined input and output queued switching, Tech. Rep. CSL-TR-97-738. [Online]. Available: citeseer.ist.psu.edu/prabhakar97speedup.html

[17] T. Chaney, J. Fingerhut, M. Flucke, and J. Turner, "Design of a gigabit ATM switch," in *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 1, Kobe, Japan, Apr. 1997, pp. 2–11.

[18] C. Chang, D. Lee, and Y. Shih, "Mailbox switch: a scalable twostage switch architecture for conflict resolution of ordered packets," 2004. [Online]. Available: citeseer.ist.psu.edu/chang04mailbox.html

[19] C.-S. C. et al., "Load balanced Birkhoff-von Neumann switches, Part I: One-stage buffering," 2001. [Online]. Available: citeseer.ist.psu.edu/article/chang01load.html

[20] C.-S. Chang, D.-S. Lee, and C.-M. Lien, "Load balanced Birkhoff-von Neumann switches, part II: Multi-stage buffering," 2001. [Online]. Available: citeseer.ist.psu.edu/chang01load.html

[21] I. Keslassy, S. Chuang, and N. McKeown, "A load-balanced switch with an arbitrary number of linecards," 2003. [Online]. Available: citeseer.ist.psu.edu/keslassy04loadbalanced.html

[22] I. Keslassy and N. McKeown, "Maintaining packet order in two-stage switches," 2002. [Online]. Available: citeseer.ist.psu.edu/keslassy02maintaining.html

[23] T. Field, U. Harder, and P. Harrison, "Network traffic behaviour in switched ethernet systems," in *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, Oct. 2002, pp. 33–42.

[24] G. K. Zipf, *The Psycho-Biology of Language: An Introduction to Dynamic Philology.* Cambridge, MA: MIT Press, 1968.

[25] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "On the implications of zipf's law for web caching, Tech. Rep. CS-TR-1998-1371, 1998. [Online]. Available: citeseer.ist.psu.edu/breslau98implications.html

[26] B. A. Mah, "An empirical model of HTTP network traffic," in *INFOCOM (2)*, 1997, pp. 592–600.

[27] I. Elhanany and D. Sadot, "DISA: a robust scheduling algorithm for scalable crosspoint-based switch fabrics," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 535–545, May 2003.

[28] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[29] M. Arpaci and J. Copeland, "Buffer management for shared-memory atm switches." [Online]. Available: citeseer.ist.psu.edu/arpaci00buffer.html

[30] J. J. Hunter, *Mathematical Techniques of Applied Probability: Discrete Time Models: Techniques and Applications.* Academic Press, 1983, vol. 2.

[31] J. G. C. T. M. L. Chaudhry, U. C. Gupta, *On the Relations Among the Distributions at Different Epochs for Discrete-Time GI/Geom/1 Queues.*

[32] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, New York, NY, Mar. 1999, pp. 126–134.

[33] F. Yu, Q. Zhang, W. Zhu, and Y.-Q. Zhang, "Qos-adaptive proxy caching for multimedia streaming over the internet," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 3, pp. 257–269, Mar. 2003.

[34] M. Soderqvist and A. Gunnar, "Performance of traffic engineering using estimated traffic matrices," in *Proc. of Radio Sciences and Communication RVK'05*, Linkoping, Sweeden, 2005.

[35] H. Bruneel, "Queueing behavior of statistical multiplexers with correlated inputs," *IEEE Transactions on Communications*, vol. 36, no. 12, pp. 1339–1341, Dec. 1988.

[36] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *Proc. of ACM SIGCOMM '95*, pp. 231–242, September 1995.

[37] C. Minkenberg, "Integrating unicast and multicast traffic scheduling in a combinedinput- and output-queued packet-switching system," in *Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on*, Las Vegas, NV, USA, 2000, pp. 127–134.

[38] G. Nong and M. Hamdi, "Providing qos guarantees for unicast/multicast traffic withfixed/variable-length packets in multiple input-queued switches," in *Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on*, Hammamet, Tunisia, 2001, pp. 166–171.

[39] S. Iyer and N. McKeown, "On the speedup required for a multicast parallel packet switch," *IEEE Communications Letters*, vol. 5, no. 6, pp. 269–271, June 2001.

[40] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Multicast traffic in input-queued switches: optimal scheduling and maximum throughput," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 465–477, June 2003.

[41] J. Lee, J. Sohn, and M. Lee, "Design of a shared multi-buffer ATM switch with enhanced throughput in multicast environments," in *ATM Workshop, 1999. IEEE Proceedings*, Kochi, Japan, 1999, pp. 455–461.

[42] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control inintegrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[43] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1030–1039, June 1999.

Vita

William Brad Matthews was born in Livingston, Tennessee, on October 13, 1975. After finishing high school in 1994, he attended Tennessee Technological University, where he received a Bachelor of Science degree in Electrical Engineering and a Master of Science degree in Electrical Engineering, in Spring 1999 and Spring 2006 respectively. Between 1999 and 2000, he worked as a hardware design engineer for Raytheon Systems. He then left to join ASIC International, which later became Flextronics, where he was a Sr. Design Engineer from 2000-2004. In Spring 2004, he entered the doctoral program at the University of Tennessee, Knoxville, United States. In Fall 2007, he received a Doctor of Philosophy degree in Computer Engineering from the Department of Electrical Engineering and Computer Science at the University of Tennessee, Knoxville.