University of Tennessee, Knoxville
TRACE: Tennessee Research and Creative Exchange

Doctoral Dissertations | Graduate School

12-2006

Scheduling Algorithms for Scalable High-Performance Packet Switching Architectures

Xike Li
University of Tennessee - Knoxville

Follow this and additional works at: https://trace.tennessee.edu/utk_graddiss

 Part of the Computer Engineering Commons

Recommended Citation

Li, Xike, "Scheduling Algorithms for Scalable High-Performance Packet Switching Architectures. " PhD diss., University of Tennessee, 2006.
https://trace.tennessee.edu/utk_graddiss/1968

This Dissertation is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a dissertation written by Xike Li entitled "Scheduling Algorithms for Scalable High-Performance Packet Switching Architectures." 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, Hairong Qi, Michael Thomason

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 Xike Li entitled "Scheduling Algorithms for Scalable High-Performance Packet Switching Architectures". 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

Hairong Qi

Michael Thomason

Accepted for the Council:

Linda Painter
Interim Dean of Graduate Studies

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

Scheduling Algorithms for Scalable High-Performance Packet Switching Architectures

A Dissertation

Presented for the Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Xike Li

December 2006

Copyright © 2006 by Xike Li.
All rights reserved.

Dedication

This dissertation is dedicated to my parents, and my wife, my love.

Acknowledgments

I would like to thank my advisor, Dr. Itamar Elhanany, who brought me into this research area and support me throughout my studies. Numerous discussions with Dr. Itamar Elhanany lead to the design of multi-crosspoint based switching architecture and frame-based scheduling algorithms that provided the main motivation for this thesis. I would further like to thank Dr. Gregory Peterson, Dr. Hairong Qi and Dr. Michael Thomason, who served on my Ph.D. committee, for their time and input to this thesis.

I would like to give thanks to numerous other people at the networking research group (NRG), in particular Zhenzhen Liu and Brad Matthew for supporting me over the years.

Finally and definitely most, I would like to thank my family. I am extremly grateful to my parents and my wife, for all your love. I dedicate this thesis to you. Thanks.

Abstract

Packet switching fabrics constitute a fundamental building block of all Internet routers. As a core technology, the switching engine is responsible for enabling multiple input (ingress) ports to be dynamically linked to output (egress) ports, thereby allowing packets to effectively traverse the router. Scheduling algorithms, which play a key role in switching fabrics, determine the dynamic configurations of the input-output matchings. The ever growing need for additional bandwidth and more sophisticated service provisioning in next-generation networks necessitates the introduction of scalable packet scheduling solutions that go beyond legacy schemes.

Switch architectures can be coarsely classified into two categories, in accordance with the queuing mechanism they employ. In input-queued (IQ) architectures arriving packets are buffered at the ingress ports, awaiting to traverse the switch. In output-queued (OQ) architectures, arriving packets are immediately transferred to their corresponding egress ports at which they are buffered until their departure time. Scheduling algorithms for these two families of architectures vary significantly, yet they share the goals of maximizing throughput while minimizing the delay experienced by the packets.

This dissertation presents novel architectures and algorithms pertaining to both IQ and OQ switch designs. In the context of IQ switches, due to the increase in link rates directly resulting in a decrease of packet duration times, packet-by-packet switching is no longer considered a pragmatic approach for designing scalable systems. To address this challenge, this dissertation advocates the utilization of frame-based algorithms that relax the timing constraints imposed on scheduling algorithms while retaining key performance characteristics. The algorithms are studied via theoretical stability analysis and evaluated by means of statistical simulations. In the context of OQ switching, an efficient memory management algorithm that alleviates some of the principal limitations of OQ designs is presented and studied.

In an effort to introduce pragmatic solutions to the challenges associated with high-capacity packet switches, the focus of this work is to guarantee performance and scalability while utilizing off-the-shelf components that can be easily combined with custom hardware circuitry. We conclude by showing that the developed architectures and algorithms provide

solid cost-efficient foundations for supporting next-generation Internet switches and routers.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 An Overview of Packet Switching Architectures

1.1.1 Fundamentals

Packet switching is the now-dominant communication paradigm in telecommunication and computer networking. The Internet is the most well-known consumer of packet switching technologies. The latter evolved primarily as an efficient way to carry data communication traffic, characterized by access with buffering, statistical multiplexing, and variable throughput and average delay. In networks utilizing packet switching, messages to be transmitted are segmented into *packets*, each of which is labeled with a destination address, and transmitted individually despite the possibility of following different paths en route to their destination Once all the packets forming a message reach the destination, they are reassembled into the original message. This process is commonly referred to as *segmentation and reassembly* (SAR). Applying the principle of statistical multiplexing results in dividing the packet switched network into segments, by first assigning groups of users or *end hosts* that will share a link, and then interconnecting the groups with *packet switches.* The main task of the packet switch is to forward packets arriving at its input links to the output links corresponding to the packets' destinations. Therefore, a packet switch consists of two consecutive logical stages. First, a lookup stage which determines the correct output link of each arriving packet according, in accordance with its destination,

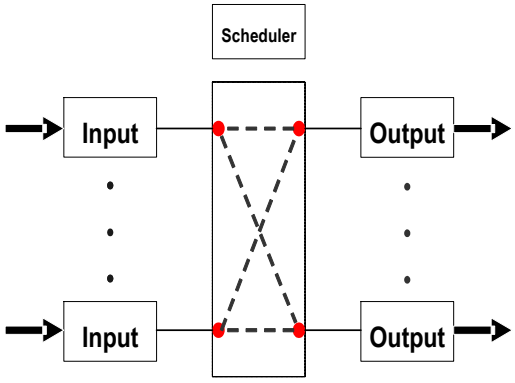

Figure 1-1: A coarse structural view of a packet switch, having N inputs and N outputs.

then a forwarding stage that transfers the packet from the input (or ingress) link to the designated output (or egress) link. This dissertation focuses on the forwarding stage of high-performance packet switches, as it constitutes a fundamental challenge in the context of designing next-generation networks.

As illustrated in figure 1-1, a packet switch can be viewed as a discrete-time system, having N inputs and N outputs, that enables packets arriving on its inputs to reach their requested destinations/outputs. It is commonly assumed that all lines have the same transmission rate, packets are of the same (fixed) size and the time axis is identically slotted into durations that are equal to a single packet-time interval. Moreover, we mainly consider the case whereby each packet is destined to a single output port, i.e., no multicast paths are allowed. Packet arrivals at the switch input have non-deterministic nature. Therefore, it is likely for more than one arriving packet to be simultaneously destined to the same output port. However, of theses packets only one can be forwarded to the output, while the others have to wait due to the fact that all links in the system have the same transmission rate. This form of *conflict* is referred to as *output contention.* When an output link is simultaneously requested by multiple input links, only one such input packet will be transmitted. The rest of the contending packets will have to be buffered in the queues associated with each input link, i.e., buffering of packets is a necessity in order to avoid output contention.

Even among packets destined for different output ports, contention for fabric-internal links may occur, e.g., in a ATM-based *Banyan* networks (Asynchronous Transfer Mode

(ATM) is a connection oriented data transfer technology, in which communication between any two end-points of the network is accomplished through the establishment of a virtual circuit that allows variable-rate full-duplex transfer of data between them). This phenomenon, commonly referred to as *blocking*, is undesirable since it 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 view of the requirements for fast switching speeds, mainly dictated by the evolution in optical switching, this dissertation will focus solely on non-blocking switch architectures.

In order to resolve output contention, we need to buffer those contending packets in the queues associated with each input link. The choice of buffering (or queueing) mechanism is of great importance with regard to the overall performance of the switch. The two fundamental queueing topologies that exist are input queuing and output queuing, which will be introduced in subsections 1.1.3 and 1.1.4, respectively. Before we delve into the two fundamental buffering mechanisms utilized in non-blocking packet switches, we first review the commonly used metrics to evaluate the performance of packet switches.

1.1.2 Performance Evaluation Metrics

In subsection 1.1.1, we mentioned that throughout this dissertation, it is assumed that all lines have the same transmission rate, packets are of identical (fixed) size and the time axis is slotted with slot sizes being equal to a single packet duration interval. The latter is equivalent to the transmission time of one single packet. Let L denotes the length of a packet in *bits,* and R the line rate in bits per second, then a packet duration interval would equal L/R. To evaluate the performance of packet switches, there are four sets of primary metrics broadly employed in most literature, which are: *average throughput* (or *output utilization), average packet delay, packet delay jitter and packet loss rate.* Detailed definitions of these metrics are provided below:

Definition 1 *Average throughput is the average number of packets that depart from the packet switch during one time slot per output port. It is also often referred to as output utilization, i.e., the fraction of time the output lines are busy.*

Definition 2 *Average packet delay is defined as the average time it takes a packet to traverse the switch.*

Definition 3 *Packet delay jitter is the distribution of packet delays around the mean.*

Definition 4 *Packet loss rate is the percentage of arriving packets dropped by the packet switch due to insufficient buffer resources.*

Throughout this dissertation, we will focus on the theoretical study of proposed scheduling algorithms for scalable high-performance packet switching architectures. The study is directed in terms of average throughput analysis under various packet traffic patterns, coupled with extensive simulation results pertaining to the average packet delay and packet delay jitter. To simplify the theoretical analysis, we assume an infinite buffer space is available, which suggests a zero packet loss rate. However, we will provide results that describe the buffer occupancy statistics, from which pragmatic buffer memory requirements can be derived.

1.1.3 Input Queued Switches

An obvious way to resolve output contention is to place buffers at the inputs of a packet switch while the await to be transmitted across the fabric. Such a buffering mechanism is called *input buffering,* or *input queueing.* A packet switch employing an input queueing mechanism is called an *input-queued* (IQ) switch.

An input-queued (IQ) switch architecture, depicted in figure 1-2, is built around a non-blocking bufferless switching fabric, interconnecting input to output lines. Examples of such fabric are crossbars, Clos networks, Benes networks and Batcher-Banyan networks [1][2][3][4]. We will focus our attention on crossbar (or crosspoint) based architectures, as they pave the way for utilizing off-the-shelf components in building the proposed systems. At any instant, the switching fabric can be configured to provide a set of parallel input/output connections, later called "matchings", such that each input (output) is connected to at most one output (input). The process of determining the dynamic configuration of the switch fabric is called *arbitration* or *scheduling*, and the mechanism governing the scheduling process is referred to as the *scheduling algorithm.* All switching fabric connections run at the

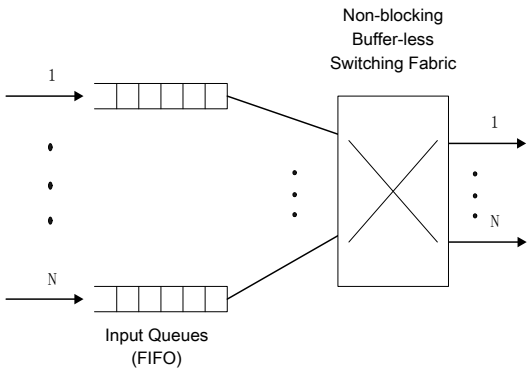

Figure 1-2: An N by N input-queued switch employing one FIFO queue for each input.port.

same speed as the external lines, (typically assumed to be equal for all inputs and outputs). As will later be shown in more details, this allows for the design of scalable, high-speed switching fabrics. Buffering and local processing are carried out for each external line at its respective *line card*, which provides termination of the line and interfacing to the fabric. In cases of contention, packets are stored in buffers at the input line cards.

In an IQ switch with FIFO queues, as shown in figure 1-2, only the Head-of-Line (HoL) packet in each FIFO queue is eligible for transmission. When output contention occurs, one of the HoL packets (out of all those destined for the same output port) can be transmitted across the switch, while other HoL packets are required to wait for service in subsequent time slots. It has been shown in [5], that an IQ switch with FIFO queues has a theoretical maximum throughput of only 58.6%, under the Bernoulli i.i.d., uniformly distributed traffic pattern. This throughput limitation is mainly caused by the fact that if a HoL packet is blocked as a result of output contention, other packets in the same FIFO queue, behind the HoL packet but destined to an idle output, can not be forwarded. This phenomenon is commonly referred to as head-of-line (HoL) blocking, as illustrated in figure 1-3

IQ switches quickly became an attractive architectural solution for the design of large-scale and high-capacity packet switches, mainly due to the creative work [6][7][8][9] which proved that the negative impact on the performance caused by HoL blocking [5] can be reduced, or completely eliminated, by adopting per-destination queueing, also called Virtual Output Queueing (VOQ), at the input stage. An input-queued switch employing the VoQ

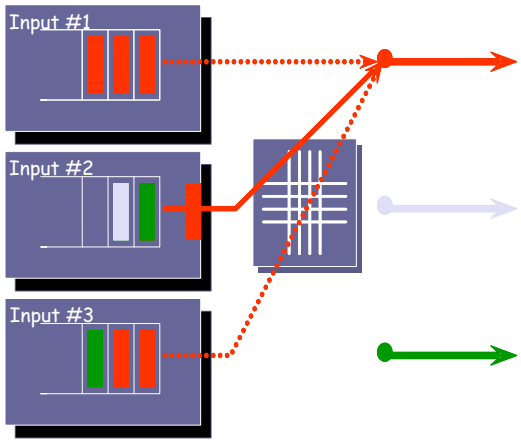

Figure 1-3: An illustration of the Head-of-Line (HoL) blocking phenomenon. Dotted lines represent blocked packets. Packets that are stored behind the block HoL packets are precluded from traversing the switch.

mechanism is depicted in figure 1-4. At each input, a separate FIFO queue is maintained for each output; hence, there are N^2 queues in total. Utilizing the VoQ mechanism can completely eliminate HoL blocking; however, this approach makes the process of determining the dynamic configuration of the switch fabric (i.e. the scheduling process) more complex since there are now N^2 head-of-line packets instead of just N. Therefore, the performance of the IQ switch is highly dependent on the scheduling algorithm it uses.

A key challenge pertaining to the design of IQ packet switches stems from the fact that access to the switching fabric must be coordinated by a scheduling algorithm, which operates on a knowledge of input queues' states. The scheduler is primarily in charge of dynamically determining the input-output matchings so as to maximize the performance of the switch. This means that control information must be exchanged among line cards, either through an additional data path, or through the switching fabric itself. Moreover, significant processing resources must be devoted to the scheduling algorithm, either at a centralized location or at the line cards in a distributed manner.

To achieve good scalability in terms of switch size and port data rates, it is essential to reduce the computational complexity of the scheduling algorithm. However, simpler algorithms typically exhibit reduced performance. Hence, a possible solution is to introduce

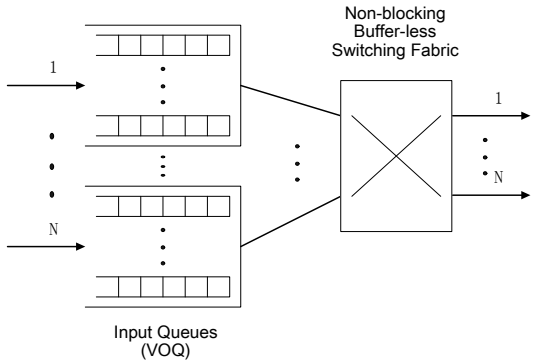

Figure 1-4: An N by N input-queued switch employing Virtual Output Queues (VOQs). Every input is associated with N VOQs, eachcorresponding to a different output.

a moderate speedup with respect to the data rate of input/output lines [10] in the switching fabric connections, as well as at the input and output memories. In this case, buffering is required at the output modules as well as at the inputs. Incorporating speedup inside the switch, necessitating input and output buffering, yields what is usually referred to as a Combined Input/Output Queued (CIOQ) switch. Obviously, when the speedup is such that the internal switching bandwidth equals the sum of the data rates on the input lines, input buffering is no longer needed as the architecture becomes a pure output-queued switch.

1.1.4 Output Queued Switches

An alternative classic solution to input queuing is output queuing, whereby the buffers are placed at the switch fabric outputs, see figure 1-5. In a pure OQ switch, all packets arriving at the same time slot, destined to the same output port, can be transferred across the switch fabric into the appropriate output FIFO within one time slot. However, only one packet can be transmitted onto the output line in one time slot; the remaining packets have to wait in the output FIFO for transmission during subsequent time slots. Therefore, with output buffering, unlike input queueing, arriving packets destined to one output do not block packets going to different outputs, which means that no HoL blocking can exist. Theoretically, output queuing offers ideal performance, since HoL blocking is avoided altogether, thus resolving the throughput limitations from which input queuing architectures

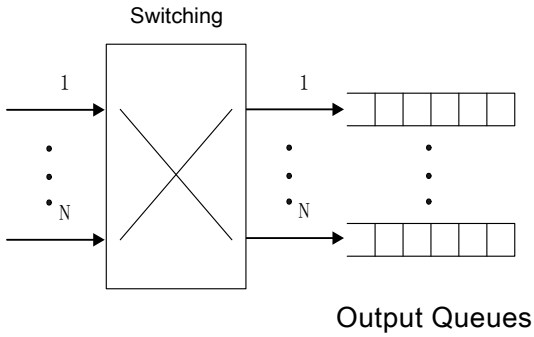

Figure 1-5: The basic architecture of an $N \times N$ output-queued switch.

suffer. Furthermore, contention is reduced to a minimum as only output contention may occur, which is unavoidable due to the statistical nature of real-life traffic. To that end, the average delay experienced by arriving packets is much lower than that observed in IQ switches. In fact, it can be shown that a non-blocking OQ switch, having queues of infinite size, offers the optimal performance in terms of both throughput and average delay. This type of switch is the only one that is truly *work conserving* under any traffic scenario, whereby work conservation is defined as follows:

Definition 5 *Working conservation: a packet switch is said to be work conserving if an output line is busy whenever there is at least one packet in the switch destined for it. If a packet switch is work conserving, its throughput is maximized, and thus the average packet delay is minimized.*

However, output queuing comes at a significant bandwidth cost. The bandwidth requirement on a single buffer is now proportional to both port speed and the number of ports. This is due to the fact that in one time slot N packets may arrive, destined to the same output, while only one can depart the switch. This suggests that at most N write operations and one read operation are needed per time slot. Thus, the per-output memory bandwidth requirement becomes $(N + 1)R$, where R denotes the port speed and N the number of output ports. To that end, the aggregate switch bandwidth equals $N(N + 1)R$, which is quadratic in the number of ports, rendering output queuing as an inherently less

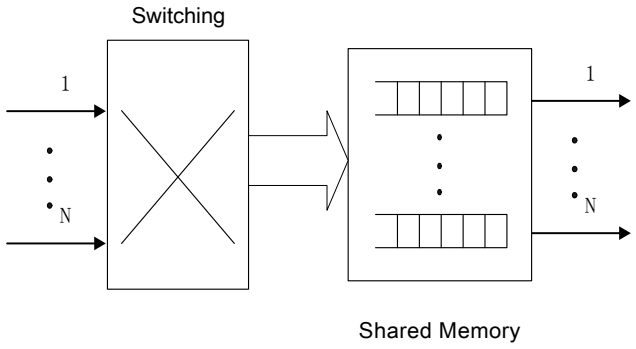

Figure 1-6: A shared memory output-queued switch

scalable solution to input queuing.

To address the aggregate switch bandwidth requirement, inherent to OQ switches, most practical implementations of OQ switches use a shared buffer instead of placing one separate memory for each output port, see figure 1-6. This approach significantly reduces the aggregate bandwidth requirement for dedicated output buffers. Clearly, shared memory OQ switches result in a memory bandwidth requirement of $2NR$, instead of $N(N+1)R$. However, although a shared memory approach has a significant advantage over the separate memory approach, the implementation of the shared memory OQ switch itself becomes a grand challenge when the capacity of switch is larger than the bandwidth of a single memory device. In order to obtain the desired switching capacity, several theoretical approaches have been proposed in the literature, including those that employ multiple memory devices in operating in parallel (interleaved memory banks), pipelined memory access schemes, and combinations thereof, which will be discussed in more detail later.

1.2 Theoretical Foundations for Stability Analysis

Along with the search for low complexity, highly scalable, well-performing switch architectures and scheduling algorithms, a relevant effort has recently been devoted to the identification of analytical methodologies that assess the performance achievable by IQ and CIOQ switch architectures. A complete set of theoretical results could indeed provide an

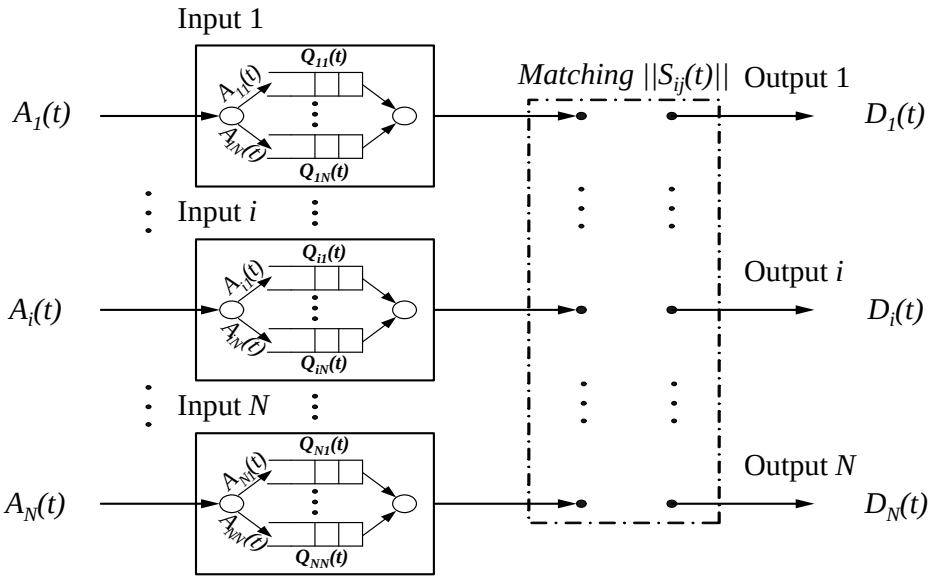

Figure 1-7: Modeling of an N by N IQ switch employing Virtual Output Queueing (VOQ).

important framework to drive applied researchers towards better performing solutions. To this end, the stochastic Lyapunov function methodology played a fundamental role, since it permitted to obtain most of the known theoretical results pertaining to the throughput of IQ and CIOQ switches. In this section we briefly introduce the stochastic Lyapunov function methodology, showing how it can be successfully applied to determine the stability region of a system of queues.

1.2.1 Modeling of Input-Queued Switches with Virtual Output Queueing

We begin with the notation of a generic queueing model pertaining to an IQ switch with virtual output queueing. Consider an N by N input-queued switch, shown in figure 1-7, connecting N input ports to N output ports. It is generally assumed that all lines have the same transmission rate, packets are of the same (fixed) size (also called *cells*) and time is slotted with slot sizes being equal to a single packet duration interval. At the beginning of each time slot, either zero or one cell arrives at each input. The arrival process at input i is represented by $A_i(t)$, $1 \leq i \leq N$. Each cell contains a label indicating its destination output j, $1 \leq j \leq N$. Cells arrive at input port i and destined to output port j are buffered at

the virtual output (FIFO) queue labeled VOQ_{ij}. The arrival process, with regard to each virtual output queue, is denote by $A_{ij}(t)$, $1 \leq i \leq N$, $1 \leq j \leq N$. Assuming the arrival process $A_{ij}(t)$, $1 \leq i \leq N$, $1 \leq j \leq N$ is stationary and ergodic, with mean rate denoted by $\lambda_{ij.}$, then $A(t) = [A_{11}(t), ..., A_{1N}(t), ..., A_{NN}(t)]^T$ is called *admissible* if no input or output is oversubscribed, i.e. if

$$\sum_{i=1}^{N} \lambda_{ij} < 1 \text{ and } \sum_{j=1}^{N} \lambda_{ij} < 1. \tag{1.1}$$

Let $Q(t) = [Q_{11}(t), ..., Q_{1N}(t), ..., Q_{NN}(t)]^T$ be the queue occupancy vector in which each component represents the number of cells currently buffered in its corresponding virtual output queue at time t. The virtual output queues are served in accordance with the policy dictated by the scheduling algorithm. The latter selects a set of conflict-free matches between the inputs and the outputs. The set of matchings is represented by a *matching matrix,* $\|S_{ij}(t)\|$, $1 \leq i \leq N$, $1 \leq j \leq N$, whose binary elements $S_{ij}(t) = 1$ iff input i is selected by the scheduler to connect to the output j, otherwise $S_{ij}(t) = 0$. It is often assumed that each input can only be connected to at most one output and visa versa. In this case, $S_{ij}(t)$ is doubly stochastic, i.e. each row and column have at most one element with a value of 1 and the rest are zeros.

Let $D(t) = [D_{11}(t), ..., D_{1N}(t), ..., D_{NN}(t)]^T$ be a vector denoting the departure process, for which the element $D_{ij}(t)$, represents the number of cells departed from virtual output queue VOQ_{ij}, during time slot t. Hence, the evolution of the queue occupancy can be written as

$$Q(t+1) = Q(t) + A(t) - D(t). \tag{1.2}$$

The IQ switch under study will be modeled by discrete time queues that, in turn, will be analyzed using Discrete Time Markov Chain (DTMC) models. In the following subsections, we will provide stability definitions of queueing systems and introduce the Lyapunov methodology.

1.2.2 Stability Definitions for Queueing Systems

We begin with basic definitions that will help guide the discussion throughout this dissertation.

Definition 6 *Under a stationary ergodic arrival process* $\{A(t)\}$*, satisfying the strong law of large numbers, i.e.*

$$\lim_{t\to\infty} \frac{\sum_{m=0}^{t-1} A(m)}{t} = \Lambda \qquad \textit{with probability 1}$$

a system of queues is rate stable, or is able to support 100% throughput, iff

$$\lim_{t\to\infty} \frac{Q(t)}{t} = 0 \quad \textit{with probability 1}$$

where $Q(t)$ *is the vector of queue sizes at time* t*.*

Definition 7 *Under a stationary ergodic arrival process* $\{A(t)\}$*, a system of queues is* weakly stable *if, for every* $\epsilon > 0$*, there exists* $B > 0$ *such that* $\lim_{n\to\infty} P\{||Q(t)|| > B\} < \epsilon$*,where* $P\{E\}$ *denotes the probability of event* E*.*

Definition 8 *Under a stationary ergodic arrival process* $\{A(t)\}$*, a system of queues is* strongly stable *if* $\lim_{t\to\infty} \sup E[||Q(t)||] < \infty$.

Any norm can be used in the two definitions above.

Strongly stability implies weak stability, and weak stability implies rate stability, or 100% throughput. However, the rate stability property allows the queue sizes to indefinitely grow at a sub-linear rate, while the weak stability property suggests that the servers in the system of queues can process the entire offered load, but the delay experienced by cells can be unbounded. Strong stability, on the other hand, provides bounds on the of average queue sizes on top of the properties offered by weak stability.

1.2.3 Lyapunov Methodology

We assume that the process governing the evolution of the virtual output queues is an irreducible DTMC, whose state vector at time t is$Y(t) = (Q(t), K(t))$, $Y(t) \in \mathbb{N}^M$, $Q(t) \in \mathbb{N}^N$, $K(t) \in \mathbb{N}^{N'}$, and $M = N + N'$. $Y(t)$ is the combination of vector $Q(t)$ and a vector $K(t)$ of N' integer parameters. Let H be the state space of the DTMC, obtained as a subset of the Cartesian product of the state space H_Q of $Q(t)$ and the state space H_K of

$K(t)$. >From Definition 7, we can immediately conclude that if all states $Y(t)$ are positive recurrent, the system of queues is weakly stable; however, the converse is generally not true, since queue sizes can remain finite even if the states of the DTMC are not positive recurrent due to instability in the sequence of parameter $\{K(t)\}$.

The following general criterion for the (weak) stability of systems that can be described with a DTMC is useful in the design of scheduling algorithms. This theorem is a straightforward extension of Foster's Criterion; see [11, 12, 13].

Theorem 1 *Given a system of queues whose evolution is described by a DTMC with state vector $Y(t) \in \mathbb{N}^M$, if a lower bounded function $L(Y(t))$, called Lyapunov function, $L : \mathbb{N}^M \to \mathbb{R}$ can be found such that $E[L(Y(t+1) \mid Y(t)] < \infty$, $\forall Y(t)$, and there exist $\epsilon \in \mathbb{R}^+$ and $B \in \mathbb{R}^+$ such that $\forall ||Y(t)|| > B$*

$$E[L(Y(t+1) - L(Y(t)) \mid Y(t)] < -\epsilon \tag{1.3}$$

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

If the state space of the DTMC, H, is a subset of the Cartesian product of the denumerable state space H_Q and a finite space H_K, the stability criterion can be slightly modified.

Corollary 1 *Given a system of queues whose evolution is described by a DTMC with state vector $Y(t) \in \mathbb{N}^M$, and whose state space H is a subset of the Cartesian product of a denumerable state space H_Q and a finite state space H_K, then, if a lower bounded function $L(Q(t))$, called Lyapunov function, $L : \mathbb{N}^M \to \mathbb{R}$ can be found such that $E[L(Q(t+1) \mid Y(t)] < \infty \quad \forall Y(t)$ and there exist $\epsilon \in \mathbb{R}^+$ and $B \in \mathbb{R}^+$ such that $\forall Y(t) : ||Q(t)|| > B$*

$$E[L(Q(t+1) - L(Q(t)) \mid Y(t)] < -\epsilon \tag{1.4}$$

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

This Corollary states that the system of queues is weakly stable iff all states of the DTMC are positive recurrent. Further, the criterion for strong stability is given as following.

Theorem 2 *Given a system of queues whose evolution is described by a DTMC with state vector* $Y(t) \in \mathbb{N}^M$*, and whose state space* H *is a subset of the Cartesian product of a denumerable state space* H_Q *and a finite state space* H_K*, then, if a lower bounded function* $L(Q(t))$*, called Lyapunov function,* $L : \mathbb{N}^M \to \mathbb{R}$ *can be found such that* $E[L(Q(t+1) \mid Y(t)] < \infty \quad \forall Y(t)$ *and there exist* $\epsilon \in \mathbb{R}^+$ *and* $B \in \mathbb{R}^+$ *such that* $\forall Y(t) : ||Q(t)|| > B$

$$E[L(Q(t+1) - L(Q(t)) \mid Y(t)] < -\epsilon ||Q(t)|| \tag{1.5}$$

then the system of queues is strongly stable.

Corollary 2 *Given a system of queues whose evolution is described by a DTMC with state vector* $Y(t) \in \mathbb{N}^M$*, and whose state space* H *is a subset of the Cartesian product of a denumerable state space* H_Q *and a finite state space* H_K*, then if there is a symmetric composite matrix* $\Lambda \in \mathbb{R}^{N \times N}$*, i.e.,* $Q(t)\Lambda Q^T(t) \geq 0$*, and and there exist* $\epsilon \in \mathbb{R}^+$ *and* $B \in \mathbb{R}^+$ *such that given the function* $L(Q(t)) = Q(t)\Lambda Q^T(t)$*,* $\forall Y(t) : ||Q(t)|| > B$*, it holds*

$$E[L(Q(t+1) - L(Q(t)) \mid Y(t)] < -\epsilon ||Q(t)|| \tag{1.6}$$

then the system of queues is strongly stable and all the polynomial moments of the queue size distribution are finite.

A set of quadratic Lyapunov functions is of particular interest, as will later be discussed. If the identity matrix I, is used as the symmetric composite matrix, we have the following corollary.

Corollary 3 *Given a system of queues whose evolution is described by a DTMC with state vector* $Y(t) \in \mathbb{N}^M$*, and whose state space* H *is a subset of the Cartesian product of a denumerable state space* H_Q *and a finite state space* H_K*, if there exist* $\epsilon \in \mathbb{R}^+$ *and* $B \in \mathbb{R}^+$ *such that given the function* $L(Q(t)) = Q(t)Q^T(t)$*,* $\forall Y(t) : ||Q(t)|| > B$*, it holds*

$$E[Q(t+1)Q^T(t+1) - Q(t)Q^T(t) \mid Y(t)] < -\epsilon ||Q(t)|| \tag{1.7}$$

then the system of queues is strongly stable and all the polynomial moments of the queue

size distribution are finite.

All theoretical assertions outlined above can be found in [14]. We shall mostly target strong stability inference wherever possible. Interestingly enough, although close-form analytical expressions pertaining to the performance of scheduling algorithms are often unattainable, proving the stability property of such algorithms is many times achievable.

1.3 Traffic Models for Performance Evaluation

Generally speaking, there are two main statistical characteristics of traffic pattern: the arrival process and the destination distribution. Arrival process may be categorized into two groups: bursty and non-bursty. Destination distribution corresponds to the probability of an arriving cell to be destined to each of the output ports. The most common and simple case is that of uniform distribution, whereby a packet has an identical likelihood of being destined to each output. Throughout this dissertation, we apply both bursty and non-bursty traffic arrival patterns. The following subsections will describe various traffic models implemented in our performance simulation environment.

1.3.1 Bernoulli Arrivals

The arrival process pertains to the nature of the correlation that may or may not exist between consecutive cell arrivals. 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, under the Bernoulli arrival process, the mean rate of the arrival process, or the average input offered load, denoted by λ, is equal to p.

1.3.2 Classical ON/OFF Model

It has been extensively shown in the literature that pragmatic network traffic tends to be correlated or "bursty" [15]. The classical bursty traffic model is based on a two-state Markov

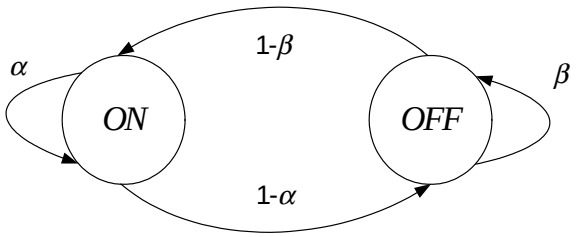

Figure 1-8: Classical bursty traffic model based on a two-state Markov chain.

chain, see figure 1-8.

Consider a discrete-time, two-state Markov chain generating arrivals modeled by an ON/OFF source which alternates between the ON and OFF states. An arrival is generated for each time slot that the Markov chain spends in the ON state. Let the parameters α and β denote the probabilities that the Markov chain remains in states ON and OFF, respectively. Using α and β the load and mean burst sizes may be directly obtained. The average input offered load can be expressed in terms of the transition probabilities as

$$\lambda = \frac{1-\beta}{2-(\alpha+\beta)}, \tag{1.8}$$

while the mean burst size is given by

$$MBS = \sum_{i=1}^{\infty} i\alpha^{i-1}(1-\alpha) = \frac{1}{1-\alpha} \tag{1.9}$$

Hence, from equation 1.8 and 1.9, we obtain the following relationship between the average input offered load and the mean burst size,

$$\lambda = \frac{MBS(1-\beta)}{MBS(1-\beta)+1} \Longrightarrow \lambda_{\max} = \frac{MBS}{MBS+1} \tag{1.10}$$

However, since at least a single OFF state separates two consecutive bursts, the maximal arrival rate is bounded by $MBS/(1+MBS)$ where MBS denotes the mean burst size. This bound is a significant limiting factor in the evaluation of high-speed packet switching fabrics. Moreover, in such evaluations there is a clear need for a heterogeneous model in

which both burst sizes and average rates can be flexibly determined [16] [17].

1.3.3 Maximal Throughput Model

To overcome the maximal traffic load constraint, we introduce the $ON/OFF/\Omega$ Markov-modulated arrival process, whereby in a transition from the ON state the process visits the Ω state while generating an additional packet to the same destination. It is only from the Ω state that the process can transition back to the OFF state. Consider a general case where an arriving packet can be destined to each of the N different destination.

As shown in figure 1-9, an arrival destined to the i^{th} output is generated for each time slot that the Markov chain spends in the ON_i state, $i = 1, 2, \ldots, N$, while no arrivals are noted for each when in the OFF state. Instead of direct transition from ON_i to OFF, a transition from ON_i to Ω_i is required. An arrival destined to the i^{th} output is still generated while the Markov chain is in the Ω_i state. To that end, there are a total of $2N + 1$ states in the proposed model, the transition probabilities of which are given by

	OFF	ON_1	$\cdots$	ON_i	$\cdots$	ON_N	Ω_1	$\cdots$	Ω_i	$\cdots$	Ω_N
OFF	β	δ_1	$\cdots$	δ_i	$\cdots$	δ_N	0	$\cdots$	0	$\cdots$	0
ON_1	0	α_1	$\cdots$	0	$\cdots$	0	$1-\alpha_1$	$\cdots$	0	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ON_i	0	0	$\cdots$	α_i	$\cdots$	0	0	$\cdots$	$1-\alpha_i$	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$P = ON_N$	0	0	$\cdots$	0	$\cdots$	α_N	0	$\cdots$	0	$\cdots$	$1-\alpha_N$
Ω_1	β_1	γ_{11}	$\cdots$	γ_{1i}	$\cdots$	γ_{1N}	0	$\vdots$	0	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Ω_i	β_i	γ_{i1}	$\cdots$	γ_{ii}	$\cdots$	γ_{iN}	0	$\cdots$	0	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Ω_N	β_N	γ_{N1}	$\cdots$	γ_{Ni}	$\cdots$	γ_{NN}	0	$\cdots$	0	$\cdots$	0

The limiting distribution of the model can be derived from the following set of equations:

$$\pi_{OFF} = \beta\pi_{OFF} + \sum_{i=1}^{N} \beta_i \pi_{\Omega_I} \tag{1.11}$$

$$\pi_{ON_i} = \delta_i \pi_{OFF} + \alpha_i \pi_{ON_i} + \sum_{k=1}^{N} \gamma_{ki} \pi_{\Omega_i} \tag{1.12}$$

$$\pi_{\Omega_i} = (1 - \alpha_i)\, \pi_{ON_i} \tag{1.13}$$

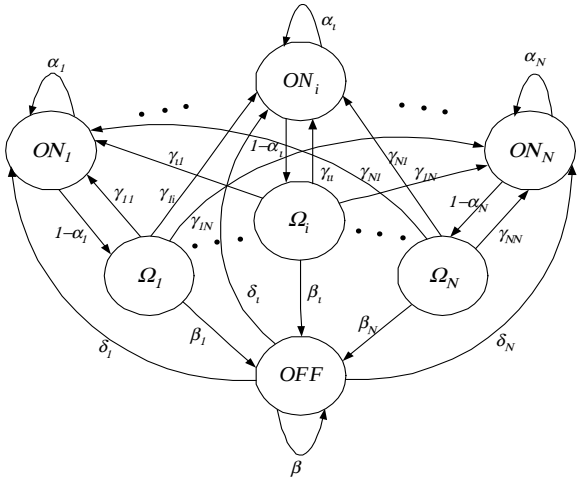

Figure 1-9: The proposed maximal throughput Markov modulated arrival process.

$$\pi_{OFF} + \sum_{k=1}^{N} \pi_{ON_i} + \sum_{k=1}^{N} \pi_{\Omega_i} = 1 \tag{1.14}$$

The mean arrival rate per output can be expressed as $\lambda_i = \pi_{ON_i} + \pi_{ON_i}$, while the mean burst size for output i is given by $MBS_i = 1 + \frac{1}{1-\alpha_i}$. For the case of uniform distribution and identical mean burst sizes ($\beta_i \equiv \beta$, $\lambda_i \equiv \lambda/N$), we have $\gamma_{ij} = \frac{1-\beta}{N-1}$, $\forall i \neq j$, and $\alpha_i = \alpha$. Solving under these assumptions yields

$$\lambda = \frac{(1-\beta)(2-\alpha)}{2-\alpha-\beta} \tag{1.15}$$

$$\lambda_i = \frac{(1-\beta)(2-\alpha)}{N(2-\alpha-\beta)} \tag{1.16}$$

$$MBS_i = 1 + \frac{1}{1-\alpha} \tag{1.17}$$

The key attribute of this model is that given any set of mean burst sizes, $MBS_i > 1$, and any traffic load distribution, λ_i, Markov chain in figure 1-9 can be constructed so as to yield the desired traffic generation engine. More importantly, the latter can achieve 100% traffic load. Without loss of generality, we can prove this statement by showing it holds for

the uniform case. We observe that

$$\alpha = \frac{MBS - 2}{MBS - 1} \tag{1.18}$$

, which means that given $MBS \geq 2$, we have $0 \leq \alpha \leq 1$. Moreover, observing that

$$\beta = \frac{(1-\lambda)(2-\alpha)}{2-\alpha-\lambda} \tag{1.19}$$

, we differentiate both sides with respect to λ, to obtain

$$\frac{d\beta}{d\lambda} = \frac{(\alpha-1)(2-\alpha)}{(2-\alpha-\lambda)^2} < 0 \tag{1.20}$$

The latter suggests that β is a decreasing function of λ, therefore, clearly, given $MBS \geq 2$ and $0 \leq \lambda \leq 1$, directly implies that $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$.

1.3.4 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

$$\sum_{j=1}^{N} p_{ij} = 1. \tag{1.21}$$

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, we have following relationships:

$$\lambda_{ij} = \lambda_i p_{ij} \tag{1.22}$$

$$\lambda_j = \sum_{i=1}^{N} \lambda_{ij} = \sum_{i=1}^{N} \lambda_i p_{ij}, \tag{1.23}$$

and the average output offered load is given by

$$\mu = \frac{1}{N} \sum_{j=1}^{N} \lambda_j. \tag{1.24}$$

Uniform Destination Distribution

The most commonly deployed destination distribution is uniform over all possible destinations. In uniform destination distribution, $p_{ij} = \frac{1}{N}$, for all i and j. Clearly, equations 1.22, 1.23, and 1.24, yield

$$\lambda_{ij} = \frac{\lambda_i}{N} \tag{1.25}$$

$$\lambda_j = \frac{1}{N} \sum_{i=1}^{N} \lambda_i \tag{1.26}$$

$$\mu = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{N} \sum_{i=1}^{N} \lambda_i = \frac{1}{N} \sum_{i=1}^{N} \lambda_i \tag{1.27}$$

The Zipf Destination Distribution

Real-life packet streams are distributed non-uniformly across the destinations, hence traffic tends to be focused on preferred, or popular, destinations. As means of evaluating the performance of the proposed architecture under non-uniform traffic conditions, we have selected a non-uniform destination distribution model named Zipf's law [18][19][20]. 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 [19]. The probability that an arriving cell is heading to destination k was thus modeled by

$$Zipf_m(k) = \lambda_m = k^{-m} \left(\sum_{j=0,1,..N} j^{-m} \right). \tag{1.28}$$

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

$$\sum_{j=1}^{N} p_{ij} = \sum_{j=1}^{N} \frac{j^{-r}}{\sum_{k=0}^{N} k^{-r}} = \frac{\sum_{j=1}^{N} j^{-r}}{\sum_{k=0}^{N} k^{-r}} = 1 \tag{1.29}$$

$$\lambda_{ij} = \frac{j^{-r}\lambda_i}{\sum_{k=0}^{N} k^{-r}} \tag{1.30}$$

$$\lambda_j = \frac{j^{-r}\sum_{i=1}^{N}\lambda_i}{\sum_{k=0}^{N} k^{-r}} \tag{1.31}$$

$$\mu = \frac{1}{N}\sum_{j=1}^{N}\lambda_j = \frac{1}{N}\frac{\sum_{j=1}^{N} j^{-r}\sum_{i=1}^{N}\lambda_i}{\sum_{k=0}^{N} k^{-r}} = \frac{1}{N}\sum_{i=1}^{N}\lambda_i \tag{1.32}$$

1.4 Motivation

This dissertation focuses on the design, analysis and performance evaluation of scalable architectures and algorithms for next-generation packet switching fabrics. In the context of IQ switches, due to the increase in link rates directly resulting in a decrease of packet duration times, packet-by-packet switching is no longer considered a pragmatic approach for designing scalable systems. To address this challenge, this dissertation advocates the utilization of frame-based algorithms that relax the timing constraints imposed on scheduling algorithms while retaining key performance characteristics. The algorithms are studied via theoretical stability analysis and evaluated by means of statistical simulations. In the context of OQ switching, an efficient memory management algorithm that alleviates some of the major limitations of OQ designs is presented and studied.

In an effort to introduce pragmatic solutions to the challenges associated with high-capacity packet switches, the focus of this work is to guarantee performance and scalability

while utilizing off-the-shelf components that can be easily combined with custom hardware circuitry. We conclude by showing that the developed architectures and algorithms provide solid cost-efficient foundations for supporting next-generation Internet switches and routers.

1.5 Dissertation Outline

Chapter 2 describes the primary features and limitations of mainstream existing scheduling algorithms, including maximum size matching and maximum weight matching. To overcome the inherent limitation raised, we introduce the notion of frame-based scheduling algorithms in chapter 3. The latter focuses in particular on two frame-based maximal weight matching algorithms, which employ queue length and waiting time of head-of-line packets as a priority metric, respectively. Both algorithms are shown to achieve 100% throughput under any admissible traffic.

In Chapter 4, we introduce a novel switch architecture that scale towards multi-Terabit/sec, offering high-performance while retaining desirable implementation simplicity attributes. The switch core can be put together using existing, off-the-shelf type products, rather than customized ASICs. We also investigate frame-based scheduling algorithms that can govern the proposed switch architecture. It is further shown that quality of service (QoS) can be supported by considering multiple classes of traffic. We prove key stability properties that clearly emphasize the advantages of the approach. A detailed performance evaluation is presented, which quantifies key performance metrics, under a range of diverse traffic scenarios. We also conduct a comparison, spanning performance, implementability and scalability attributes, between the solution proposed and existing ones. Finally, we investigated the above in the context of optical packet switching..

In chapter 5 we propose a novel memory management approach for parallel shared memory (PSM) switches that aims to emulate OQ switching. The proposed scheme utilizes a pipelined hardware architecture to achieve the throughput gain required. The algorithm parallelizes the packet placement process, thereby gaining execution speed at the expense of a fixed latency. In addition, we analyze the conditions under which the PSM switch can successfully emulate a First-Come-First-Served (FCFS) output-queued switch. Chapter 6

provides a summary of the contributions made in this dissertation.

Chapter 2

Literature Review

2.1 Scheduling Algorithms in Input Queued Switches

In this chapter we summarize a selection of scheduling algorithms presented in the literature for input-queued switches. In input queueing (IQ) architectures, arriving packets are buffered at the ingress ports, awaiting to traverse the switch, as directed by the scheduler. To that end, the following will provide an overview of scheduling algorithms proposed for VOQ-based input-queued switches.

2.1.1 Mathematical Modeling of the Matching Process

From a mathematical point of view, the scheduling algorithm in input queued switches selects a set of input-output index pairs with no conflicts, which implies that each input can be connected to at most one output, and visa versa. As shown in figure 2-1, the scheduling problem faced by crossbar-based, non-blocking input queued switches can be mapped to a bipartite graph-matching problem. Considering an N by N input queued switch employing VOQ, let $Q_{ij}(t)$ denote the VOQ_{ij} occupancies at time t. In each scheduling cycle (one time slot for slot by slot scheduling, and one frame for frame-based scheduling), we construct a graph $G = (V, E)$,that consists of a set of $2N$ vertices, among which N vertices for inputs and the other N vertices for outputs. The set of edges E has one edge for each non-empty VOQ. A matching M on this graph G is any subset of E such that no more than one edges in M has a common vertex.

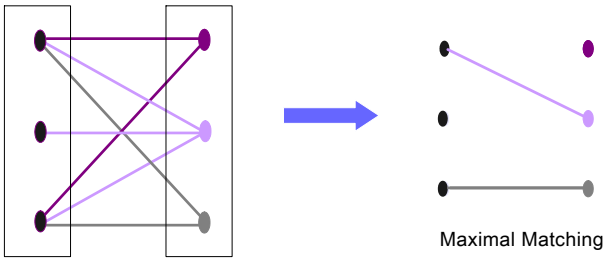

Figure 2-1: An example illustration of Maximal matching based on a Bipartite graph.

The interpretation of matching in this context is that for each edge that is part of the matching, connecting vertices i and j, one cell is allowed to be transferred from input i to output j. The definition of matching guarantees that only one packet per input and output needs to be transferred, thus satisfying the bandwidth restrictions imposed by the crossbar. A *maximum matching* is the largest size matching that can be made on a given graph, see figure 2-2. A *maximal matching* is a matching to which no further edges can be added without first removing an already matched edge, see figure 2-1. Therefore, any maximum match is also a maximal match, but not vice versa. On a given graph, there can be multiple maximum and multiple maximal matches.

The bipartite graph-matching problem can naturally be mapped to a request matrix $R = \|r_{ij}\|, 1 \leq i, j \leq N$, where $r_{ij} = 1$ if there is an edge between input i and output j in the graph, zero otherwise. Solving the matching problem now becomes finding a subset S of the set of requests such that at most one request can be selected per row and per column, i.e., mathematically, equivalent to following conditions, see equation 2.1.

$$\forall i, \quad \sum_{(i,j)\epsilon S} r_{ij} \leq 1 \tag{2.1}$$

$$\forall j, \quad \sum_{(i,j)\epsilon S} r_{ij} \leq 1 \tag{2.2}$$

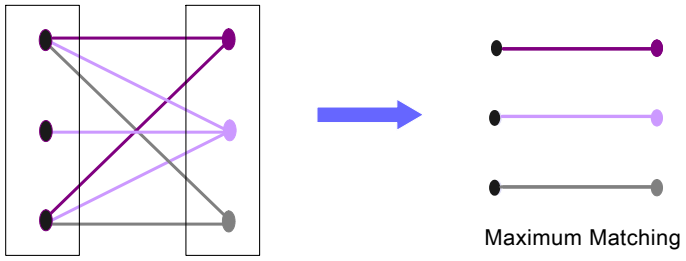

Figure 2-2: An example illustration of Maximum matching based on a Bipartitie graph.

Clearly, for the examples shown in figure 2-2 and figure 2-1, we have

$$R = \begin{Vmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{Vmatrix}, \; S_{Maximum} = \begin{Vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{Vmatrix}, \; S_{Maximal} = \begin{Vmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{Vmatrix} \tag{2.3}$$

2.1.2 Maximum Size Matching

A maximum size matching M on graph G described above is one that maximizes the number of edges in M. The fastest known solution to the maximum size matching problem executes in $O(N^{2.5})$ time [21] [22]. In [23] a proof is given that maximum size matchings can achieve 100% throughput under uniform i.i.d. Bernoulli traffic. However, it has been shown in [24] that this no longer holds under non-uniform traffic, and moreover, that maximum size matching can lead to instability and unfairness under admissible traffic, whereas under inadmissible traffic it can even lead to starvation (scenarios in which some queues are not serviced for very long periods of time). Therefore, maximum size matching must be considered unsuitable.

2.1.3 Maximum Weight Matching

In assigning weights $w_{ij}(t)$ to the edges of G we can perform maximum weight matching on G, by finding a matching M that maximizes the total weight $W(t) = \sum_{(i,j)\epsilon M} w_{ij}(t)$.

Maximum size matching is clearly a specific case of maximum weight matching with all weights $w_{ij}(t) = 1$. The weights are a function of time t, as the state of the system variables, from which the weights are computed, change over time.

Tassiulas and Ephremides [13] [25] showed, initially in the context of multihop packet radio networks, that maximum throughput is achieved if a maximum weight policy is adopted using the queue occupancies as weights, i.e., $w_{ij}(t) = Q_{ij}(t)$. McKeown et al. [24] arrived at the same conclusion with their longest queue first (LQF) algorithm, which is proven to be stable under the condition of admissible, i.i.d. arrivals. However, LQF was shown to lead to starvation of short queues. Mekkittikul and McKeown proposed the oldest cell first (OCF) algorithm in [26], which uses the age of the HoL packets as weights, thus guaranteeing that no queue will remain unserved indefinitely, because the weight of unserved HoL packets will increase until they eventually are served. This algorithm was also shown to be stable under all i.i.d. arrivals.

The most efficient known algorithm that solves the maximum weight optimization problem requires $O(N^3 \log(N))$ execution time [27], which is realistically infeasible for large port density switches. This renders many of the proposed algorithms not suitable for practical implementations. Tassiulas proposed [25] a randomized algorithm of linear complexity that employs an incremental updating rule, using the heuristic that the matching obtained for the previous cycle is likely to be a good matching also for the current cycle, as the state of the input queues changes only slightly. Basically, in each cycle a random matching (distributed such that there is a non-zero probability that the optimum, maximum weight matching is chosen) is selected. If the new matching leads to a higher weight than the previous one, the new one is selected, otherwise the old one is kept. This approach was shown to achieve maximum throughput [25].

McKeown proposed practically implementable heuristic, iterative versions of LQF and OCF, called i-LQF and i-OCF. These algorithms operate very similar to i-SLIP, with the difference that instead of just one-bit requests, each request carries a weight, equal to the queue length in the case of i-LQF and to the age of the HoL packet in the case of i-OCF.

Mekkittikul and McKeown [28] designed the longest port first (LPF) algorithm to overcome the complexity issues raised by LQF. The weights are equal to the port occupancy,

which is defined as the sum of the queue occupancies of the corresponding input and output, i.e. $w_{ij}(t) = \sum_k Q_{ik}(t) + \sum_l Q_{lj}(t)$. The algorithm was shown to be stable under all i.i.d. arrivals, and has a complexity of $O(N^{2.5})$, which is lower than LQF because LPF is based on maximum size matching with a preference to choosing the maximum size matching with the largest weight (which could be less than the actual maximum weight matching). A heuristic, iterative version of LPF called i-LPF was proposed for implementation using high-speed hardware.

Finally, the i.i.d. arrivals assumption required by [24] [29] [13] was lifted by Dai and Prabhakar, who proved using fluid model techniques that an IQ switch using any maximum weight matching algorithm achieves a throughput of 100%, as long as no input or output is oversubscribed and the input arrivals satisfy the strong law of large numbers, which are very mild conditions that most real traffic patterns satisfy [30].

2.1.4 Heuristic Matchings

As both maximum size and weight algorithms are too complex and therefore unsuitable for application in high-speed packet switches, research has focused on heuristic matching algorithms that execute in linear time. In practice, these algorithm will converge on a maximal instead of maximum match, a maximal match being defined as a match to which no edges can be added without first removing previously matched edges.

The most straightforward way to obtain a maximal match is by sequential matching. This algorithm works as follows. Initially, all inputs and outputs are unmatched. Every input sends a request to each output for which it has at least one packet. The algorithm iterates over all outputs; in each iteration one output gives a grant to one of the inputs it received a request from, excluding inputs that have been already been previously matched.

To ensure fairness and avoid starvation, each output provides grants in a round robin fashion: it keeps a pointer to the input it has granted to last and will give the next grant to the input that appears next in the round robin. Similarly, the sequence in which the outputs are matched is determined in a round-robin fashion. The main drawback of this algorithm is that although it obtains a maximal match, it requires $O(N^2)$ execution time. To improve and achieve linear execution time, parallel matching algorithms have been de-

veloped. In [31] an algorithm for input-buffered switches is described that can be classified as a sequential matching algorithm, although it does not explicitly employ VOQ. In [32][33] RPA is presented, a sequential matching algorithm that operates as described above, but with an extension to support multiple traffic classes.

Parallel Iterative Matching

The sequential algorithms described above require $O(N)$ iterations to arrive at a maximal match. A second class of heuristic matching algorithms allows multiple input and output pairs to be matched simultaneously, thus converging faster on a maximal match. One such parallel matching algorithm is the Parallel Iterative Matching (PIM) algorithm invented by DEC [6]. Its operation is as follows: Initially, all inputs and outputs are unmatched. In each iteration of the algorithm, the following request, grant, and accept steps are executed:

1. Request: each unmatched input i sends a request to every output j which is non-empty;

2. Grant: if an unmatched output receives any requests, it randomly, in a uniform manner,selects one that will receive the grant;

3. Accept: if an input receives any grants, it randomly, in a uniform manner, selects one to accept.

It has been shown that the algorithm completes in $O(\log N)$ iterations on average [6]. However, PIM will never starve a connection like maximum size matching does, but it has its disadvantages: With a single iteration, throughput is limited to just 63%, and it may also cause unfairness among connections [23]. Furthermore, implementing random selections at high speeds is difficult and costly.

WPIM

WPIM (Weighted PIM) [34] is a variant of PIM that adds support for providing bandwidth guarantees between individual input-output pairs in the presence of ill-behaved flows. Regular PIM always provides equal bandwidth to all input-output pairs because of its statistical nature. If an output is not oversubscribed, that is, the sum of arriving flows does

not exceed the output's bandwidth, this is not a problem, but when one input exceeds its allocated share of bandwidth, all others suffer as well. To prevent this, WPIM introduces bandwidth-allocation coefficients that are used by each output to decide individually whether a particular input has exceeded its allocated share of bandwidth—those that have will not receive a grant as long as other inputs have not received their allocated share. Thus, WPIM enables bandwidth provisioning and flow isolation in an input-queued switch using VOQ. Like regular PIM, it completes in $O(\log N)$ iterations.

iSLIP

To make PIM faster, more efficient and more fair, iSLIP was introduced in [23]. iSLIP operates as follows:

1. Request: each unmatched input sends a request to each output it has at least one packet for;

2. Grant: each output that has received at least one request selects one request to grant by means of its round-robin arbiter: it chooses the input that appears next in the round robin, starting from the input currently being pointed to. The pointer is advanced to one beyond the input just granted if and only if the grant is accepted in step 3;

3. Accept: similarly, each input that has received at least one grant selects one grant to accept by means of its round-robin arbiter: it chooses the output that appears next in the round robin, starting from the output currently being pointed to. The pointer is advanced (modulo N) to one beyond the output just accepted.

Under uniform i.i.d. Bernoulli arrivals, a maximum throughput of close to 100% is achieved by SLIP, which removes the throughput limitation of PIM. Further, iSLIP, which executes mutiple iterations of SLIP in each time slot, improves the speed of SLIP. For a detailed analysis of performance and properties of SLIP and i-SLIP, the reader is referred to [23] and [35].

2.2 Architectures for High Capacity Routers

2.2.1 Challenges in Single-Path Switch Architecture

Most of the work appearing in the literature defines a flow as the set of all packets arriving at the same input ports and destined to the same output ports. Currently, in the most common router architecture, packets from the same flow generally take the same path through the router. We shall refer to such architecture as single-path architecture. Due to the fact that all packets from the same flow take the same path, those single-path architectures may face serious challenges when applied to the design of high capacity routers. They either require a complex centralized scheduler, which is very undesirable due to the computational complexity involved, or do not provide 100% throughput guarantees. Moreover, they are sensitive to single-points of failure.

2.2.2 PPS, PSM and DSM

Single-path architecture is not a good choice when considering large routers. Recently, much attention has been drawn to introducing parallelism, that is, in contrary to single-path, packets from the same flow are allowed to take different paths through the router. Examples of such multi-path switches include the Parallel Packet Switch (PPS) [36], the Parallel Shared Memory (PSM) switch and Distributed Shared Memory (DSM) switch [37].

The PPS is comprised of multiple identical lower speed packet switches operating independently and in parallel. An incoming stream of packets is distributed, packet by packet, by a demultiplexer across the slower packet switches, then recombined by a multiplexer at the output. It has been shown theoretically in [36] that with a speedup factor of at least 2, a PPS can emulate a FCFS-OQ switch. Furthermore, a PPS which achieves high capacity by placing multiple packet switches in parallel, rather than in series as is common in multistage switch designs. Hence, each packet that passes through the system encounters only a single stage of buffering; In addition, and of greatest interest, the packet buffer memories in the center-stage switches operate slower than the line rate. However, the main fascinating attribute of PPS is that it requires a centralized scheduler, which is totally impractical due to large computational and communication complexity involved. Similar to PPS architecture,

PSM and DSM suffer the same problem. They either require a centralized scheduler with large computational and communication complexity, or use simple distributed scheduling algorithms but do not provide 100% throughput guarantees.

2.2.3 The Load-Balanced Switch

A load-balanced switch consists of a single stage of buffering, sandwiched by two identical stages of switching. This architecture is based on the concept of load-balancing packets uniformly inside the router prior to forwarding them to their destinations, an idea first introduced by Valiant et al. [38]. Unlike the most common current router architectures, a basic load-balanced router does not need any scheduling since the sequence of crossbar permutations is predetermined. In addition, C.-S. Chang et al. showed in [39] that a load-balanced router does not require any scheduler and that it can guarantee 100% throughput for a broad class of traffic. A basic load-balanced router simplifies the linecards, and especially the buffering stage. It only uses one stage of buffering, while typical centralized-scheduler switches run the switch fabric faster than the line rate and require two stages of buffering. All these advantages make the load-balanced switch very appealing. However, even though the load-balanced router guarantees 100% throughput for a large class of traffic patterns, it is still vulnerable to some pathological traffic patterns. Also, the architecture suffers from several other problems, such as packet reordering and the need for frequent switch fabric reconfigurations, which is undesirable when building high-capacity optical switch fabrics.

Chapter 3

Frame-based Maximal Weight Matching

Input-queued packet switching architectures are commonly utilized in Internet routers as they offer pragmatic scalability while requiring moderate memory bandwidth. A switch with a speedup of 1 is said to allow at most one packet from each input to traverse the crossbar during one time slot. If a switch has a speedup of s, where $s \in \{1, ..., N\}$, it is said to issue s scheduling decisions, and correspondingly s transmissions of packets from input buffers to output ports, during a single time slot. When $s > 1$ buffers are required at the output ports as well. Such architectures are commonly referred to as combined input-and-output-queued (CIOQ)[30]. Naturally, the need for speedup greater than 1 introduces stringent hardware constrains, as it necessitates high memory bandwidth resources. Many scheduling algorithms have been proposed in recent years, with a common goal of which to offer scalability (with respect to port densities and link rates) as well as high-performance. In the context of performance, a fundamental requirement from any scheduling algorithm is stability. Stated coarsely, a switch is said to be stable if all its queues are bounded and, hence, never backlog indefinitely. Once a switch has been proven to be stable, its performance can be evaluated by means of simulations with reasonable confidence.

It has been shown that for a broad class of traffic arrival patterns, all *maximal matching* algorithms yield a stable switch of any size with a speedup of 2 [30][40]. This stability prop-

erty holds while delivering a throughput of up to 100%. One subset of maximal matching algorithms is *maximal weight matching* (MWM) algorithms, in which greedy convergence to a maximal aggregate matching weight is obtained. The increase in link rates directly causes a decrease in packet duration times to a point where packet-by-packet switching is no longer considered a pragmatic approach. To address this point, we introduce the frame-based maximal weight matching (FMWM) algorithm, in which scheduling decisions are issued in accordance with the MWM algorithm, however they are kept unchanged for a duration of k consecutive time slots. By reconfiguring the crossbar switch once every several time slots we relax the timing constraints imposed on the scheduling algorithm. An immediate key question pertains to what are the speedup requirements under which such an algorithm yields a stable switch. Furthermore, given that stability is guaranteed, a complementing question would be: what are the implications of increased switching intervals on the performance? This chapter aims to address these important questions.

3.1 Frame-based LQF

3.1.1 Stability of the FMWM/LQF Algorithm

Consider a CIOQ switch with N ports, as illustrated in figure 3-1. Let $Q_{ij}(t)$ denote the VOQ size at input i holding packets destined to output j at time t. We define the corresponding random arrival process, $A_{ij}(t) \in \{0, 1\}$, with a mean rate of packet arrivals from input i to output j, $E[A_{ij}(t)] = \lambda_{ij} \leq 1$. We consider the simple FMWM which consists on an iterative process whereby in each iteration the maximal weight is found and a match is registered between its associated input-output pair. Each time a match is generated, the respective input and output are removed from contending during the following iterations. Assuming the weight matrix is not completely null, the number of iterations ranges from 1 to N. The configuration of the crossbar, which is the outcome of the FMWM algorithm, can be represented by the permutation matrix $S(t) = \{S_{ij}(t)\}$, where $Sij(t) = 1$ if input i is matched to output j at time t, otherwise $Sij(t) = 0$.

Without loss of generality, let us assume that at time t we have explicit knowledge of $Q_{ij}(t)$. We thus assign a weight value, equal to the queue length, to each VOQ based on

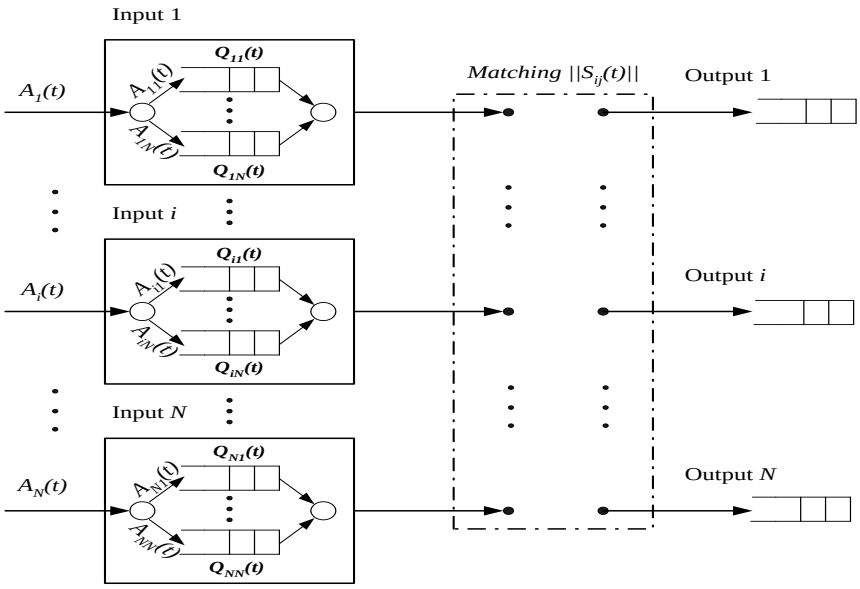

Figure 3-1: An N-port combined-input-output-queued (CIOQ) switch architecture.

which the scheduler establishes the maximal weight matching for k consecutive time slots. As a result, at time $t+k$, we have a new matching/scheduling matrix $S_{ij}(t+k)$. As depicted in figure 3-2, the matching matrix remains unchanged during the following k time slots. It should be noted that although we restrict our attention to a weighting scheme which reflects only on the queue occupancies, a broader definition of queue weights may be applied.

In the interest of incorporating such reconfiguration delays, let us define κ_R as the mean duration (in bit times) required for the system to reconfigure itself. Letting p denotes the number of payload bits in each fixed-size packet, the total number of bits in a frame is $\kappa_R + k \cdot p$.An obvious way of overcoming the inefficiency introduced by the reconfiguration dead-times, is to speedup the transfer rate of packets from the VOQs to the output buffers. Correspondingly, a speedup factor of 2 suggests that up to two packets can be transferred from each VOQ to an output buffer during a single time slot. The output buffers transmit the queued packets at the link rate, in the order they arrive. It should be noted that while the notion of speedup commonly implies multiple scheduling cycles during each time slot, here we simply assume an increase in the internal packet transfer rate such that no increase of computational effort is required.

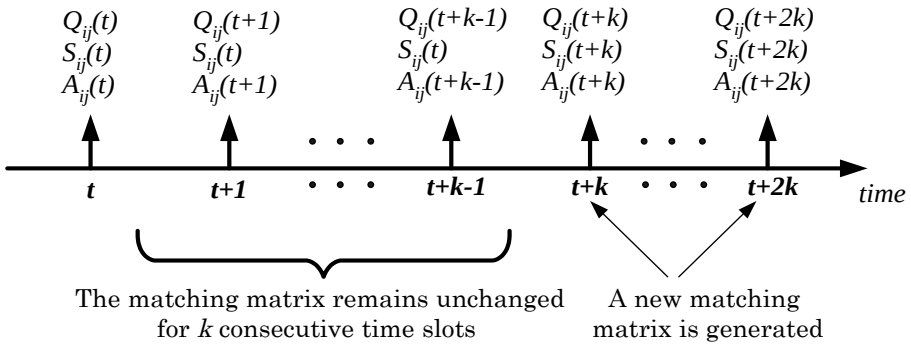

Figure 3-2: Buffer dynamics under the FMWM scheduling algorithm.

Definition 9 *The weight produced by the FMWM algorithm at time t is given by*

$$W^{FMWM}(t) = \left\langle Q(t), S^{FMWM}(t) \right\rangle = \sum_{i,j} Q_{ij}(t) S_{ij}^{FMWM}(t) \tag{3.1}$$

where $S_{ij}^{FMWM}(t)$ denotes the matching configurations established by the algorithm at time t.

Theorem 3 *An input-queued switch with an average reconfiguration delay of κ_R, running the FMWM scheduling algorithm with a speedup of $2\left(1+\frac{\kappa_R}{kp}\right)$ is stable under admissible traffic for any finite frame size k.*

Proof. We will derive the sufficient speedup value, η, as follows. Since at most k packets may arrive during k time slots, when applying the FMWM algorithm the following inequality holds

$$Q_{ij}(t+k-1) - Q_{ij}(t) \leq k, \tag{3.2}$$

from which we can write

$$Q_{ij}(t+k-1) - Q_{ij}(t) \leq \sum_{m=0}^{k-1} A_{ij}(t+m) - \eta k S_{ij}(t), \tag{3.3}$$

for $Q_{ij}(t) \geq \eta k$. The term $\eta k S_{ij}(t)$ expresses the ηk consecutive transmissions that may

occur during a frame interval. Next, we construct a discrete-time quadratic Lyapunov function, $L(t)$, such that $L(t) = \langle Q_t, Q_t \rangle = \sum_{i,j} Q_{ij}^2(t)$ [41][28].In order to prove the algorithm yields a stable queueing system, we would like to show that beyond a given threshold of maximal weight there is a negative drift in the state (queue occupancies) of the system. As an expression of a k time slot lag, we can write

$$\begin{aligned} &L(t+k-1) - L(t) = \\ &\sum_{ij} \left(Q_{ij}(t+k-1) - Q_{ij}(t)\right) \left(Q_{ij}(t+k-1) + Q_{ij}(t)\right). \end{aligned} \tag{3.4}$$

By partitioning the above into the case of $Q_{ij}(t) < \eta k$ and $Q_{ij}(t) \geq \eta k$, we deduct the following

$$\begin{aligned} &E\left[L(t+k-1) - L(t) | Q(t)\right] \leq \\ &\sum_{ij} \left(\sum_{m=0}^{k-1} A_{ij}(t+m) - \eta k S_{ij}(t) \right) E\left[2Q_{ij}(t) + k\right] \cdot \\ &\Pr\left(Q_{ij}(t) \geq \eta k\right) + \sum_{ij} k(2\eta k + k) \cdot \Pr\left(Q_{ij}(t) < \eta k\right) \\ &\leq \sum_{ij} \left(\sum_{m=0}^{k-1} A_{ij}(t+m) - \eta k S_{ij}(t) \right) E\left[2Q_{ij}(t) + k\right] \\ &+ \sum_{ij} k(2k\eta + k) \\ &\leq \sum_{ij} 2E\left[Q_{ij}(t)\right] \left(k\lambda_{ij} - \eta k S_{ij}(t)\right) \\ &+ \sum_{ij} k^2 + k(2\eta k + k) \\ &\leq 2k\left[\langle \Lambda, Q_t \rangle - \eta \langle S, Q_t \rangle\right] + 2k^2 N^2 (1+\eta) \end{aligned} \tag{3.5}$$

where $\Lambda = \|\lambda_{ij}\|$ denotes the admissible arrival rate matrix, which is doubly stochastic. We further observe that for all $S_{ij}(t) \neq 0$,

$$2S_{ij}(t) = 2 > \sum_{l=1}^{N} \left(\lambda_{il} + \lambda_{lj}\right) \tag{3.6}$$

which stems from the fact that FMWM guarantees that $S_{ij}(t) \neq 0$ always points to the largest value on raw i and column j,respectively. Since FMWM removes raw i and column j after each iteration, (3.6) holds for all iterations. Hence, since Q_t is referred to identically on both sides of the inequality in (3.6), we conclude that $\langle \Lambda, Q_t \rangle < 2 \langle S, Q_t \rangle$. However, the latter does not take into account the additional speedup needed to overcome the reconfiguration delays. If κ_R denotes the portion of the frame that is "wasted" on reconfiguration, it implies that by speeding up the transmission of actual payload bits by $1+\frac{\kappa_R}{kp}$, the dead-times can be compensated for. Since the algorithm speedup requirement of 2 is independent of the additional speedup needed to compensate for reconfiguration dead-times, we conclude that for $\eta \geq 2\left(1+\frac{\kappa_R}{kp}\right)$ we have $\langle \Lambda, Q_t \rangle < \eta \langle S, Q_t \rangle = \eta W^{FMWM}(t)$. This suggests that there exists a value $\bar{\alpha} < 1$ for which $\langle \Lambda, Q_t \rangle < \bar{\alpha}\eta W^{FMWM}(t)$. Applying the latter to (3.6) yields

$$\begin{aligned} & E\left[L(t+k-1) - L(t) | Q(t)\right] \\ & \leq 2k(\bar{\alpha}-1)W^{FMWM}(t) + 2k^2N^2(1+\eta). \end{aligned} \tag{3.7}$$

Thus, for all $W^{FMWM}(t) > \frac{kN^2(1+\eta)}{(1-\bar{\alpha})}$, we obtain $E\left[L(t+k-1) - L(t) | Q(t)\right] < 0$, which concludes the stability proof. ■

3.1.2 Simulation Results

In order to evaluate the performance of the FMWM algorithm under different traffic conditions and interval durations, five simulation sets were carried out. In all simulations a 6-port switch was considered with a speedup of $2\left(1+\frac{\kappa_R}{kp}\right)$. In the first simulation set, the arrival process was Bernoulli i.i.d. with uniformly distributed destination distribution. Figure 3-3 depicts the mean delay when employing FMWM with different switching frame sizes (k). As can be intuitively appreciated, the longer the frame the larger the mean delay, which stems from the fact that during many switching intervals less than k consecutive packets are being transmitted. Moreover, it is noted that larger frame sizes exhibit faster delay growth (steeper slope). This is because once a matching matrix is generated, the unmatched VOQs will not transmit any cells during the follow k time slots, yet still buffer newly arriving cells

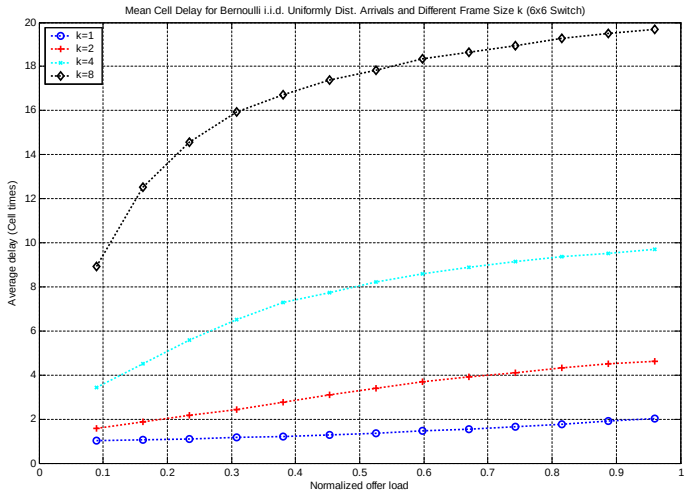

Figure 3-3: Mean delay for Bernoulli i.i.d. uniformly distributed arrivals for different frame sizes.

(which contribute to the average waiting time).

The second and third simulations were targeted at examining the packet delay distributions and VOQ size distributions, respectively. Figure 3-4 shows the packet delay distributions for the FMWM/LQF under uniform Bernoulli i.i.d. arrival process. The input offered load is 0.9, while the frame size is 8. Figure 3-5 depicts the probability distribution of the virtual output queue size.

The fourth set of simulations examined the impact of bursty traffic on the performance of the FMWM algorithm, Uniformly distributed bursty traffic with identical mean burst size (MBS) of 4 cells was applied. Figure 3-6 depicts the resulting mean delay for different frame sizes ($k = 1, 2, 4, 8$) but same mean burst size ($MBS = 8$). An interesting observation here is that the difference in delay between the first three frame sizes ($k = 1, 2$ and 4) is small relative to the delay increase shown for $k = 8$. This can be explained by the fact that bursts that are larger or equal to the frame size result in fully utilized transmission intervals, while packets in bursts that are larger than the frame size experience higher average delay.

The fifth set of simulations was targeted at examining the impact of the mean burst size on the delay performance. Once again, a 6 port switch was considered whereby bursts are

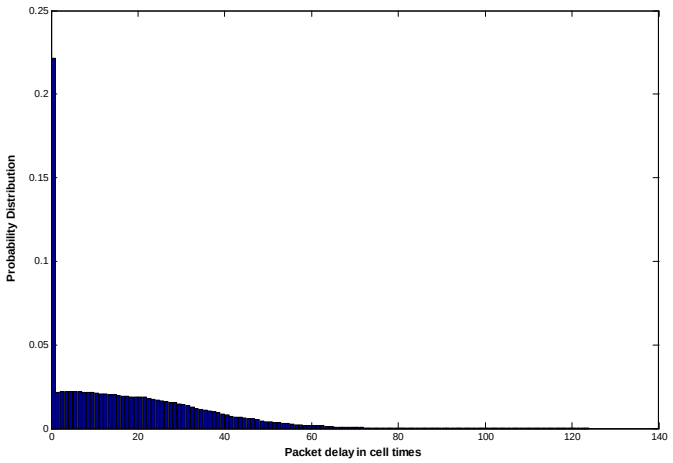

Figure 3-4: Packet delay distributions for FMWM/LQF under uniform Bernoulli i.i.d. arrival process. The input offer load is 0.9, while the frame size equals 8.

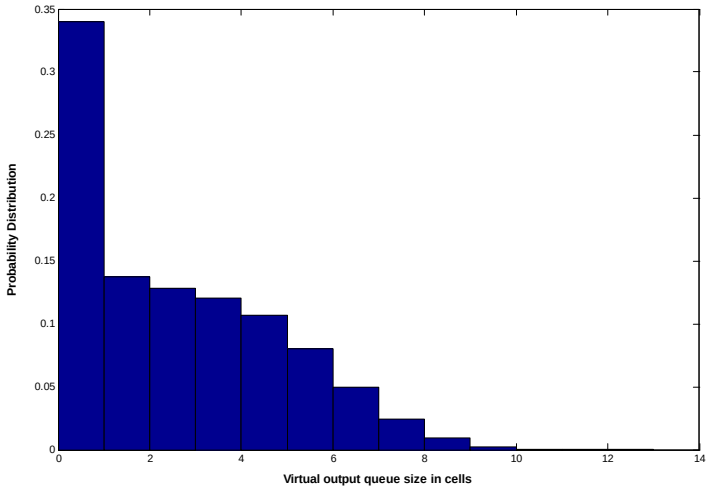

Figure 3-5: Virtual output queue size distributions for FMWM/LQF under uniform Bernoulli i.i.d. arrival process. The input offer load is 0.9, while the frame size equals 8.

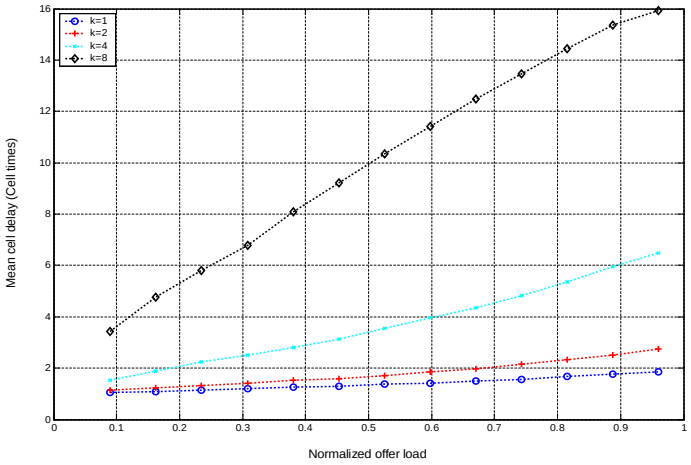

Figure 3-6: Mean cell delay as a function of the offered load for uniformly distributed arrivals with mean burst size of 8 cells and different frame size k

uniformly distributed across the outputs. Figure 3-7 shows the mean delay as a function of the mean burst sizes for a fixed frame size of 8 cells. A clear difference in relative performance is observed between the lower load and higher load regions. The reason for this is that at lower loads queues storing packets are served often and with little delay. To that end, as long as the bursty is smaller or equal to the switching interval, the burst size has little impact on the mean delay. However, at the higher loads, queue inter-service times (cycles) are longer, thus explaining the greater differentiation between the various MBS curves.

The last two simulations are intended to examine the packet delay distributions and VOQ size distributions under bursty arrival process, with input offered load of 0.9, while both frame size and mean burst size equal to 8. Figure 3-8 shows the packet delay distributions while figure 3-9 shows the probability distribution of the virtual output queue size.

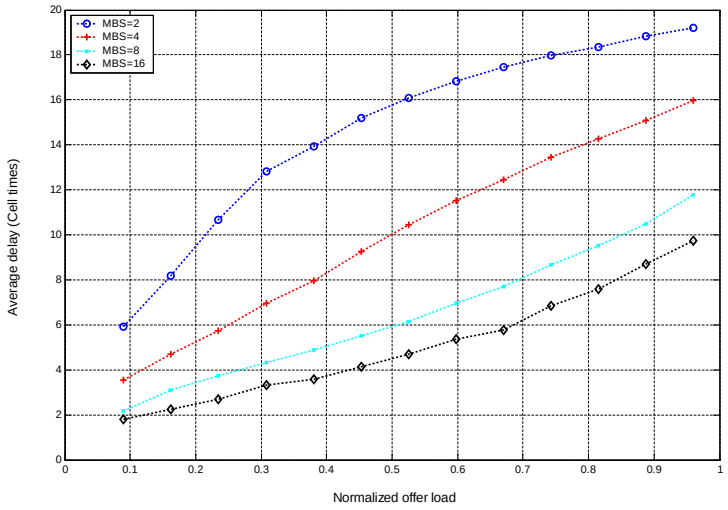

Figure 3-7: Mean cell delay as a function of the mean burst size (MBS) for a frame size of 8 cells.

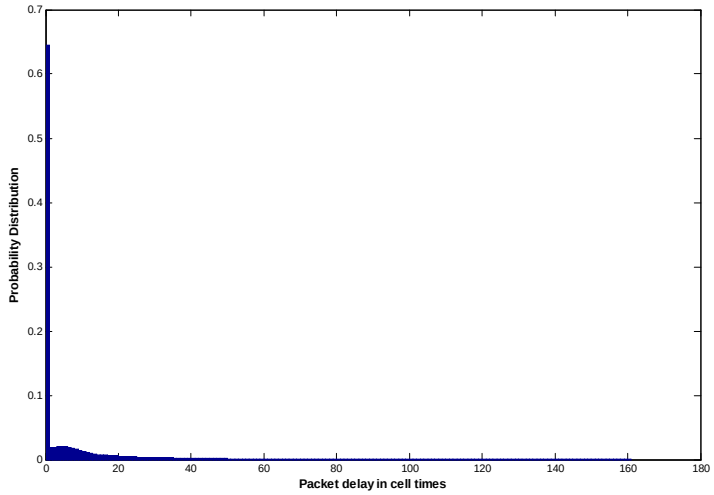

Figure 3-8: Packet delay distributions for FMWM/LQF under a bursty arrival process. The input offered load is 0.9, while the frame size and mean burst size both equal 8.

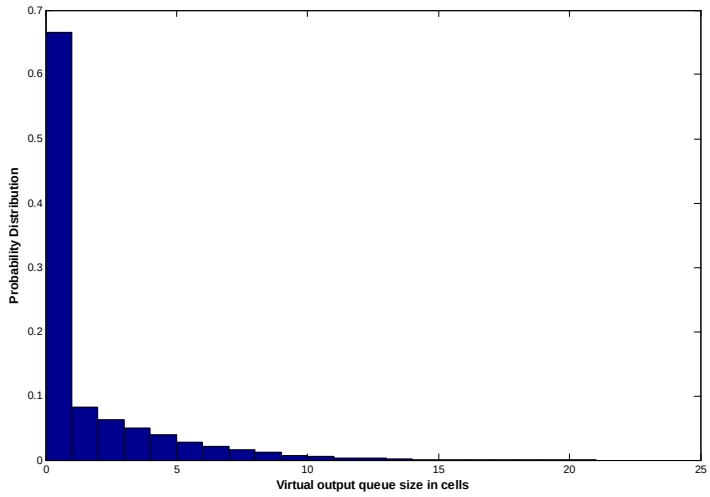

Figure 3-9: Virtual output queue size distributions for FMWM/LQF under bursty arrivals. The input offered load is 0.9, while the frame size equals 8 and the mean burst size is also 8.

3.2 Frame-based OCF

3.2.1 FMWM/OCF with a Single Class of Service

We next analyze the FMWM algorithm whereby the queue priority metric is the age of its head-of-line packet. First, we refer to the following definitions.

Definition 10 *Let the waiting time vector of the HoL cells at time t be*

$$\Omega(t) = \Big[T_{11}(t), \quad \cdots, \quad T_{1N}(t), \quad \cdots, \quad T_{NN}(t)\Big]^T \tag{3.8}$$

where $T_{ij}(t)$ is the waiting time of the HoL cell in queue (i,j) at time t.

Definition 11 *Let $\tau_{ij}^{(m)}(t)$ denote the inter-arrival time between two consecutive cells, m and $m+1$, both of which are stored in queue (i,j), and correspondingly, let $\tau_{ij}(t) = \max\{\tau_{ij}^{(m)}(t),\, m = 1, 2, ..., Q_{ij}(t)\}$.*

Definition 12 *The transfer speedup, $\eta \geq 1$, shall be referred to as the maximal number of cells that each input queue can transfer to the output ports during a single time slot.*

It is important to emphasize that while in the literature the notion of speedup entails the requirement for multiple *scheduling* decisions to be issued in each time slot, the concept of transfer speedup is computationally much more relaxed as it *does not require multiple scheduling decisions.*

Theorem 4 *A CIOQ switch employing the FMWM/OCF scheduling algorithm with a transfer speedup of 2 is stable under admissible traffic for any frame size k.*

Proof. Our proof will focus on deriving a speedup sufficiency condition under which the system will be stable. The buffer dynamics under the FMWM algorithm dictate that for $Q_{ij}(t) > \eta k$

$$T_{ij}(t+k) = T_{ij}(t) + k - S_{ij}(t) \sum_{m=1}^{\eta k} \tau_{ij}^{(m)}(t) \tag{3.9}$$

while for $Q_{ij}(t) \leq \eta k$

$$\begin{aligned} & T_{ij}(t+k) \leq T_{ij}(t) + k \leq Q_{ij}(t) \cdot \tau_{ij}(t) + k \\ \leq \quad & \eta k \tau_{ij}(t) + k \end{aligned} \tag{3.10}$$

Next, we construct a discrete-time quadratic Lyapunov function, $L(t)$, such that $L(t) = \sum_{ij} \lambda_{ij} T_{ij}^2(t)$. Consequently, we have

$$L(t+k) - L(t) = \sum_{ij} \lambda_{ij} T_{ij}^2(t+k) - \sum_{ij} \lambda_{ij} T_{ij}^2(t)$$

By partitioning the above into the case of $Q_{ij}(t) \leq \eta k$ and $Q_{ij}(t) > \eta k$, we deduct that if $Q_{ij}(t) > \eta k$

$$\begin{aligned} & T_{ij}^2(t+k) - T_{ij}^2(t) \\ \leq \quad & k^2 + 2 \left[k - S_{ij}(t) \sum_{m=1}^{\eta k} \tau_{ij}^{(m)}(t) \right] T_{ij}(t) \end{aligned} \tag{3.11}$$

and if $Q_{ij}(t) \leq \eta k$

$$\begin{aligned} & T_{ij}^2(t+k) - T_{ij}^2(t) \leq k^2 + 2kT_{ij}(t) \\ \leq \quad & k^2 + 2\eta k^2 \tau_{ij}(t) \end{aligned} \tag{3.12}$$

Therefore, by applying equations (3.11) and (3.12), we observe that the drift in the Lyapunov function is

$$\begin{aligned} & E\left[L(t+k) - L(t) | \Omega(t)\right] \\ \leq \quad & \sum_{ij} \lambda_{ij} \left(k^2 + 2 \left[k - S_{ij}(t) \sum_{m=1}^{\eta k} E\left[\tau_{ij}^{(m)}(t) \right] \right] T_{ij}(t) \right) \\ & + \sum_{ij} \lambda_{ij} E\left[k^2 + 2\eta k^2 \tau_{ij}(t) \right] \\ \leq \quad & 2 \sum_{ij} \lambda_{ij} \left[k - \frac{\eta k S_{ij}(t)}{\lambda_{ij}} \right] T_{ij}(t) + C \end{aligned} \tag{3.13}$$

where $C = 2\sum_{ij} \lambda_{ij} k^2 + 2\sum_{ij} \eta k^2$. Hence, we conclude that

$$\begin{aligned} & E\left[L(t+k) - L(t) | \Omega(t)\right] \\ \leq \quad & 2 \sum_{ij} \lambda_{ij} \left[k - \frac{\eta k S_{ij}(t)}{\lambda_{ij}} \right] T_{ij}(t) + C \\ \leq \quad & 2k \left(\langle \Lambda, \Omega(t) \rangle - \eta \langle S, \Omega(t) \rangle \right) + C \end{aligned} \tag{3.14}$$

where $\Lambda = \|\lambda_{ij}\|$ denotes the admissible arrival rate matrix, which is doubly stochastic. We further note that for all $S_{ij}(t) \neq 0$,

$$2S_{ij}(t) = 2 > \sum_{l=1}^{N} (\lambda_{il} + \lambda_{lj}) \tag{3.15}$$

which stems from the fact that the FMWM algorithm guarantees that $S_{ij}(t) \neq 0$ always points to the largest value on row i and column j, respectively. Since FMWM removes row i and column j after each iteration, (3.15) holds for all iterations. Given that $\Omega(t)$ is referred to identically on both sides of the inequality in (3.15), we conclude that $\langle \Lambda, \Omega(t) \rangle < 2 \langle S, \Omega(t) \rangle$. This implies that for $\eta \geq 2$ we have $\langle \Lambda, \Omega(t) \rangle < \eta \langle S, \Omega(t) \rangle = \eta W^{FMWM}(t)$,

which suggests that there exists a value $\bar{\alpha} < 1$ for which $\langle \Lambda, \Omega_t \rangle < \bar{\alpha}\eta W^{FMWM}(t)$. Applying the latter to (3.15) yields

$$\begin{aligned} & E\left[L(t+k) - L(t)|\Omega(t)\right] \\ & \leq 2k(\bar{\alpha} - 1)W^{FMWM}(t) + C \end{aligned} \tag{3.16}$$

Thus, for all $W^{FMWM}(t) > \frac{C}{2k(1-\bar{\alpha})}$, we infer that $E\left[L(t+k) - L(t)|\Omega(t)\right] < 0$, which concludes the stability proof. ■

3.2.2 FMWM/OCF with Multiple Classes of Service

Next, we extend the analysis to show that when multiple classes of service are employed at each input port, the same stability property holds. The underlying assumption is that the notion of virtual output queueing is expanded such that there are now a set of L queues residing in input i destined to output j, where by each of the L queues has a different priority level associated with it. The priority is reflected by a weighted scheme such that the age of the oldest cell in the queue is multiplied by a per-class coefficient to yield the weight used by the scheduler to issue the matching decisions. To clarify this point, we shall refer to the following definitions:

Definition 13 *With weighted priorities, an arrival process is said to be admissible iff*

$$\sum_{il} \lambda_{ijl} \leq 1, \quad \sum_{jl} \lambda_{ijl} \leq 1 \tag{3.17}$$

where λ_{ijl} denotes the normalized offered load from input i to output j, in class $l \in [1, L]$.

Definition 14 *Let the scheduling weight vector at time t be defined as*

$$\Omega(t) = \left[W_{111}, \ \cdots, \ W_{1N1}, \ \cdots, \ W_{NNL}\right]^T \tag{3.18}$$

where $W_{ijl}(t) = C_l T_{ijl}(t)$, $i, j = [1, N]$, $l = [1, L]$, and C_l is the (positive) weight coefficient of class l.

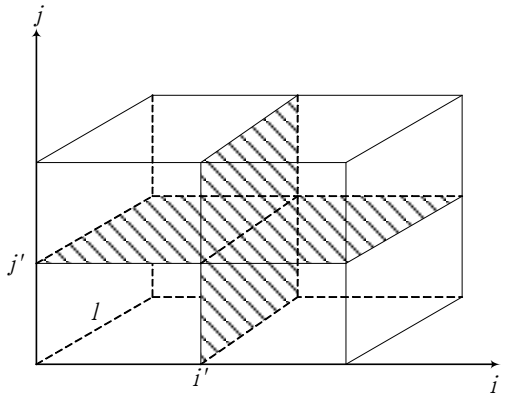

Figure 3-10: Upon generating an input-output match, two plains are removed from the weight tensor, thereby reducing the set of contending nodes in subsequent iterations of the algorithm.

Definition 15 *Let $S(t) = \{S_{ijl}(t)\}$ denote the permutation matrix, where $S_{ijl}(t) = 1$ if class l in input i is matched to output j at time t, otherwise $S_{ijl}(t) = 0$. Note that the maximal number of possible matches remains N (i.e. $\sum_{i,j,l} S_{ijl} \leq N$), since at most N inputs can be matched to N outputs.*

Definition 16 *Let $\tau_{ijl}^{(m)}(t)$ denote the inter-arrival time between two consecutive cells, m and $m+1$, both of which are stored in VOQ $Q_{ijl}(t)$, and let $\tau_{ijl}(t) = \max\{\tau_{ijl}^{(m)}(t),\ m = 1, 2, ..., Q_{ijl}(t)\}$.*

The configuration of the crossbar, which is the outcome of the FMWM algorithm, can be represented by the permutation tensor of order three, as illustrated in figure 3-10. Without loss of generality, let us assume that at time t we have explicit knowledge of $W_{ijl}(t)$. The FMWM algorithm guarantees that $S_{ijl}(t) = 1$ always points to the largest value on row i, column j and class of service l, respectively. Following each iteration, the two plains which contained the largest weight are removed, as illustrated in figure 3-10 (in this example plains $\{i', j, l\}$ and $\{i, j', l\}$ are removed). This leads to the following theorem:

Theorem 5 *A CIOQ switch employing the FMWM/OCF scheduling algorithm with a transfer speedup of 2 and multiple classes of service, realized using positive per-class weight coefficients, is stable under admissible traffic for any frame size k.*

Proof. Similarly to (3.11), when $Q_{ijl}(t) > \eta k$ we may write

$$T_{ijl}(t+k) = T_{ijl}(t) + k - S_{ijl}(t) \sum_{m=1}^{\eta k} \tau_{ijl}^{(m)}(t) \tag{3.19}$$

and if $Q_{ijl}(t) \leq \eta k$

$$\begin{aligned} T_{ijl}(t+k) &\leq T_{ijl}(t) + k \leq Q_{ijl}(t) \cdot \tau_{ijl}(t) + k \\ &\leq \eta k \tau_{ijl}(t) + k \end{aligned} \tag{3.20}$$

Applying (3.19) and (3.20) with weighted per-classes OCF, the following holds for $Q_{ijl}(t) > \eta k$

$$W_{ijl}(t+k) = W_{ijl}(t) + C_l k - C_l S_{ijl}(t) \sum_{m=1}^{\eta k} \tau_{ijl}^{(m)}(t) \tag{3.21}$$

otherwise,

$$\begin{aligned} & W_{ijl}(t+k) \leq W_{ijl}(t) + C_l k \\ \leq\ & C_l Q_{ijl}(t) \cdot \tau_{ijl}(t) + C_l k \\ \leq\ & C_l \eta k \tau_{ijl}(t) + C_l k \end{aligned} \tag{3.22}$$

Next, we construct a discrete-time quadratic Lyapunov function, $L(t)$, in the form $L(t) = \sum_{ijl} \frac{\lambda_{ijl}}{C_l} W_{ijl}^2(t)$. Consequently, we obtain

$$L(t+k) - L(t) = \sum_{ijl} \frac{\lambda_{ijl}}{C_l} \left[W_{ij}^2(t+k) - W_{ij}^2(t) \right] \tag{3.23}$$

From (3.21) and (3.22), we have the following inequality

$$
\begin{aligned}
& L(t+k) - L(t) \qquad (3.24)\\
\leq\ & \sum_{ijl} \frac{\lambda_{ijl}}{C_l} \left[C_l k - C_l S_{ijl}(t) \sum_{m=1}^{\eta k} \tau_{ijl}^{(m)}(t) \right]^2 + \\
& \sum_{ijl} 2\frac{\lambda_{ijl}}{C_l} \left[C_l k - C_l S_{ijl}(t) \sum_{m=1}^{\eta k} \tau_{ijl}^{(m)}(t) \right] W_{ijl}(t) + \\
& \sum_{ijl} \frac{\lambda_{ijl}}{C_l} C_l^2 \left[k^2 + 2\eta k^2 \tau_{ijl}(t) \right]
\end{aligned}
$$

therefore,

$$
\begin{aligned}
& E\left[L(t+k) - L(t) | \Omega(t) \right] \qquad (3.25)\\
\leq\ & \sum_{ijl} 2\lambda_{ijl} \left[k - S_{ijl}(t) \sum_{m=1}^{\eta k} E\left[\tau_{ijl}^{(m)}(t) \right] \right] W_{ijl}(t) + \\
& \sum_{ijl} \frac{\lambda_{ijl}}{C_l} \left[2C_l^2 k^2 + 2C_l^2 \eta k^2 \tau_{ijl}(t) \right] \\
\leq\ & \sum_{ijl} 2\lambda_{ijl} \left[k - S_{ijl}(t) \frac{\eta k}{\lambda_{ijl}} \right] W_{ijl}(t) + C \\
\leq\ & 2k \sum_{ijl} \left[\lambda_{ijl} W_{ijl}(t) - \eta W_{ijl}(t) S_{ijl}(t) \right] + C
\end{aligned}
$$

where $C = 2\sum_{ijl} C_l \left[\lambda_{ijl} + \eta \right] k^2$. We further observe that for all $S_{ijl}(t) \neq 0$,

$$
2S_{ijl}(t) = 2 > \sum_{il} \lambda_{ijl} + \sum_{jl} \lambda_{ijl} \qquad (3.26)
$$

Since FMWM removes two plains following each iteration, (3.26) holds for all iterations, and thus we conclude that $\sum_{ijl} \lambda_{ijl} W_{ijl}(t) < \sum_{ijl} 2W_{ijl}(t) S_{ijl}(t)$, suggesting that for $\eta \geq 2$ there exists a value $\bar{\alpha} < 1$ for which $\sum_{ijl} \lambda_{ijl} W_{ijl}(t) < \bar{\alpha}\eta \sum_{ijl} W_{ijl}(t) S_{ijl}(t)$, Applying the

latter to (3.25) yields

$$E\left[L(t+k) - L(t)|\Omega(t)\right] \leq 2k(\bar{\alpha}-1)W^{FMWM}(t) + C \tag{3.27}$$

Thus, for all $W^{FMWM}(t) > \frac{C}{2k(1-\bar{\alpha})}$, we infer that $E\left[L(t+k) - L(t)|\Omega(t)\right] < 0$, which concludes the stability proof. ■

3.2.3 Simulation Results

In order to evaluate the performance of the FMWM algorithm under different traffic conditions and interval durations, two simulation sets were conducted. In all simulations a 6-port switch was considered with a transfer speedup of 2. In the first set of simulations, the arrival process was Bernoulli i.i.d. with uniformly distributed destination distribution. Figure 3-11 shows the mean delay when employing FMWM/OCF with different switching frame sizes (k). As can be intuitively appreciated, the longer the frame the larger the mean delay, which stems from the fact that during many switching intervals less than k consecutive cells are being transmitted. Moreover, it is noted that larger frame sizes exhibit faster delay growth (steeper slope). This can be explained by the fact that once a matching matrix is generated, the unmatched VOQs will not transmit any cells during k time slots, yet they continue to buffer newly arriving cells (which contribute to the increase in the average queue waiting time).

Figure 3-12 shows the packet delay distributions for the FMWM/OCF under uniform Bernoulli i.i.d. arrival process. The input offered load is 0.9, while the frame size equals 8. Figure 3-13 shows the probability distribution of the virtual output queue size.

The second set of simulations was targeted at examining the impact of multiple classes of service on the delay performance. Linear priority coefficients were used, such that $W_{ijl}(t) = l \times T_{ijl}(t)$. Once again, a 6 port switch with 3 classes of service was examined. The arrival processes was the same as before, while traffic was uniformly distributed across the classes of service. The frame size was set to 8 cells. As can be observed in figure 3-14, the higher the priority level the lower the mean delay experienced by arriving packets, a result of the fact

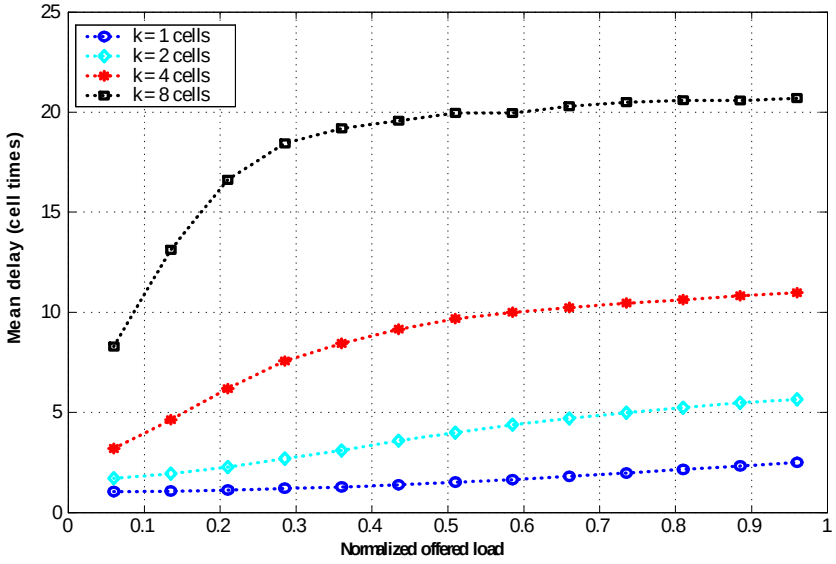

Figure 3-11: MWMF/OCF mean queueing delay as a function of the frame size (k). Arrival process is uniformly distributed Bernoulli i.i.d.

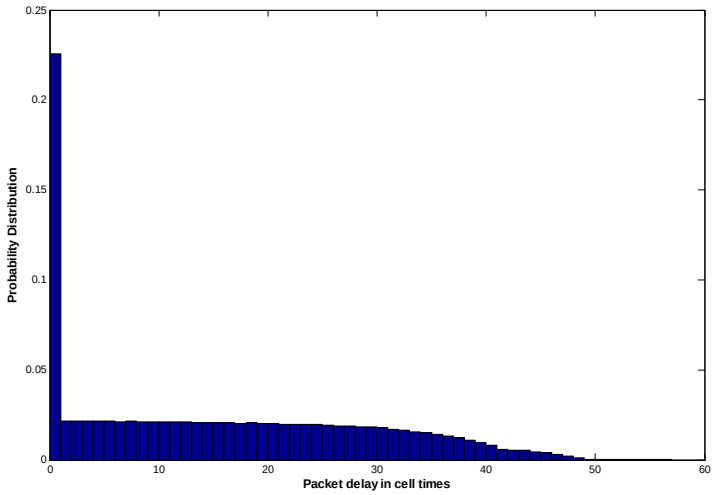

Figure 3-12: Packet delay distributions for FMWM/OCF under uniform Bernoulli i.i.d. arrival process. The input offer load is 0.9, while the frame size equals 8.

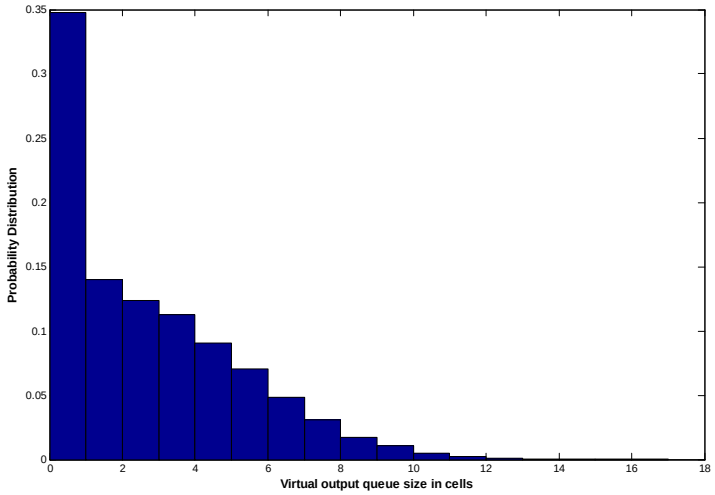

Figure 3-13: Virtual output queue size distributions for FMWM/OCF under uniform Bernoulli i.i.d. arrival process. The input offered load is 0.9, while the frame size equals 8.

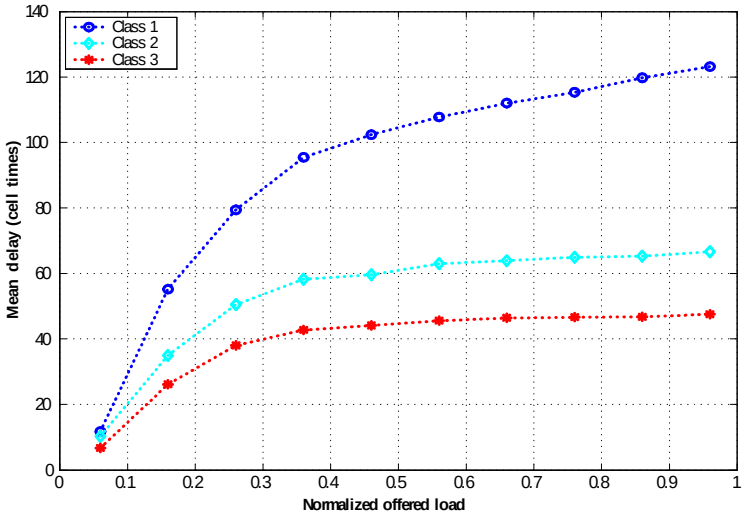

Figure 3-14: FMWM/OCF mean delay for multiple classes of service. Classes of service are differentiated via linear priority coefficients.

that the scheduler tends to select the higher weights during each iteration. The disparity between the different classes of service can be controlled by tuning the priority coefficients. The theory presented in this chapter clearly states that as long as the class coefficients are non-zero, the system is guaranteed to be stable.

Chapter 4

A Scalable Multi-Crosspoint Based Packet Switching Architecture

4.1 Overview

We next introduce a novel scalable packet switching architecture which is straightforward to implementation, as it employs a group of memoryless passive crosspoint switches. The architecture is a CIOQ switch whereby an $N \times N$ switch is partitioned into G identical and independent switching groups, each hosting a pool of smaller crosspoint switches. The motivation to employ such an architecture is to benefit from the idea of partitioning one (large) crossbar into several small crosspoints [42], so as to facilitate scalability and reduce the timing requirements from the scheduling algorithm. The approach is characterized by offering 100% throughput guarantee for a broad class of traffic patterns, no need for consideration of packet reordering and infrequent switch fabric reconfiguration. Moreover, we extend our analysis to address the scenario of multiple classes of service in which a weighted queue-priority scheme is employed.

4.2 Switch Architecture

The proposed switch architecture is based on that which was first introduced in [42]. Consider an $N \times N$ switch as shown in figure 4-1, whereby N input modules are equally parti-

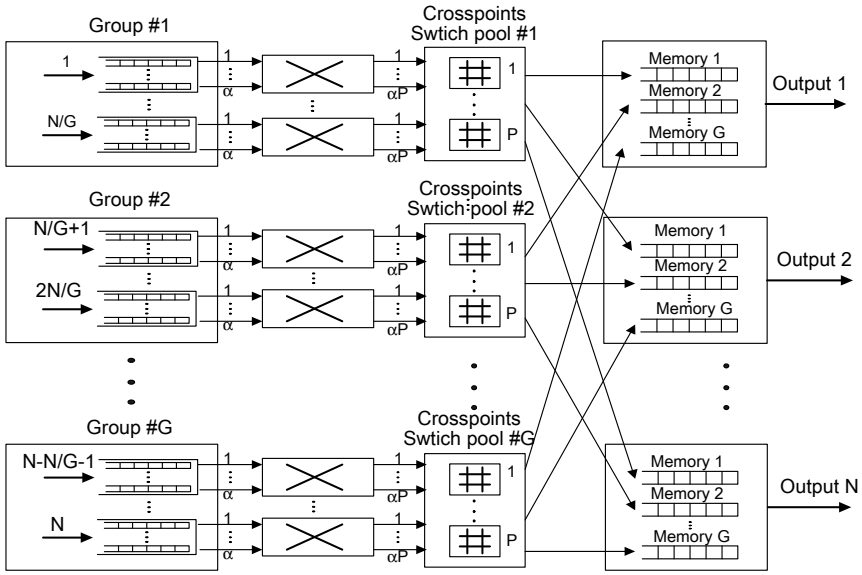

Figure 4-1: Proposed switch architecture in which an N-by-N CIOQ switch is equally partioned into G independent groups employing multiple passive crosspoints.

tioned into G groups, each of which is independently connected to all N outputs via a pool of non-blocking crosspoint switches. It is assumed that each of the crosspoint switches is small, so as to facilitate practical scalable switch implementations.

Throughout this dissertation we shall refer to a *flow* as the collection of all packets with the same input and output index values. We further let a *group flow* denote the set of all packets from a given group destined to a unique output. All packets belonging to the same *group flow* will be buffered in the same memory module at their destination output. For example, all packets from group #1 that are destined to output N will be buffered in memory 1 of output N. Hence, multiple memory modules must be maintained at each output to hold packets from different group flows. Clearly, the number of memory modules maintained at each output is G, since for each output, there can be at most G different *group flows*, each of which corresponds to one group. We shall let all packets from the same *flow* traverse through the same path, i.e., we only consider single-path switching, discarding multi-path scenarios since these generally incur packet reordering problems.

The core switching fabric comprises two stages of passive crosspoint switches. The first

connects the ingress ports to the rest of the fabric, hosting a pool of $\alpha \times \alpha K$ crosspoint switches per group, whereby P denotes the number of crosspoint switches in the second stage and α is the maximal number of output that can be matched to a single input. Note that $\alpha = 1$ represents the common case in which each input can be matched to at most one output. If $\alpha > 1$ then a transfer speedup is required. Each of the crosspoint switches in the second stage has $\alpha\frac{N}{G}$ inputs and $\frac{N}{P}$ outputs. By placing an $\alpha \times \alpha P$ switch between the crosspoints pool and each input port, maximal traffic throughput is guaranteed, as will be elaborated on in the following section. From a crosspoint optimization perspective, since the highest number of inputs or outputs on any crosspoint device in the system is a key scalability metric, we note that the maximal port count on any is given by $max\{\alpha P, \alpha\frac{N}{G}\}$. For example, if one wishes to design a 512-port switch (i.e. $N = 512$), where $\alpha = 2$, $G = 16$, $P = 8$, then $max\{\alpha P, \alpha\frac{N}{G}\} = 64$,suggesting that the switch can be realized using existing off-the-shelf crosspoint devices.

4.3 Stability Analysis Under Admissible Traffic

4.3.1 FMWM/OCF for a Single Class of Service

In this section we derive the necessary conditions for a switch supporting a single class of service to be stable. With reference to figure 4-1, let $Q_{ij}(t)$ denote the VOQ size at input i holding packets destined to output j at time t. Let us also define the corresponding random arrival process, $A_{ij}(t) \in \{0, 1\}$, with a mean (normalized) rate of packet arrivals from input i to output j, $E[A_{ij}(t)] = \lambda_{ij} \leq 1$. Since the switch is equally partitioned into G independent groups and each group supports its own non-blocking paths, stability analysis focused on any particular group can be easily extended to all other groups with minor modifications, as will later be described.

We consider a simple FMWM/OCF scheduling algorithm pertaining to the g^{th} group. The algorithm consists of an iterative process whereby during each iteration the maximal weight among the currently contending set of nodes is found, and a match is registered between the corresponding input-output pairs. An iteration example is depicted in figure 4-2. Upon matching an input to an output, the respective input and output pair is removed from

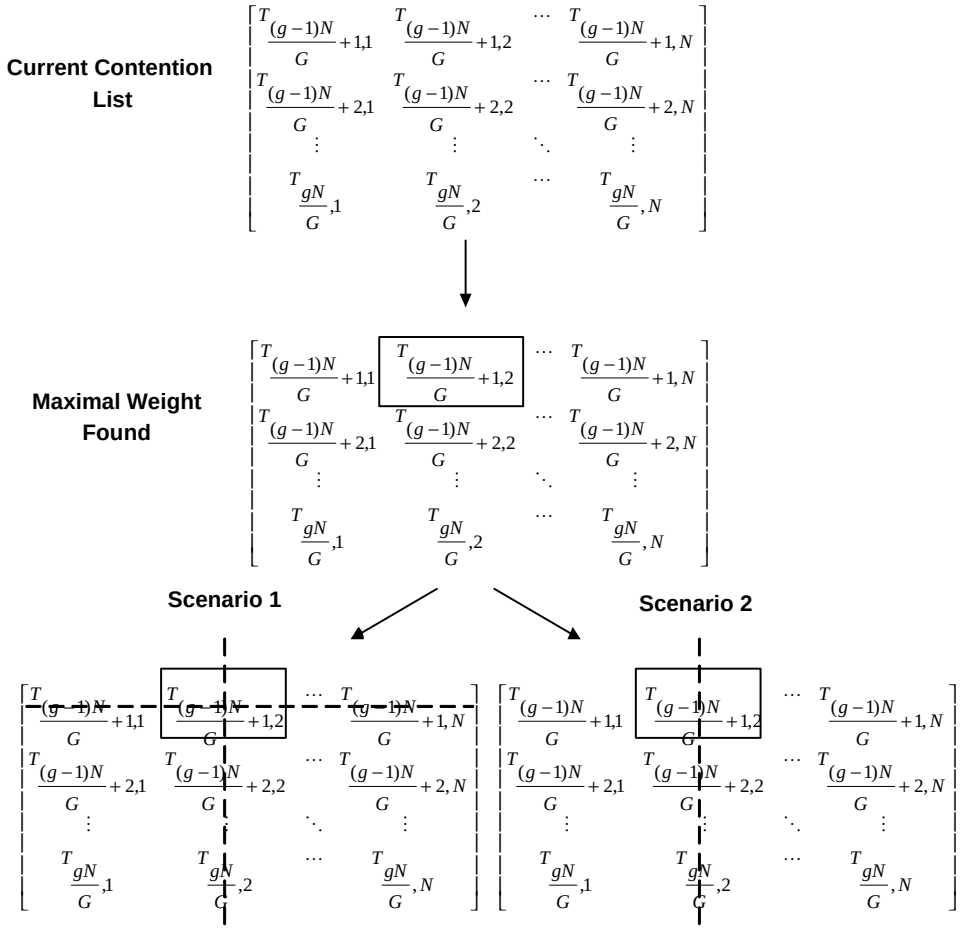

Figure 4-2: Example of an iteration at group g, for the FMWM/OCF scheduling algorithm. Each element represents a weighted request for service. Scenario 1 represents a single input/output matching, while Scenario 2 pertains to the case where more than one input from the same group is matched to an output.

contending during subsequent iterations (shown in scenario 1 of figure 4-2). Alternatively, only the associated output is removed from future contention, as illustrated in scenario 2 of figure 4-2, allowing other inputs from the same group to be matched to available outputs. Assuming the weight matrix is not completely null, the number of iterations can range between 1 and N/G.

Configuration of the crosspoints, which is the outcome of the FMWM/OCF algorithm, can be represented by a service matrix, $S(t) = \{S_{ij}(t)\}$, where $S_{ij}(t) = 1$ if input i is matched to output j at time t, otherwise $S_{ij}(t) = 0$. Based on the weights of the queues, a schedule is obtained which remains unchanged for k consecutive time slots. A new schedule will only occur at time $t + k$, reflected by $S_{ij}(t + k)$.

Next, we establish the following set of definitions:

Definition 17 *Let Λ_g denote the traffic rate matrix for the g^{th} group, as given by*

$$\Lambda_g = \begin{bmatrix} \lambda_{\frac{(g-1)N}{G}+1,1} & \lambda_{\frac{(g-1)N}{G}+1,2} & \cdots & \lambda_{\frac{(g-1)N}{G}+1,N} \\ \lambda_{\frac{(g-1)N}{M}+2,1} & \lambda_{\frac{(g-1)N}{M}+2,2} & \cdots & \lambda_{\frac{(g-1)N}{M}+2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{\frac{gN}{G},1} & \lambda_{\frac{gN}{G},2} & \cdots & \lambda_{\frac{gN}{G},N} \end{bmatrix}$$

Definition 18 *Let $\Omega_g(t)$ denote the weight matrix at time t, such that*

$$\Omega_g(t) = \begin{bmatrix} T_{\frac{(g-1)N}{G}+1,1} & T_{\frac{(g-1)N}{G}+1,2} & \cdots & T_{\frac{(g-1)N}{G}+1,N} \\ T_{\frac{(g-1)N}{G}+2,1} & T_{\frac{(g-1)N}{G}+2,2} & \cdots & T_{\frac{(g-1)N}{G}+2,N} \\ \vdots & \vdots & \ddots & \vdots \\ T_{\frac{gN}{G},1} & T_{\frac{gN}{G},2} & \cdots & T_{\frac{gN}{G},N} \end{bmatrix}$$

where $T_{ij}(t)$ is the waiting time of the HoL cell in queue (i, j) at time t.

Definition 19 *The (aggregate) weight of the FMWM/OCF algorithm at time t is given by*

$$\begin{aligned} W^{FMWM/OCF}(t) &= \sum_{i,j} T_{ij}(t) S_{ij}^{FMWM/OCF}(t) \\ &= \left\langle \Omega_g(t), S_{ij}^{FMWM/OCF}(t) \right\rangle \end{aligned}$$

where $S_{ij}^{FMWM/OCF}(t)$ *denotes the matching configurations determined by the scheduling algorithm at time* t.

Definition 20 *Let* $\tau_{ij}^{(m)}(t)$ *denote the inter-arrival time between two consecutive cells,* m *and* $m+1$*, both of which are stored in queue* (i,j)*, and correspondingly, let* $\tau_{ij}(t) = \max\{\tau_{ij}^{(m)}(t),\, m = 1, 2, ..., Q_{ij}(t)\}$.

Definition 21 *Let* α *denote the maximal number of matches allowed to be made for each input port during a single scheduling period.*

The buffer dynamics under the FMWM/OCF algorithm dictate that for $Q_{ij}(t) > \eta k$

$$T_{ij}(t+k) = T_{ij}(t) + k - S_{ij}(t) \sum_{m=1}^{\eta k} \tau_{ij}^{(m)}(t), \tag{4.1}$$

where η is the internal transfer speedup, while for $Q_{ij}(t) \leq \eta k$

$$\begin{aligned} & T_{ij}(t+k) \leq T_{ij}(t) + k \leq Q_{ij}(t) \cdot \tau_{ij}(t) + k \\ \leq \;& \eta k \tau_{ij}(t) + k \end{aligned} \tag{4.2}$$

from which we can write

$$\begin{aligned} & T_{ij}^2(t+k) - T_{ij}^2(t) \\ \leq \;& k^2 + 2\left[k - S_{ij}(t) \sum_{m=1}^{\eta k} \tau_{ij}^{(m)}(t)\right] T_{ij}(t) \end{aligned} \tag{4.3}$$

for $Q_{ij}(t) > \eta k$, and

$$\begin{aligned} & T_{ij}^2(t+k) - T_{ij}^2(t) \leq k^2 + 2kT_{ij}(t) \\ \leq \;& k^2 + 2\eta k^2 \tau_{ij}(t) \end{aligned} \tag{4.4}$$

for $Q_{ij}(t) \leq \eta k$. The term $\eta k S_{ij}(t)$ expresses the ηk consecutive transmissions that may occur during a single frame interval. Next, we construct a discrete-time quadratic Lyapunov function, $L(t)$, such that $L(t) = \sum_{ij} \lambda_{ij} T_{ij}^2(t)$ [28][41][43]. As an expression of a k time slot

lag, we write

$$L(t+k) - L(t) = \sum_{ij} \lambda_{ij} \left(T_{ij}^2(t+k) - T_{ij}^2(t) \right). \tag{4.5}$$

By partitioning the above into the case of $Q_{ij}(t) < \eta k$ and $Q_{ij}(t) \geq \eta k$, we obtain the following:

$$\begin{aligned}
&E\left[L(t+k) - L(t) | Q(t)\right] \leq \\
&\leq \sum_{ij} \lambda_{ij} \left(k^2 + 2 \left[k - S_{ij}(t) \sum_{m=1}^{\eta k} E\left[\tau_{ij}^{(m)}(t) \right] \right] T_{ij}(t) \right) \cdot \\
&\Pr\left(Q_{ij}(t) \geq \eta k\right) + \sum_{ij} \lambda_{ij} E\left[k^2 + 2\eta k^2 \tau_{ij}(t)\right] \\
&\cdot \Pr\left(Q_{ij}(t) < \eta k\right) \\
&\leq 2 \sum_{ij} \lambda_{ij} \left[k - \frac{\eta k S_{ij}(t)}{\lambda_{ij}} \right] T_{ij}(t) + \sum_{ij} 2(\lambda_{ij} k^2 + \eta k^2) \\
&\leq 2(\eta + \lambda_{ij}) k^2 \frac{N^2}{G} + \sum_{ij} 2 T_{ij}(t) \left[k\lambda_{ij} - \eta k S_{ij}(t) \right] \\
&\leq 2k \left[\langle \Lambda_g, \Omega_g(t) \rangle - \eta \left\langle S_g^{FMWM/OCF}(t), \Omega_g(t) \right\rangle \right] + C
\end{aligned} \tag{4.6}$$

where $C = 2(\eta + \lambda_{ij}) k^2 \frac{N^2}{G}$, is a constant.

In order to prove that the algorithm yields a stable queueing system, we would like to show that beyond a given threshold of maximal weight there is a negative drift in the state of the system, as reflected by the Lyapunov function. Mathematically speaking, from inequality (4.6), an appropriate value for η, such that $\langle \Lambda_g, \Omega_g(t) \rangle < \eta \left\langle S_g^{FMWM/OCF}(t), \Omega_g(t) \right\rangle$, guarantees that the algorithm is stable. Hence, we focus our attention on the two basic scenarios described above, as illustrated in figure 4-2:

Scenario 1: For each matching generated, its respective input and output pair is removed from contending during subsequent iterations.

Theorem 6 *For scenario 1, an $N \times N$ switch running the FMWM/OCF scheduling algorithm with a transfer speedup of 2 is stable under admissible traffic for any frame size, k.*

Proof. *Without loss of generality, assume that following a round of matching, VOQ_{sl},*

where $s\epsilon\left\{\frac{(g-1)N}{G}+1,...,\frac{gN}{G}\right\}$ *and* $l\epsilon[1,...N]$ *is selected. It then follows that all of the elements in row s and column l of the weight matrix are removed from future contention. By decomposing* $<\Lambda_g,\Omega_g>\leq\eta<S_g,\Omega_g>$ *into each round, we have* $\sum_{j=1}^{N}\lambda_{sj}+\sum_{i=\frac{(g-1)N}{M}+1}^{\frac{gN}{M}}\lambda_{il}\leq\eta$. *For any admissible traffic pattern, we know that* $\sum_{j=1}^{N}\lambda_{sj}\leq 1$ *and* $\sum_{i=\frac{(g-1)N}{G}+1}^{\frac{gN}{G}}\lambda_{il}\leq 1$, *from which we deduct that* $\sum_{j=1}^{N}\lambda_{sj}+\sum_{i=\frac{(g-1)N}{G}+1}^{\frac{gN}{G}}\lambda_{il}\leq 2$. *Hence,* $\eta=2$ *is sufficient to guarantee stability.* ■

Corollary 4 *For scenario 1, an* $N\times N$ *switch running the FMWM/OCF scheduling algorithm without transfer speedup is stable under uniform admissible traffic for any frame size, k.*

Proof. *Following similar assertions to those outlined above, under uniform admissible traffic conditions, for any* $i,j\epsilon\{1,2,...,N\}$, $\lambda_{ij}=1/N$. *Therefore,*

$$\sum_{j=1}^{N}\lambda_{sj}-\lambda_{sl}+\sum_{i=\frac{(g-1)N}{G}+1}^{\frac{gN}{G}}\lambda_{il}=\frac{1}{N}(N-1)+\frac{G}{N}\frac{1}{N}\leq\eta.$$

Clearly, $\frac{1}{N}(N-1)+\frac{G}{N}\frac{1}{N}=1-\frac{N-G}{N}<1$, *which implies that* $\eta=1$ *is sufficient to guarantee stability in this scenario.* ■

*Scenario 2***:** *For each match generated, only the associated output is removed from future contentions. In this case, we remove the restriction of only one VOQ being matched per ingress port, such that there can now be up to* α *VOQs matched per ingress port.*

We extend the stability analysis devised thus far to address scenario 2, i.e. the case in which up to $\alpha>1$ VOQs can be matched in each ingress port during every *schedule* (note that α is a fixed number, although in each schedule different input ports may have variant number of actual matches, but the number of matches can not exceed α). A schedule here comprises of multiple rounds/iterations, each of which produces one input-output matching. As such, a schedule round may have at most $\frac{\alpha N}{G}$ rounds/iterations per group.

Theorem 7 *In the case of scenario 2, an* $N\times N$ *switch running the FMWM/OCF scheduling algorithm with transfer speedup of* $\eta\geqslant 1+1/\alpha$ *is stable under admissible traffic for any frame size.*

Proof. Let us first briefly review the matching process. At the beginning of each schedule interval, the VOQ with largest weight is selected; later all VOQs with the same output as that chosen are removed from subsequent contention rounds. Next, the scheduler chooses the VOQ with the largest weight value among those in the current contention list, and then removes all VOQs with same output as the one chosen. The scheduler also checks to see if the number of matches along the same input has reached α. If so, then it removes all VOQs in the same input from future contention. This process is repeated until no more matchings can be made.

Without loss of generality, assume that a schedule produces a total of β matches in a given interval/frame. Hence, there are $\beta \leq \frac{\alpha N}{G}$ rounds/iterations and, each of which corresponds to precisely one match. Further, suppose that during the k^{th} round/iteration, where $k\epsilon\{1, 2, ..., \beta\}$, VOQ_{i_k,j_k}, which has the largest weight value among those in the current contending list, is selected by the scheduler. Moreover, we let I_k, J_k denote the set of all possible input and output indices for the current contention list, respectively; for example, clearly, if $k = 1$, $I_1 = \{\frac{(g-1)N}{G} + 1, \frac{(g-1)N}{G} + 2, ..., \frac{gN}{G}\}$ and $J_1 = \{1, 2, ..., N\}$. If the number of matches (including the most recent one) at the same input, e.g. input $i_k\epsilon I_k$, is α, then all $VOQs$ in the current contention list, with the same output and input as the selected one, are removed from future rounds.

Let $W_{i_k^1,j_k^1}, W_{i_k^2,j_k^2}, ..., W_{i_k^\alpha,j_k^\alpha}$ denote the weight values of the 1^{st},2^{nd},..., α matching of the input, which were selected by the scheduler during rounds $k_1^{th}, k_2^{th}, ..., k_\alpha^{th}$, respectively. Clearly $W_{i_k^1,j_k^1} \geq W_{i_k^2,j_k^2} \geq \cdots \geq W_{i_k^\alpha,j_k^\alpha}$; hence $W_{i_k^\alpha,j_k^\alpha} \leq \frac{1}{\alpha}(W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\alpha,j_k^\alpha})$. Furthermore, let $I_k^1, ..., I_k^\alpha = I_k$ denote the set of all possible input indices for the contending list during rounds $k_1^{th}, k_2^{th}, ..., k_\alpha^{th}$, respectively; and let $J_k^1, ..., J_k^\alpha = J_k$ denote the set of all possible output indices for the contending list using similar round notation. We thus have,

$$\begin{aligned} &\sum_{j\epsilon J_k} \lambda_{i_k,j} W_{i_k,j} \qquad (4.7)\\ &\leq W_{i_k^\alpha,j_k^\alpha} \\ &\leq \frac{1}{\alpha}(W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\alpha,j_k^\alpha}), \end{aligned}$$

and

$$\sum_{i\epsilon I_k^1} \lambda_{i,j_k^1} W_{i,j_k^1} + \sum_{i\epsilon I_k^2} \lambda_{i,j_k^2} W_{i,j_k^2} + ... + \sum_{i\epsilon I_k^\alpha = I_k} \lambda_{i,j_k^\alpha} W_{i,j_k^\alpha} \tag{4.8}$$
$$\leq W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\alpha,j_k^\alpha},$$

hence,

$$\sum_{i\epsilon I_k^1} \lambda_{i,j_k^1} W_{i,j_k^1} + \sum_{i\epsilon I_k^2} \lambda_{i,j_k^2} W_{i,j_k^2} + ... + \sum_{i\epsilon I_k^\alpha = I_k} \lambda_{i,j_k^\alpha} W_{i,j_k^\alpha} \tag{4.9}$$
$$+ \sum_{j\epsilon J_k} \lambda_{i_k,j} W_{i_k,j}$$
$$\leq (1+\frac{1}{\alpha})(W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\alpha,j_k^\alpha})$$

Alternatively, if the number of matches (including the new one) at the same input is α, we have

$$\sum_{i\epsilon I_k^1} \lambda_{i,j_k^1} W_{i,j_k^1} + \sum_{i\epsilon I_k^2} \lambda_{i,j_k^2} W_{i,j_k^2} + ... + \sum_{i\epsilon I_k^\gamma = I_k} \lambda_{i,j_k^\alpha} W_{i,j_k^\alpha} \tag{4.10}$$
$$\leq W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\gamma,j_k^\gamma}$$
$$< (1+\frac{1}{\alpha})(W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\gamma,j_k^\gamma})$$

This represents a general result for every round/iteration. Therefore, by decomposing $< \Lambda_g, \Omega_g > \leq \eta < S_g, \Omega_g >$ into each round according to inequalities (4.9) and (4.10), we conclude that

$$< \Lambda_g, \Omega_g > \leq (1 + \frac{1}{\alpha}) < S_g, \Omega_g >,$$

which implies that $\eta \geqslant 1 + 1/\alpha$ is a sufficient condition to guarantee stability. ■

4.3.2 FMWM/OCF with Multiple Classes of Service

Next, we extend the analysis to show that when multiple classes of service are employed at each input port, the same stability property holds. The underlying assumption is that the notion of virtual output queueing is expanded such that there are now a set of L queues

residing in input i destined to output j, where by each of the queues has a different priority level associated with it. The priority is reflected by a weighted scheme such that the age of the oldest cell in the queue is multiplied by a per-class coefficient to yield the weight used by the scheduler to issue the matching decisions. To clarify this point, we refer to the following definitions:

Definition 22 *Let the scheduling weight vector at time t be defined as*

$$\Omega\left(t\right)=\left[W_{\frac{(g-1)N}{G}+1,1,1},\cdots,W_{\frac{(g-1)N}{G}+1,N,1},\cdots,W_{\frac{gN}{G},N,L}\right]^{T}$$

where $W_{ijl}(t)=C_lT_{ijl}(t)$, $i\epsilon[\frac{(g-1)N}{G}+1,\frac{gN}{G}]$, $j\epsilon[1,N]$, $l=[1,L]$, *and* C_l *is a (positive) weight coefficient of class* l.

Definition 23 *Let* $S_g(t)=\{S_{ijl}(t)\}$ *denote the schedule permutation matrix, where* $S_{ijl}(t)=1$ *if class* l *in input* i *is matched to output* j *at time* t, *otherwise* $S_{ijl}(t)=0$, $i\epsilon[\frac{(g-1)N}{G}+1,\frac{gN}{G}]$, $j\epsilon[1,N]$, $l\epsilon[1,L]$.

Definition 24 *Let* $\tau_{ijl}^{(m)}(t)$ *denote the inter-arrival time between two consecutive cells,* m *and* $m+1$, *both of which are stored in VOQ* $Q_{ijl}(t)$, *and let* $\tau_{ijl}(t)=\max\{\tau_{ijl}^{(m)}(t),\ m=1,2,...,Q_{ijl}(t)\}$.

The configuration of the crossbar, which is the outcome of the FMWM/OCF algorithm, can be represented by the permutation tensor of order three, as illustrated in figure 4-3. Without loss of generality, let us assume that at time t we have explicit knowledge of $W_{ijl}(t)$. The FMWM/OCF algorithm guarantees that $S_{ijl}(t)=1$ always points to the largest value on row i, column j and class of service l, respectively. Following each iteration, the two plains which contained the largest weight are removed, as illustrated in figure 4-3 (in this example plains $\{i',j,l\}$ and $\{i,j',l\}$ are removed).

Similarly to (4.1) and (4.2), when $Q_{ijl}(t)>\eta k$ we may write

$$T_{ijl}(t+k)=T_{ijl}(t)+k-S_{ijl}(t)\sum_{m=1}^{\eta k}\tau_{ijl}^{(m)}(t) \tag{4.11}$$

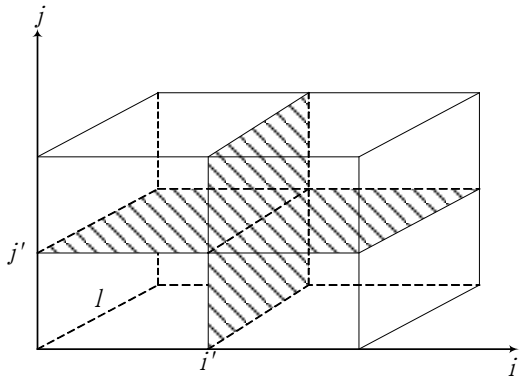

Figure 4-3: Upon generating an input-output match, two plains are removed from the weight tensor, thereby reducing the set of contending nodes in subsequent iterations of the algorithm.

and if $Q_{ijl}(t) \leq \eta k$

$$\begin{aligned} T_{ijl}(t+k) &\leq T_{ijl}(t) + k \leq Q_{ijl}(t) \cdot \tau_{ijl}(t) + k \\ &\leq \eta k \tau_{ijl}(t) + k \end{aligned} \tag{4.12}$$

Applying (4.11) and (4.12) with weighted per-classes OCF, the following holds for $Q_{ijl}(t) > \eta k$

$$W_{ijl}(t+k) = W_{ijl}(t) + C_l k - C_l S_{ijl}(t) \sum_{m=1}^{\eta k} \tau_{ijl}^{(m)}(t) \tag{4.13}$$

otherwise,

$$\begin{aligned} & W_{ijl}(t+k) \leq W_{ijl}(t) + C_l k \\ \leq\ & C_l Q_{ijl}(t) \cdot \tau_{ijl}(t) + C_l k \\ \leq\ & C_l \eta k \tau_{ijl}(t) + C_l k \end{aligned} \tag{4.14}$$

Next, we construct a discrete-time quadratic Lyapunov function, $L(t)$, in the form $L(t) = \sum_{ijl} \frac{\lambda_{ijl}}{C_l} W_{ijl}^2(t)$, for which

$$L(t+k) - L(t) = \sum_{ijl} \frac{\lambda_{ijl}}{C_l} \left[W_{ijl}^2(t+k) - W_{ijl}^2(t) \right] \tag{4.15}$$

>From (4.13) and (4.14), we have the following inequality

$$\begin{aligned} & L(t+k) - L(t) \\ \leq & \sum_{ijl} \frac{\lambda_{ijl}}{C_l} \left[C_l k - C_l S_{ijl}(t) \sum_{m=1}^{\eta k} \tau_{ijl}^{(m)}(t) \right]^2 + \\ & \sum_{ijl} 2 \frac{\lambda_{ijl}}{C_l} \left[C_l k - C_l S_{ijl}(t) \sum_{m=1}^{\eta k} \tau_{ijl}^{(m)}(t) \right] W_{ijl}(t) + \\ & \sum_{ijl} \frac{\lambda_{ijl}}{C_l} C_l^2 \left[k^2 + 2\eta k^2 \tau_{ijl}(t) \right] \end{aligned} \tag{4.16}$$

therefore,

$$\begin{aligned} & E\left[L(t+k) - L(t) | Q(t) \right] \\ \leq & \sum_{ijl} 2\lambda_{ijl} \left[k - S_{ijl}(t) \sum_{m=1}^{\eta k} E\left[\tau_{ijl}^{(m)}(t) \right] \right] W_{ijl}(t) + \\ & \sum_{ijl} \frac{\lambda_{ijl}}{C_l} \left[2C_l^2 k^2 + 2C_l^2 \eta k^2 \tau_{ijl}(t) \right] \\ \leq & \sum_{ijl} 2\lambda_{ijl} \left[k - S_{ijl}(t) \frac{\eta k}{\lambda_{ijl}} \right] W_{ijl}(t) + C \\ \leq & 2k \left[\langle \Lambda_g, \Omega_g(t) \rangle - \eta \left\langle S_g(t), \Omega_g(t) \right\rangle \right] + C \end{aligned} \tag{4.17}$$

where $C = 2 \sum_{ijl} C_l \left[\lambda_{ijl} + \eta \right] k^2$ is a constant.

In order to prove that the algorithm yields a stable queueing system, we would like to show that beyond a given threshold of maximal weight there is a negative drift in the state of the system. From inequality (4.17), an appropriate finite value for η, such that $\langle \Lambda_g, \Omega_g(t) \rangle < \eta \left\langle S_g^{FMWM/OCF}(t), \Omega_g(t) \right\rangle$, would guarantee that the algorithm is stable.

Theorem 8 *For scenario 1, an $N \times N$ switch running the FMWM/OCF scheduling algorithm with a transfer speedup of 2 and multiple classes of service, realized using positive per-class weight coefficients, is stable under admissible traffic for any frame size k.*

Proof. *Without loss of generality, assume that following a round of matching, VOQ_{sml} is selected, where $s\epsilon\left\{\frac{(g-1)N}{G}+1,...,\frac{gN}{G}\right\}$, $m\epsilon\{1,...N\}$,and $l\epsilon\{1,...,L\}$. Hence, all elements in the horizontal plane s and vertical plane m of the weight tensor are removed from future contention. By decomposing $<\Lambda_g,\Omega_g>\leq\eta<S_g,\Omega_g>$ into each round, we have $\sum_{j=1}^{N}\sum_{l=1}^{L}\lambda_{sjl}+\sum_{i=\frac{(g-1)N}{G}+1}^{\frac{gN}{G}}\sum_{l=1}^{L}\lambda_{ijl}\leq\eta$. For any admissible traffic pattern, we know that $\sum_{j=1}^{N}\sum_{l=1}^{L}\lambda_{sjl}\leq 1$ and $\sum_{i=\frac{(g-1)N}{G}+1}^{\frac{gN}{G}}\sum_{l=1}^{L}\lambda_{ijl}\leq 1$, from which we can deduct that $\sum_{j=1}^{N}\sum_{l=1}^{L}\lambda_{sjl}+\sum_{i=\frac{(g-1)N}{G}+1}^{\frac{gN}{G}}\sum_{l=1}^{L}\lambda_{ijl}\leq 2$. Therefore, $\eta=2$, is sufficient to guarantee stability.* ■

Theorem 9 *In the case of scenario 2, an $N\times N$ switch running the FMWM/OCF scheduling algorithm with a transfer speedup of 2 and multiple classes of service, realized using positive per-class weight coefficients, is stable under admissible traffic for any frame size.*

Proof. Without loss of generality, assume that a schedule produces a total of β matches. Hence, there are $\beta\leq\frac{\alpha N}{G}$ rounds/iterations, each of which yields precisely one match. Further, suppose that during the k^{th} round/iteration, where $k\epsilon\{1,2,...,\beta\}$, VOQ_{i_k,j_k,l_k}, which has the largest weight value among those in the current contending list, is selected by the scheduler. Also, we let I_k, J_k, and L_k denote the set of all possible input, output and class of service indices for the current contending list, respectively. For example, if $k=1$ then $I_1=\{\frac{(g-1)N}{G}+1,\frac{(g-1)N}{G}+2,...,\frac{gN}{G}\}$, $J_1=\{1,2,...,N\}$ and $L_1=\{1,2,...,L\}$ If the number of matches (including the most recent one) along the same input, i.e. input $i_k\epsilon I_k$, is α, then all *VOQs* in the current contention list, with the same output and input as the selected one, are removed from future contention rounds.

Let $W_{i_k^1,j_k^1,l_k^1},W_{i_k^2,j_k^2,l_k^2},...,W_{i_k^\alpha,j_k^\alpha,l_k^\alpha}$ denote the weight values of the 1^{st},2^{nd},..., α^{th} match of the input, which were selected by the scheduler during the k^{th} round. Clearly $W_{i_k^1,j_k^1,l_k^1}\geq W_{i_k^2,j_k^2,l_k^2}\geq\cdots\geq W_{i_k^\alpha,j_k^\alpha,l_k^\alpha}$; hence $W_{i_k^\alpha,j_k^\alpha,l_k^\alpha}\leq\frac{1}{\alpha}(W_{i_k^1,j_k^1,l_k^1}+W_{i_k^2,j_k^2,l_k^2}+...+W_{i_k^\alpha,j_k^\alpha,l_k^\alpha})$. Further, let $I_k^1,...,I_k^\alpha=I_k$ denote the set of all possible input indices for the contending list during the k^{th} round; let $J_k^1,...,J_k^\alpha=J_k$ represent the set of all possible output indices for the

contending list and $L_k^1, ..., L_k^\alpha = L_k$ the set of all possible class of service indices. We thus have,

$$\begin{aligned} &\sum_{j\epsilon J_k}\sum_{l\epsilon L_k}\lambda_{i_k,j,l}W_{i_k,j,l} \\ &\leq W_{i_k^\alpha,j_k^\alpha,l_k^\alpha} \\ &\leq \frac{1}{\alpha}(W_{i_k^1,j_k^1,l_k^1} + W_{i_k^2,j_k^2,l_k^2} + ... + W_{i_k^\alpha,j_k^\alpha,l_k^\alpha}), \end{aligned} \tag{4.18}$$

and

$$\begin{aligned} &\sum_{i\epsilon I_k^1}\sum_{l\epsilon L_k^1}\lambda_{i,j_k^1,l}W_{i,j_k^1,l} + \sum_{i\epsilon I_k^2}\sum_{l\epsilon L_k^2}\lambda_{i,j_k^2,l}W_{i,j_k^2,l} + ... \\ &+ \sum_{i\epsilon I_k^\alpha = I_k}\sum_{l\epsilon L_k^\alpha}\lambda_{i,j_k^\alpha,l}W_{i,j_k^\alpha,l} \leq W_{i_k^1,j_k^1,l_k^1} + W_{i_k^2,j_k^2,l_k^2} + \\ &... + W_{i_k^\alpha,j_k^\alpha,l_k^\alpha}, \end{aligned} \tag{4.19}$$

hence,

$$\begin{aligned} &\sum_{i\epsilon I_k^1}\sum_{l\epsilon L_k^1}\lambda_{i,j_k^1,l}W_{i,j_k^1,l} + \sum_{i\epsilon I_k^2}\sum_{l\epsilon L_k^2}\lambda_{i,j_k^2,l}W_{i,j_k^2,l} + ... \\ &+ \sum_{i\epsilon I_k^\alpha = I_k}\sum_{l\epsilon L_k^\alpha}\lambda_{i,j_k^\alpha,l}W_{i,j_k^\alpha,l} + \sum_{j\epsilon J_k}\sum_{l\epsilon L_k}\lambda_{i_k,j,l}W_{i_k,j,l} \\ &\leq (1+\frac{1}{\alpha})(W_{i_k^1,j_k^1,l_k^1} + W_{i_k^2,j_k^2,l_k^2} + ... + W_{i_k^\alpha,j_k^\alpha,l_k^\alpha}) \end{aligned} \tag{4.20}$$

Alternatively, if the number of matches (including the new one) along the same input is equal to $\gamma < \alpha$, we have

$$\sum_{i\epsilon I_k^1}\sum_{l\epsilon L_k^1}\lambda_{i,j_k^1,l}W_{i,j_k^1,l}+\sum_{i\epsilon I_k^2}\sum_{l\epsilon L_k^2}\lambda_{i,j_k^2,l}W_{i,j_k^2,l}+... \tag{4.21}$$
$$+\sum_{i\epsilon I_k^\alpha=I_k}\sum_{l\epsilon L_k^\alpha}\lambda_{i,j_k^\alpha,l}W_{i,j_k^\alpha,l}$$

$$\leq W_{i_k^1,j_k^1,l_k^1}+W_{i_k^2,j_k^2,l_k^2}+...+W_{i_k^\alpha,j_k^\alpha,l_k^\alpha} \tag{4.22}$$
$$<(1+\frac{1}{\alpha})(W_{i_k^1,j_k^1,l_k^1}+W_{i_k^2,j_k^2,l_k^2}+...+W_{i_k^\alpha,j_k^\alpha,l_k^\alpha})$$

This is a general result for every round/iteration. Therefore, by decomposing $<\Lambda_g,\Omega_g>\leq \eta<S_g,\Omega_g>$ into each round according to inequalities (4.20) and (4.21), we obtain

$$<\Lambda_g,\Omega_g>\leq(1+\frac{1}{\alpha})<S_g,\Omega_g>,$$

which implies that $\eta \geqslant 1+1/\alpha$ is a sufficient condition to guarantee stability. ■

4.3.3 FMWM/LQF for a Single Class of Service

Definition 25 *Let Λ_g denote the traffic rate matrix for the g^{th} group, as given by*

$$\Lambda_g=\begin{bmatrix}\lambda_{\frac{(g-1)N}{M}+1,1} & \lambda_{\frac{(g-1)N}{M}+1,2} & \cdots & \lambda_{\frac{(g-1)N}{M}+1,N}\\ \lambda_{\frac{(g-1)N}{M}+2,1} & \lambda_{\frac{(g-1)N}{M}+2,2} & \cdots & \lambda_{\frac{(g-1)N}{M}+2,N}\\ \vdots & \vdots & \ddots & \vdots\\ \lambda_{\frac{gN}{M},1} & \lambda_{\frac{gN}{M},2} & \cdots & \lambda_{\frac{gN}{M},N}\end{bmatrix} \tag{4.23}$$

Definition 26 *Let $\Omega_g(t)$ denote the weight matrix at time t, such that*

$$\Omega_g(t)=\begin{bmatrix}Q_{\frac{(g-1)N}{M}+1,1} & Q_{\frac{(g-1)N}{M}+1,2} & \cdots & Q_{\frac{(g-1)N}{M}+1,N}\\ Q_{\frac{(g-1)N}{M}+2,1} & Q_{\frac{(g-1)N}{M}+2,2} & \cdots & Q_{\frac{(g-1)N}{M}+2,N}\\ \vdots & \vdots & \ddots & \vdots\\ Q_{\frac{gN}{M},1} & Q_{\frac{gN}{M},2} & \cdots & Q_{\frac{gN}{M},N}\end{bmatrix} \tag{4.24}$$

Definition 27 *The weight of the FMWM algorithm at time t is given by*

$$\begin{aligned} W^{FMWM}(t) &= \sum_{i,j} Q_{ij}(t) S_{ij}^{FMWM}(t) \\ &= \left\langle Q(t), S_g^{FMWM}(t) \right\rangle \end{aligned} \tag{4.25}$$

where $S_{ij}^{FMWM}(t)$ denotes the matching configurations established by the algorithm at time t.

Since at most k packets may arrive during k time slots, when applying the FMWM algorithm, the following equality holds

$$Q_{ij}(t+k) = [Q_{ij}(t) - \eta k S_{ij}(t)]^+ + \sum_{m=0}^{k-1} A_{ij}(t+m), \tag{4.26}$$

from which we can write

$$\begin{aligned} & Q_{ij}^2(t+k) - Q_{ij}^2(t) \\ \leq \;& 2Q_{ij}(t) \left[\left(\sum_{m=0}^{k-1} A_{ij}(t+m) \right) - \eta k S_{ij}(t) \right] + k^2, \end{aligned} \tag{4.27}$$

for $Q_{ij}(t) > \eta k$, and

$$Q_{ij}^2(t+k) - Q_{ij}^2(t) \leq 2\eta k^2 + k^2, \tag{4.28}$$

for $Q_{ij}(t) \leq \eta k$. The term $\eta k S_{ij}(t)$ expresses the ηk consecutive transmissions that may occur during a frame interval. Next, we construct a discrete-time quadratic Lyapunov function, $L(t)$, such that $L(t) = \langle \Omega_g(t), \Omega_g(t) \rangle = \sum_{i,j} Q_{ij}^2(t)$ [28][41][43]. As an expression of a k time slot lag, we write

$$L(t+k) - L(t) = \sum_{ij} \left(Q_{ij}^2(t+k) - Q_{ij}^2(t) \right). \tag{4.29}$$

By partitioning the above into the case of $Q_{ij}(t) < \eta k$ and $Q_{ij}(t) \geq \eta k$, we derive the following:

$$
\begin{aligned}
& E\left[L(t+k) - L(t) | Q(t)\right] \leq \qquad (4.30) \\
& \sum_{ij} \left\{ 2Q_{ij}(t) \left[E\left(\sum_{m=0}^{k-1} A_{ij}(t+m) \right) - \eta k S_{ij}(t) \right] + k^2 \right\} \cdot \\
& \Pr\left(Q_{ij}(t) \geq \eta k\right) + \sum_{ij} k(2\eta k + k) \cdot \Pr\left(Q_{ij}(t) < \eta k\right) \\
& \leq \sum_{ij} 2Q_{ij}(t) \left[E\left(\sum_{m=0}^{k-1} A_{ij}(t+m) \right) - \eta k S_{ij}(t) \right] \\
& + \sum_{ij} 2(\eta+1)k^2 \\
& \leq 2(\eta+1)k^2 \frac{N^2}{M} + \sum_{ij} 2Q_{ij}(t) \left[k\lambda_{ij} - \eta k S_{ij}(t) \right] \\
& \leq 2k \left[\langle \Lambda_g, \Omega_g(t) \rangle - \eta \left\langle S_g^{FMWM}(t), \Omega_g(t) \right\rangle \right] + \\
& \quad 2(\eta+1)k^2 \frac{N^2}{M} \qquad (4.31)
\end{aligned}
$$

In order to prove that the algorithm yields a stable queueing system, we would like to show that beyond a given threshold of maximal weight there is a negative drift in the state (queue occupancies) of the system. Mathematically speaking, from inequality 4.30, if we can find an appropriate value of η, such that $\langle \Lambda_g, \Omega_g(t) \rangle < \eta \left\langle S_g^{FMWM}(t), \Omega_g(t) \right\rangle$, then it follows that the algorithm is stable. We focus our attention on the two basic scenarios described above (as illustrated in figure *4-2)*:

Scenario 1: *Each time a match is generated, its respective input and output pair is removed from contending during subsequent iterations (illustrated in scenario 1 of figure 4-2)*

Theorem 10 *For scenario 1, an $N \times N$ switch running the FMWM scheduling algorithm with a transfer speedup of 2 is stable under admissible traffic for any frame size, k.*

Proof. *Without loss of generality, assume that following a round of matching, VOQ_{sl}, where $s \epsilon \left\{ \frac{(g-1)N}{M} + 1, ..., \frac{gN}{M} \right\}$ and $l \epsilon [1, ...N]$, is selected, then all of elements in row s and*

column l of the weight matrix are removed from future contention. By decomposing $< \Lambda_g, \Omega_g > \leq \eta < S_g, \Omega_g >$ *into each round, we have* $\sum_{j=1}^{N} \lambda_{sj} + \sum_{i=\frac{(g-1)N}{M}+1}^{\frac{gN}{M}} \lambda_{il} \leq \eta$. *For any admissible traffic pattern, we know that* $\sum_{j=1}^{N} \lambda_{sj} \leq 1$ *and* $\sum_{i=\frac{(g-1)N}{M}+1}^{\frac{gN}{M}} \lambda_{il} \leq 1$, *from which we can deduct that* $\sum_{j=1}^{N} \lambda_{sj} + \sum_{i=\frac{(g-1)N}{M}+1}^{\frac{gN}{M}} \lambda_{il} \leq 2$. *Therefore,* $\eta = 2$, *is sufficient to guarantee stability.* ■

Theorem 11 *For scenario 1, an $N \times N$ switch running the FMWM scheduling algorithm without transfer speedup is stable under uniform admissible traffic for any frame size, k.*

Proof. *Following similar assertions as those shown for Theorem 1, under uniform admissible traffic conditions, for any* $i, j \epsilon \{1, 2, ..., N\}$, $\lambda_{ij} = 1/N$. *Hence,* $\sum_{j=1}^{N} \lambda_{sj} - \lambda_{sl} + \sum_{i=\frac{(g-1)N}{M}+1}^{\frac{gN}{M}} \lambda_{il} = \frac{1}{N}(N-1) + \frac{M}{N}\frac{1}{N} \leq \eta$. *Clearly,* $\frac{1}{N}(N-1) + \frac{M}{N}\frac{1}{N} = 1 - \frac{N-M}{N} < 1$. *Therefore,* $\eta = 1$ *is sufficient to guarantee stability.* ■

Scenario 2: *Each time a match is generated, only the associated output is removed from future contentions (as illustrated in scenario 2 of figure 4-2). In this case, we remove the limitation of only one VOQ being matched per ingress port, such that there can now be up to α VOQs matched per ingress port.*

We extend the stability analysis devised thus far to address scenario 2, i.e. the case in which up to α VOQs can be matched in each ingress port during every *schedule* (a schedule consists of multiple rounds/iterations, each of which produces one input-output matching). In this case a schedule may have at most $\frac{\alpha N}{M}$ rounds/iterations.

Theorem 12 *In the case of scenario 2, an $N \times N$ switch running the FMWM scheduling algorithm with transfer speedup of* $\eta = 1 + 1/\alpha$ *is stable under any admissible traffic for any frame size.*

Proof. Let us first briefly review the matching process. At the beginning of each schedule, the VOQ with largest weight is selected; later all VOQs with the same output as the chosen one are removed from subsequent contention rounds. Next, the scheduler chooses the VOQ with the largest weight value among those in the current contention list, and then removes all VOQs with same output as the one chosen. The scheduler also checks

to see if the number of matches along the same input has reached α. If so, then it removes all the VOQs in the same input from future contention. This process is repeated until no more matches can be made. Let β denote the total number of matches made per schedule. It follows that after β rounds/iterations, all remaining VOQs have zero weight value.

Without loss of generality, assume that a schedule produces a total of β matches matches. Hence, there are β rounds/iterations and $\beta \leq \frac{\alpha N}{M}$, each of which yields precisely one match. Further, suppose that during the k^{th} round/iteration, where $k\epsilon\{1, 2, ..., \beta\}$, VOQ_{i_k,j_k}, which has the largest weight value among those in the current contending list, is selected by the scheduler. Also, we let I_k, J_k denote the set of all possible input indices and output indices for the current contending list, respectively; for example,clearly, if $k = 1$, $I_1 = \{\frac{(g-1)N}{M} + 1, \frac{(g-1)N}{M} + 2, ..., \frac{gN}{M}\}$ and $J_1 = \{1, 2, ..., N\}$. If the number of matches (including the new one) along the same input, say the input $i_k\epsilon I_k$, is α, then all $VOQs$ in the current contention list, with the same output and input as the selected one, are removed from future contention rounds. Let $W_{i_k^1,j_k^1}, W_{i_k^2,j_k^2}, ..., W_{i_k^\alpha,j_k^\alpha}$ denote the weight values of the 1^{st},2^{nd},..., α^{th} match of the input, which were selected by the scheduler during the $k_1^{th}, k_2^{th}, ..., k_\alpha^{th} = k^{th}$ round, respectively. Clearly $W_{i_k^1,j_k^1} \geq W_{i_k^2,j_k^2} \geq \cdots \geq W_{i_k^\alpha,j_k^\alpha}$; hence $W_{i_k^\alpha,j_k^\alpha} \leq \frac{1}{\alpha}(W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\alpha,j_k^\alpha})$. Further, let $I_k^1, ..., I_k^\alpha = I_k$ denote the set of all possible input indices for the contending list during the $k_1^{th}, k_2^{th}, ..., k_\alpha^{th} = k^{th}$ round, respectively; and let $J_k^1, ..., J_k^\alpha = J_k$ denote the set of all possible output indices for the contending list during the $k_1^{th}, k_2^{th}, ..., k_\alpha^{th} = k^{th}$ round, respectively. We thus have,

$$\begin{aligned}
&\sum_{j\epsilon J_k} \lambda_{i_k,j} W_{i_k,j} \\
&\leq W_{i_k^\alpha,j_k^\alpha} \\
&\leq \frac{1}{\alpha}(W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\alpha,j_k^\alpha})
\end{aligned}$$

and

$$\begin{aligned}
&\sum_{i\epsilon I_k^1} \lambda_{i,j_k^1} W_{i,j_k^1} + \sum_{i\epsilon I_k^2} \lambda_{i,j_k^2} W_{i,j_k^2} + ... + \sum_{i\epsilon I_k^\alpha = I_k} \lambda_{i,j_k^\alpha} W_{i,j_k^\alpha} \\
&\leq W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\alpha,j_k^\alpha}
\end{aligned}$$

hence,

$$\begin{aligned}
&\sum_{i\epsilon I_k^1} \lambda_{i,j_k^1} W_{i,j_k^1} + \sum_{i\epsilon I_k^2} \lambda_{i,j_k^2} W_{i,j_k^2} + ... + \sum_{i\epsilon I_k^\alpha = I_k} \lambda_{i,j_k^\alpha} W_{i,j_k^\alpha} \\
&+ \sum_{j\epsilon J_k} \lambda_{i_k,j} W_{i_k,j} \\
&\leq (1+\frac{1}{\alpha})(W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\alpha,j_k^\alpha})
\end{aligned} \tag{4.32}$$

Or else, if the number of matches (including the new one) along the same input is equal to $\gamma < \alpha$, we still have

$$\begin{aligned}
&\sum_{i\epsilon I_k^1} \lambda_{i,j_k^1} W_{i,j_k^1} + \sum_{i\epsilon I_k^2} \lambda_{i,j_k^2} W_{i,j_k^2} + ... + \sum_{i\epsilon I_k^\gamma = I_k} \lambda_{i,j_k^\alpha} W_{i,j_k^\alpha} \\
&\leq W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\gamma,j_k^\gamma} \\
&< (1+\frac{1}{\alpha})(W_{i_k^1,j_k^1} + W_{i_k^2,j_k^2} + ... + W_{i_k^\gamma,j_k^\gamma})
\end{aligned} \tag{4.33}$$

This is a general result for every round/iteration. Therefore, by decomposing $< \Lambda_g, \Omega_g > \leq \eta < S_g, \Omega_g >$ into each round according to the inequalities (4.32) and (4.33), we have

$$< \Lambda_g, \Omega_g > \leq (1 + \frac{1}{\alpha}) < S_g, \Omega_g >$$

which means that $\eta = 1 + 1/\alpha$ is a sufficient condition to guarantee stability. ■

4.4 Reconfiguration Delay Analysis

There have been many crossbar technologies introduced in the literature with applications to input-queued packet switching fabrics. These range from electronics-based to optical technologies, such as wavelength division multiplexing (WDM) passive stars [44][45]. In most practical systems, there is a non-negligable reconfiguration delay that is introduced when the crosspoint switch is configured. Such delay can range from several nanoseconds to a few tens of microseconds. In addition, high-speed serializer/deserializer (SerDes) devices typically require somewhere in the order of 100 bit-times to lock on to a new signal. This

locking process becomes ever more intricate as link speeds continue to increase. Similar durations are observed in optical WDM tunable lasers, which require time to switch from one wavelength to another. The use of optical switch fabrics in building high capacity switches is often desirable since it allows higher port speed and less power consumption than electronic interconnects. However, the traditional packet-by-packet IQ switching architecture with maximal matching algorithms is not suitable for switches built on optical switch fabrics because of the slow reconfiguration delay of high capacity optical switch fabrics. The reconfiguration time of an optical switch fabric is much longer than that of an electronic fabric. The total reconfiguration overhead for a typical optical switch fabric can be in the range of 50-100ns [46]. However, with 64-byte packets and speeds of 40 Gbps, a reconfiguration time of a few nanoseconds is needed for the packet-by-packet switching mechanism.

Let R denote the link rate, L denote the length of a packet, and κ_R as the mean duration required for fabric reconfiguration. If packet-by-packet scheduling scheme is used, the ratio of actual transmission of packet is $\frac{L/R}{L/R+\kappa_R}$. With 64-byte packets and speeds of 40 Gbps, assume the mean reconfiguration delay $\kappa_R = 100ns$. The ratio of actual transmission of packet is about 0.09. This suggests that the traditional packet-by-packet IQ switching architecture with maximal matching algorithms is impractical for switches built on optical switch fabrics.

We consider the FMWM which consists of an iterative process whereby in each iteration the maximal weight is found and a match is registered between its associated input-ouput pair. The matching matrix remains unchanged during the following k time slots. In the interest of incorporating such reconfiguration delays, let us define κ_R as the mean duration required for the system to reconfigure itself. Letting p denotes the number of payload bits in each fixed-size packet, then the ratio of actual transmission of packet is $\frac{kp/R}{\kappa_R+kp/R}$. With 64-byte packets and speeds of 40 Gbps, assume the mean reconfiguration delay $\kappa_R = 100ns$ and take $k = 16$ as the frame size, then the ratio of actual transmission of packet is about 0.61, compared with packet-by-packet scheduling, which is a big improvement.

In table 4.1, we outline the ratio of actual to gross transmission time for different frame sizes. Clearly, when the frame size reaches 64 time slots, this ratio can be as high as 86%,

Table 4.1: Ratio of actual to gross transmission time for different frame sizes

Frame Size	Ratio of actual to gross transmission time
$k = 1$	$\frac{kp/R}{\kappa_R + kp/R} = \frac{10}{110} = 0.09$
$k = 8$	$\frac{kp/R}{\kappa_R + kp/R} = \frac{80}{180} = 0.44$
$k = 16$	$\frac{kp/R}{\kappa_R + kp/R} = \frac{160}{260} = 0.61$
$k = 32$	$\frac{kp/R}{\kappa_R + kp/R} = \frac{320}{420} = 0.76$
$k = 64$	$\frac{kp/R}{\kappa_R + kp/R} = \frac{640}{740} = 0.86$

which suggests that by using frame-based scheduling algorithm, we can significantly relax the timing constraints imposed on the optical switch fabrics by the reconfiguration delay.

4.5 Performance Simulation Results

4.5.1 Single Crosspoint based Architecture

In order to evaluate the performance of the FMWM/OCF algorithm under the multi-crosspoints based architecture proposed, four sets of simulations were carried out. In all cases, a 12×12 switch was considered with a transfer speedup of 2. The switch was partitioned into 4 independent switching groups, each of which supported 3 ingress ports. In the first three sets of simulations $\alpha = 1$ (i.e. the transfer speedup is 2).

In the first set of simulations, the arrival process was Bernoulli i.i.d. with uniformly distributed destination distribution. Figure 4-4 shows the mean delay measured when employing FMWM/OCF with different switching frame sizes (k). As can be intuitively appreciated, the longer the frame the larger the mean delay, which stems from the fact that during many switching intervals less than k consecutive cells are being transmitted. Moreover, it is noted that larger frame sizes exhibit faster delay growth (steeper slope). This can be explained by the fact that once a matching matrix is generated, the unmatched VOQs will not transmit any cells during k time slots, yet they continue to accumulate newly arriving cells (which contribute to the increase of the average queue waiting time).

The second set of simulations was targeted at examining the impact of bursty traffic on the delay characteristics. A two-state Markov-modulated (ON/OFF) process was employed [47], whereby bursts are uniformly distributed across the outputs. Figure 4-5 shows the

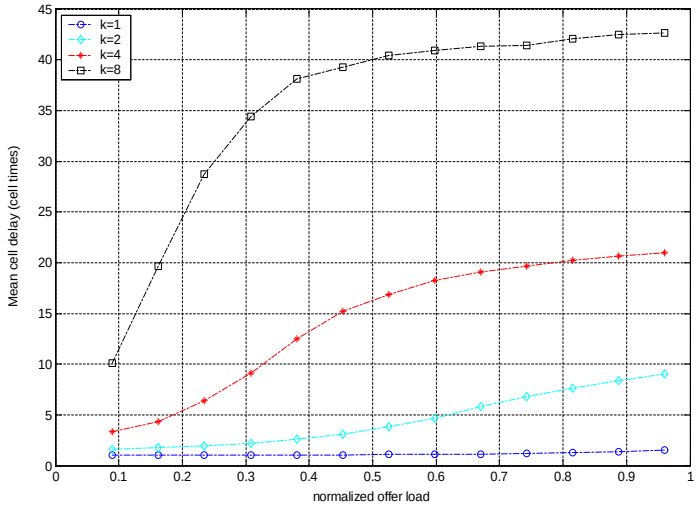

Figure 4-4: Average cell delay when arrivals are Bernoulli i.i.d. and uniformly distributed, for different frame sizes (k). The FMWM algorithm can issue at most 1 match per ingress port

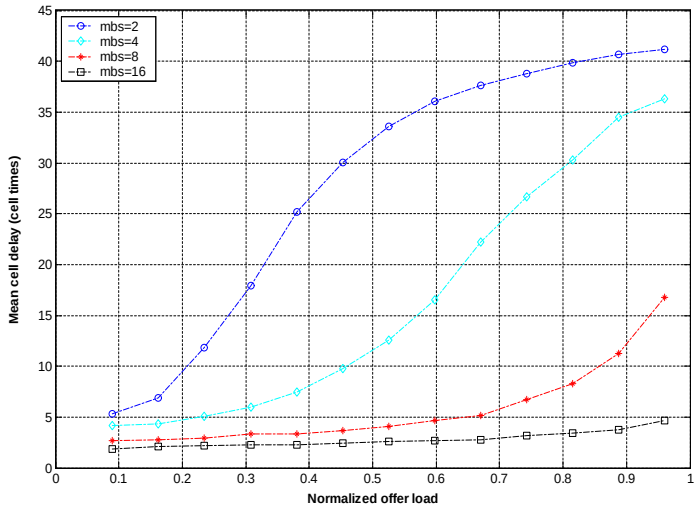

Figure 4-5: Average cell delay as a function of the mean burst size (MBS) for a fixed frame size of 8 cells. The FMWM/OCF algorithm can issue at most 1 match per ingress port.

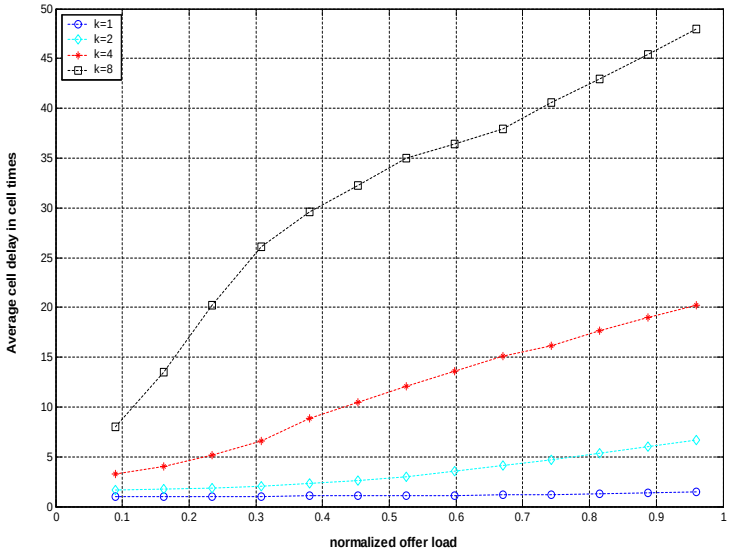

Figure 4-6: Average cell delay when arrivals are Bernoulli i.i.d. and distributed (non-uniformly) according to the Zipf law (with $r = 1$). The FMWM/OCF algorithm can issue at most 1 match per ingress port.

average delay as a function of the mean burst sizes (MBS) for a fixed frame size of 8 packets. An inverse relationship between the mean bursty size and the average delay is observed. The reason is that since the FMWM scheduling discipline is inherently correlated, bursty traffic better utilizes the transmission intervals.

Figure 4-6 depicts the average delay for Zipf distributed packet streams with Bernoulli arrival characteristics, as a function of the offered traffic load.

In the fourth set of simulations, the FMWM/OCF algorithm was allowed to make up to 2 matches per ingress port. Correspondingly, the transfer speedup is dropped to 1.5. The arrival process was Bernoulli i.i.d with uniformly distributed destination distribution. Figure 4-7 depicts the average delay measured for different frame sizes. Compared with the first simulation set, whereby the scheduling algorithm can only issue at most 1 match for each ingress port, the average delay for each frame size is much lower than its counterpart, which is due to the fact that allowing multiple matches per ingress port generally better utilizes the transmission intervals. It is observed that despite the relaxed switching times

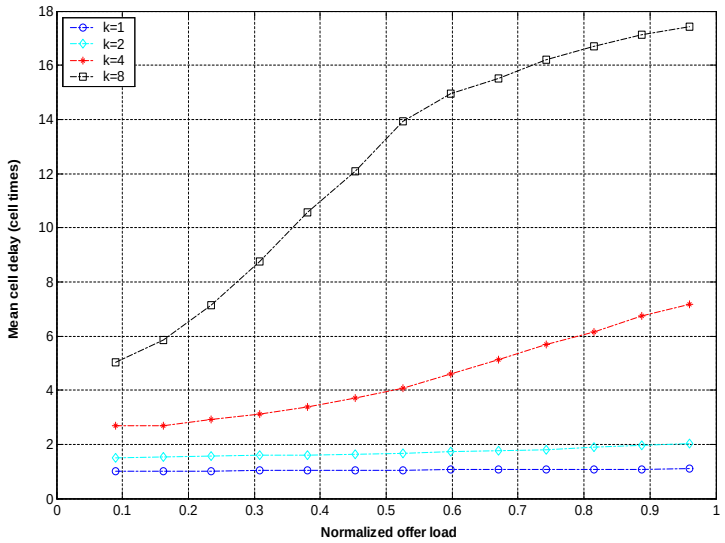

Figure 4-7: Average cell delay when arrivals are Bernoulli i.i.d. and uniformly distributed, for different frame sizes (k). The FMWM algorithm issues up to 2 matches per ingress port.

and distributed passive crosspoint switches, the overall performance is kept high.

The last set of simulations was targeted at examining the impact of multiple classes of service on the delay performance. Linear priority coefficients were used, such that $W_{ijl}(t) = l \times T_{ijl}(t)$, where $l \in \{1, 2, 3\}$ denotes the class of service index. Once again, a 12-port switch with 3 classes of service was examined. The arrival processes was identical to that described above, while traffic was uniformly distributed across the classes of service. The frame size was set to 8 cells. As can be observed in figure 4-8, the higher the priority level the lower the mean delay experienced by arriving packets, a result of the fact that the scheduler tends to service queues with higher weights during each iteration. The disparity between the different classes of service can be controlled by tuning the priority coefficients. The theory presented in this chapter clearly suggests that so long as the class coefficients are non-zero, under the FMWM/OCF scheduler, the system is guaranteed to remain stable.

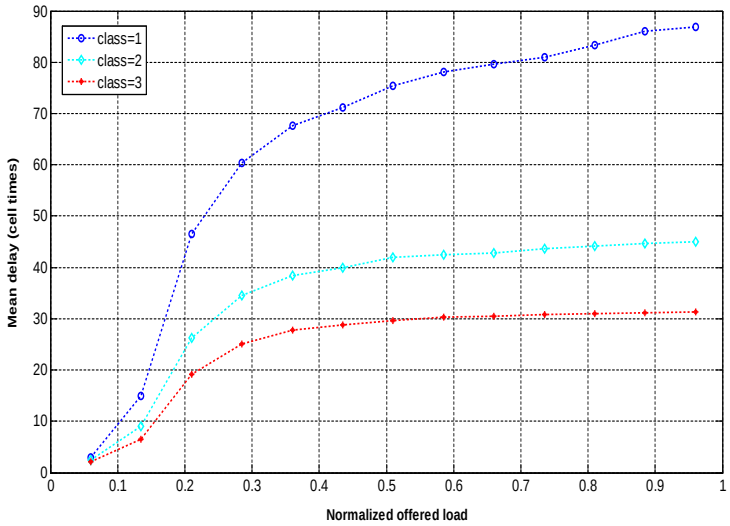

Figure 4-8: Mean delay of FMWM/OCF for multiple classes of service. Classes of service are differentiated via linear priority coefficients.

4.5.2 Simulation Results

In order to evaluate the performance of the FMWM algorithm under the multi-crosspoints based architecture proposed, three sets of simulations were carried out. In all cases, a 12×12 switch was considered with a transfer speedup of 2. The switch was partitioned into 4 independent switching groups, each of which contained 3 ingress ports. While the first two sets of simulations were dedicated to examining performance for the FMWM algorithm in the case of *scenario 1* ($\alpha = 1$) under both uniform Bernoulli and bursty traffic, the third set of simulations was targeted at examining the performance in the case of *scenario 2* ($\alpha > 1$).

In the first simulation set, the arrival process was Bernoulli i.i.d. with uniformly distributed destination distribution. Figure 4-9 depicts the average delay when employing FMWM for different frame sizes. As can be intuitively appreciated, the longer the frame the larger the average delay, which is due to the fact that during many switching intervals many queues may store less than k packets thereby under-utilizing the transmission intervals (i.e. frames). Moreover, it is noted that larger frame sizes exhibit faster delay growth

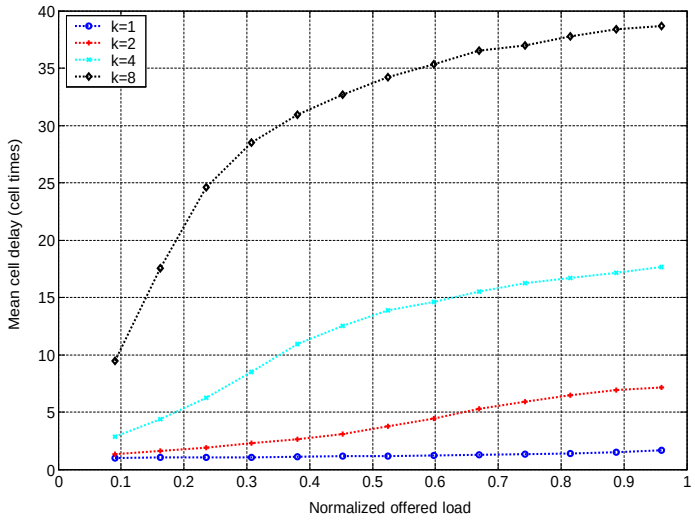

Figure 4-9: Average cell delay when arrivals are Bernoulli i.i.d. and uniformly distributed, for different frame sizes (k). The FMWM algorithm can issue at most 1 match per ingress port

(steeper slope), since when a matching matrix is generated, the unmatched VOQs will not transmit any packets during the k consecutive time slots, yet they continue to buffer newly arriving packets.

The second set of simulations was targeted at examining the impact of bursty traffic on the delay characteristics. A two-state Markov-modulated (ON/OFF) process was employed, whereby bursts are uniformly distributed across the outputs. Figure 4-10 shows the average delay as a function of the mean burst sizes (MBS) for a fixed frame size of 8 packets. An inverse relationship between the mean bursty size and the average delay is observed. The reason is that since the service discipline is inherently correlated, bursty traffic better utilizes the transmission intervals.

In the third set of simulations, the FMWM algorithm was allowed to make up to 2 matches per ingress port. The arrival process was Bernoulli i.i.d with uniformly distributed destination distribution. Figure 4-11 depicts the average delay for different frame sizes. Compared with the first simulation set, whereby the scheduling algorithm can only issue at most 1 match for each ingress port, the average delay for each frame size is much lower than

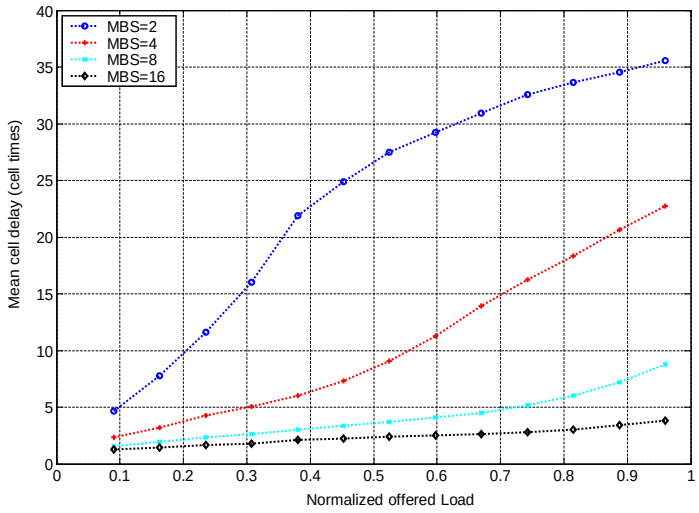

Figure 4-10: Average cell delay as a function of the mean burst size (MBS) for a fixed frame size of 8 cells

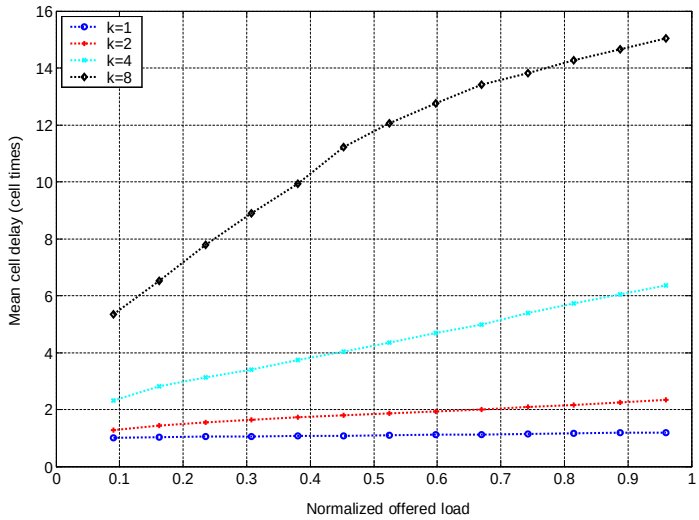

Figure 4-11: Average cell delay when arrivals are Bernoulli i.i.d. and uniformly distributed, for different frame sizes (k). The FMWM algorithm can issue up to 2 matches per ingress port

its counterpart in the first simulation set, which is due to the fact that allowing multiple matches per ingress port generally better utilizes the transmission intervals. It is seen that despite the relaxed switching times and distributed passive crosspoint switches, the overall performance is kept high.

Chapter 5

A Scalable Packet Placement Architecture for PSM Switches

Output queued (OQ) switches offer several highly desirable performance characteristics, including minimal average packet delay, controllable Quality of Service (QoS) provisioning and work-conservation under any admissible traffic [48][49][50][51]. 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. The latter is derived from the need to be able to accept (write) up to N arriving packets while, simultaneously, transmit (read) up to N departing packets. This requirement significantly limits scalability of OQ switches with respect to their aggregate switching capacity. In an aim to mimic 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, have recently received attention [37]. In order to operate properly, the PSM switch must have sufficient bandwidth. At the core of distributed shared memory architectures is the memory management algorithm, which 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[37], thereby inherently limiting the scalability of the scheme.

In this chapter, we propose a novel memory management approach for PSM switches

which employs a pipelined hardware architecture. The algorithm parallelizes the packet placement process, thereby gaining execution speed at the expense of a fixed latency. In addition, we analyze the conditions on the number of memories required for the given algorithm to guarantee that the PSM switch successfully emulates a First-Come-First-Served (FCFS) output-queued switch.

5.1 The Distributed Shared Memory Switch

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 $K = 3N - 1$ [37]. The latter can be proven by employing constraint sets analysis (also known as the "pigeon hole" principal), 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, each arriving packet must not 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 last 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 bound 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 [52]. Regardless of the precise minimal number of memories used, a key challenge pertains 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 order to make the PSM switch practical, it is highly desirable to obtain an algorithm which performs the packet placement process in a shorter period of time.

5.2 Sequential Packet Placement Algorithm

Let k denote the number of memory units available in a PSM switch. We assume that we have knowledge of the departure time of each arriving packet prior to its placement in the shared memory bank. For FCFS emulation, establishing the departure time of arriving packets is rather straightforward. In addition, we begin by assuming that the shared memories are all dual-port, which means that they perform a read *and* write operation during a single time slot. By employing constraints sets analysis, we initially observe that no arriving packet can be placed in a memory that has already been selected by another packet arriving at the same time slot. This translates to a need for at least N available memory units. Moreover, a packet can not be placed in a memory which hosts another packet with the same departure time (destined to a different output). Since at any given time there can be at most N packets with the same departure time, we conclude that there should be at least $2N - 1$ memories available in order to guarantee that an arriving packet will have a valid memory unit to be forwarded to.

Although the above observations have significant theoretical implications, the bottleneck preventing this technology from scaling is clearly the sequential nature of the memory management mechanism. We next address the inherent limitations imposed on any memory management algorithm for a range of PSM architectures. Let us begin our discussion by investigating an intuitively ideal scenario in which there exist N^2 shared memory units.In this case a sequential packet placement algorithm becomes trivial: an arriving packet from input port i destined to output port j with departure time τ is placed in one of the N memories (out of the N^2) associated with output port j. Hence, there is little decision making to be done on the part of the memory management algorithm. However, the requirement for N^2 memories limits the scalability of the switch with respect to port density, which suggests that an alternative approach to the packet placement algorithm should be explored.

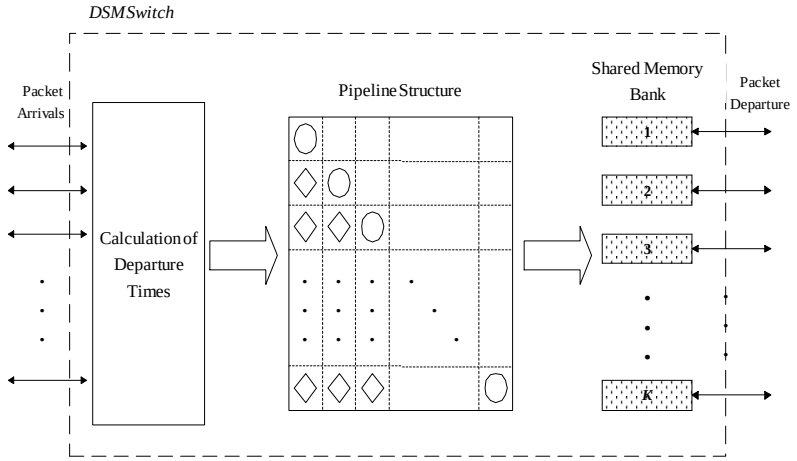

Figure 5-1: Pipelined packet placement algorithm. Time-space tradeoff is exploited to gain execution speed at the cost of a fixed latency of N time slots.

5.3 Pipelined Memory Management Algorithm Without Speedup

5.3.1 Pipelined Architecture

Consider an N-port PSM switch in which arriving packets progress through a pipelined structure prior to being placed into the shared memories, as illustrated in figure 5-1. Let the pipelined structure be represented as an $N \times N$ matrix, P. Columns of P contain packets that have arrived at the same time, while the rows correspond to the input port numbers. Packets arriving at time t are placed in the left-most (first) column of the pipeline matrix. Following each time slot, all columns are shifted one position to the right and the newly arriving packets are, once again, positioned in the first column. During each time slot, the algorithm will issue decisions (i.e. select one available memory for each packet) in parallel for all arriving packets.

Copying of packets from the pipeline structure to the shared memories occurs only for packets that are on the diagonal of P (marked by circles in figure 5-1). Before we elaborate on the rationale behind the placement of packets in the memories, we note that the preliminary processing stage in which packets are assigned departure times can be arbitrarily long, so long as it is deterministic. The respective delay simply adds a fixed

latency to the overall switching process.

Packets that arrive at the same time are copied to memories during different times (due to the diagonal placement approach). In order to make sure they depart at the correct relative time, a constant N is added to the calculated departure time of each packet. The latter guarantees similar identical departure times to packets arriving to the switch simultaneously at the cost of a fixed latency of N time slots. What remains to be described is the manner by which the diagonal packets in the pipeline structure are assigned to memory units. We define the term *availability map* of an arriving packet to denote the set of all valid memories to which this packet can be copied. The underlying concept governing the memory assignment process is that an identical availability map is generated for all incoming packets so as to guarantee that each packet is assigned a valid and non-conflicting memory.

5.3.2 Packet Placement Algorithm

In order to illustrate the main constrains imposed on the memory management algorithm and derive the minimal number of memories needed, we consider the extreme case in which all N arriving packets have identical departure times. Clearly, each of the packets is destined to a distinct output. Let's look at the conditions that need to be met in order for the placements to be valid and non-conflicting:

1. Even though packets with the same departure time are copied to the memories at different time steps (off the diagonal), each must select a distinct memory since they will be required to depart (be read from memory) at the same time in the future.

2. Each packet must not be assigned a memory unit which has been previously selected by other packets on its diagonal. This is to guarantee that N packets on the diagonal are each written to a different memory unit.

If a single and identical availability map is to be generated for all arriving packets, the number of memories needed is at least $N(N-1)/2$, corresponding to selections already made by packets residing in the lower triangle of the pipeline structure (excluding the first

column). Since all packets may have the same departure time, additional $N-1$ memories should be available.

Last, there should be a strict mechanism under which two or more packets with identical departure time select different memories (when referencing the similar availability map). The most straightforward way of achieving this goal is for packets arriving at input port i to select the i^{th} available memory. This can be easily implemented using a programmable priority encoder design, the computational complexity of which is $O(N)$. An extreme case is that in which the N^{th} (last) packet has a unique departure time, in which case $N-1$ additional memories are needed to guarantee no conflicts. In conclusion, the minimal number of memories required is given by

$$k > \underbrace{\frac{N(N-1)}{2}}_{\substack{pipeline \\ allocations}} + \underbrace{N-1}_{\substack{identical \\ departure\ time}} + \underbrace{N-1}_{\substack{N^{th}\ available \\ memory}} \tag{5.1}$$

$$= \frac{(N+4)(N-1)}{2}. \tag{5.2}$$

5.4 Pipelined Memory Management Algorithm With Speedup

The assumption thus far has been that the concurrent assignment of arriving packets to memories required an entire time slot (packet time) to complete. However, since the algorithm comprises simple, albeit large, binary operations over the availability map, it is reasonable to assume that the propagation delay of such operations is some fraction of a packet duration.

We let the processing speedup factor, η, denote the ratio of packet duration to a single packet assignment time. The reader will note that since N packets may be arriving at any given time slot, N decisions regarding packet-to-memory assignments must be made concurrently. If the latter does not hold, packets will inevitably be lost. Practical values of η may be as high as 8 if, for example, the assignment operation consumes ~6 $nsec$ while a packet time is ~50 $nsec$.We exploit this processing speedup by condensing the computation process and complexity of the pipeline architecture, as shown in figure 5-2. Here, each triangle shown in the figure corresponds to a step in the assignment process in which N/η

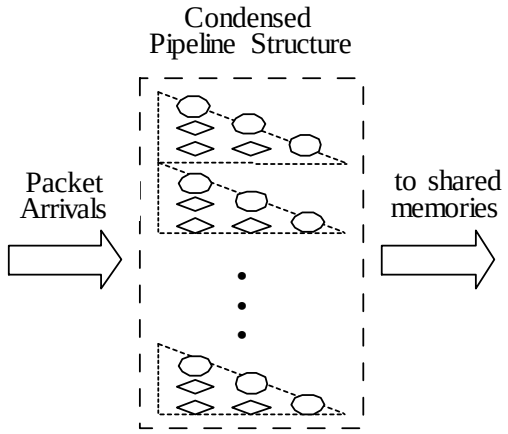

Figure 5-2: The pipelined packet placement algorithm with processing speedup ($s > 1$). The structure comprises of s triangular structures each corresponding to N/s time steps.

packets are assigned to memories.

In order to complete N assignments, η consecutive iterations are needed during each time slot. By partitioning the process in this manner, we achieve several goals. First, the latency introduced by the pipeline is reduced to N/η time slots, as can be observed by the reduction in the number of columns. Second, and more importantly, by following the same rationale as in (5.1) the number of overall memory units needed is reduced to $\frac{N}{2}\left(\frac{N}{\eta}+3\right)$.Note that for $\eta = N$ we have $k = 2N$, which corresponds to the result obtained in 5-1 when dual-port memories are employed in a sequential memory management algorithm. In addition, since the requirement from the memory access time is still one time slot, we may further exploit the fact that a packet may be written to a memory in a shorter period of time. Consequently, we may reduce the number of physical memories required by a factor of r to yield

$$k \geq \frac{N}{2r}\left(\frac{N}{\eta}+3\right). \tag{5.3}$$

Consideration of both contributing speedup factors yields a very low requirement for physical memory devices and on chip area.

Chapter 6

Conclusions

6.1 Summary of Dissertation Contributions

6.1.1 Heterogeneous Maximal-Throughput Bursty Traffic Model

In Chapter 1 we presented an analysis of a discrete time Markov-modulated packet arrival process for generating heterogeneous bursty traffic in input-queued switches. The goals of the model described were two-fold. First, it constitutes an expansion of the classic ON/OFF process to address scenarios in which multiple queues receive bursts from a shared input stream. Second, in concert with the first goal, it enables 100% link utilization (hence maximal traffic generation throughput) by avoiding the mandatory OFF slots introduced by classic ON/OFF models. This work is especially valuable in evaluating the performance of packet switching fabrics under bursty conditions.

6.1.2 Frame-Based Maximal Weight Matching

In chapter 3 we proposed and studied the frame-based maximal weight matching algorithm, with LQF and OCF priorities, as a scalable scheduling scheme for large port-density input-queued switches. Through the use of Lyapunov functions, it has been shown that a transfer speedup of 2 is sufficient to guarantee stability for architectures hosting both single and multiple classes of service. The need for transfer speedup, as opposed to the scheduling speedup considered in the literature, renders the approach highly attractive from an implementation

perspective. Moreover, the frame-based analysis presented here can be broadened to address a range of input-queued switching architectures and scheduling algorithms. Although these stability analysis is based on the unicast assumption, but for those IQ switch architectures based on internal copy networks so that multicast cells are replicated at inputs and treated like unicast cells, the results still holds as long as the admissibility of arrvial traffic is maintained.

6.1.3 Scalable Multi-Crosspoint Based Packet Switching Architecture

In Chapter 4, we developed a novel switch architecture that scale to multi-Terabit/sec, offers high-performance and retains implementation simplicity. The switch core can be put together using existing, off-the-shelf type products, rather than customized ASICs. We also Investigated frame-based scheduling algorithms that can govern the switch architecture proposed. This involved consideration of multi-class traffic (i.e. quality of service provisioning). We proved stability properties that clearly quantify the limitations of the approach. A detailed performance evaluation of the developed switch fabric is conducted, including applying a range of traffic models that include non-uniformly distributed destinations and bursty arrivals. We also conducted a comparison, spanning performance, implementability and scalability attributes, between the solution proposed and existing ones (e.g. parallel packet switch, load-balanced switch etc.) Finally, we investigated the above in the context of WDM-based packet switching..

6.1.4 Pipelined Memory Management Algorithm

In chapter 5, we have described a pipelined memory management algorithm for PSM switches that emulates FCFS output-queued switches. The proposed scheme exploits a time-space tradeoff to offer reduces computational complexity of the memory management process, thereby gaining scalability at the cost of fixed latency. Moreover, by considering memory access speedup, the number of physical shared memory units required is further reduced, yielding a highly scalable and pragmatic switch architecture.

6.2 Relevant Publications

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

- **X. Li**, I Elhanany, "Stability of a Frame-Based Oldest-Cell-First Maximal Weight Matching Algorithm", to appear in IEEE Transactions on Communications.
- **X. Li**, I. Elhanany, "Stability of a Frame-Based Maximal Weight Matching Algorithms with Transfer Speedup," *IEEE Communication Letters*, 2005.
- **X. Li**, I. Elhanany, "Stability of Frame-Based Maximal Weight Matching Algorithms with Reconfiguration Delays," *Proc. 2005 IEEE High-Performance Switching and Routing Symposium*, Hong-Kong, May 12-14, 2005.
- **X. Li**, I. Elhanany, "Heterogeneous Maximal-Throughput Bursty Traffic Model with Application to Input-Queued Switches," P*roc. 2005 IEEE Sarnoff Symposium,* April 18-19, Princeton, New Jersey.
- **X. Li**, I. Elhanany, "A Pipelined Memory Management Algorithm for Distributed Shared Memory Switches," P*roc. IEEE International Conference on Communications (ICC) 2005*, Seoul, Korea.
- M. J. McCollum, **X.Li**, I. Elhanany, "A Mutli-Stage Pipelined Memory Management Algorithm for Parallel Shared Memory Switches," *Proc. IEEE 48th Midwest Symposium on Circuits & Systems (MWSCAS 2005)*, Cincinnati, Ohio, August 2005.

Bibliography

Bibliography

[1] R. Y. Awdeh and H. T. Mouftah, "Survey of atm switch architectures," *Comput. Netw. ISDN Syst.*, vol. 27, no. 12, pp. 1567–1613, 1995.

[2] R. Melen and J. S. Turner, "Nonblocking multirate networks," *SIAM J. Comput.*, vol. 18, no. 2, pp. 301–313, 1989.

[3] Y.-M. Yeh and T. yun Feng, "On a class of rearrangeable networks," *IEEE Trans. Comput.*, vol. 41, no. 11, pp. 1361–1379, 1992.

[4] J.N.Giacopelli, J. Hickey, W.S.Marcus, W.D.Sincoskie, and M.Littlewood, "Sunshine: a high performance self-routing broadband packet switcharchitecture," *Selected Areas in Communications, IEEE Journal on*, vol. 9, no. 8, pp. 1289–1298, 1991.

[5] M.Karol, M.Hluchyj, and S.Morgan, "Input versus output queuing on a space division switch," *IEEE Transactions on Communications*, vol. 35, no. 12, pp. 1347–1356, Dec 1987.

[6] T.Anderson, S.Owicki, J.Saxe, and C.Thacker, "High speed switch scheduling for local area networks," *ACM Transactions on Computer Systems*, pp. 319–352, Nov 1993.

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

[8] Y.Tamir and G.Frazier, "High performance multi-queue buffers for vlsi communication switches," *15th Annual Symposium on Computer Architecture, Washington DC*, pp. 343–354, June 1988.

[9] Y.Tamir and H.C.Chi, "Symmetric crossbar arbiters for vlsi communication switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 1, pp. 13–27, Jan 1993.

[10] S.T.Chuang, A.Goel, N.McKeown, and B.Prabhakar, "Matching output queuing with combined input and output queuing," *IEEE JSAC*, vol. 17, no. 6, pp. 1030–1039, Dec 1999.

[11] G.Fayolle, "On random walks arising in queueing systems: ergodicity and transience via quadratic forms as lyapunov functions – part i," *Queueing Systems*, vol. 5, pp. 167–184, 1989.

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

[13] L.Tassiulas and A.Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, Dec 1992.

[14] I. Elhanany and M. Hamdi, *High-Performance Packet Switching Architectures*. Springer, 2007.

[15] C. Williamson, "Internet traffic measurement," *IEEE Internet Comput.*, vol. 5, pp. 70–74, Dec 2001.

[16] A. L. Corte, A. Lombardo, and G. Schembra, "Modeling superposition of on-off correlated traffic sources in multimedia applications," *IEEE INFORCOM 1995*, pp. 993–1000, 1995.

[17] I. Elhanany, D. Chiou, V. Tabatabaee, R. Noro, and A. Poursepanj, "The network processing forum switch fabric benchmark specifications: An overview„" *IEEE Communications Magzine*, 2003.

[18] G.K.Zipf, *Psycho-Biology of languages*. MIT Press, 1965.

[19] R.E.Willis, "Measuring scientific prose with rank frequency ('zipf') curves: A new use for an old phenomenon," *Proc. American Society for Information Science*, vol. 12, pp. 30–31, 1975.

[20] B.M.Hill, "A simple general approach to inference about the tail of a distribution," *Anals. of Statistics*, vol. 13, pp. 1163–1174, 1975.

[21] E.A.Dinic, "Algorithms for a solution of a problem of maximum flow in a network with power estimation," *Soviet Math. Dokl.*, vol. 11, pp. 1277–1280, 1970.

[22] J.E.Hopcroft and R.Karp, "An n2.5 algorithm for maximum matching in bipartite graphs," *Soc. Ind. Appl. Math. J. Computation*, vol. 2, no. 4, pp. 225–231, Dec 1973.

[23] N.Mckeown, "Scheduling algorithms for input-queued switches," *PhD Thesis, The University of California at Berkeley*, 1995.

[24] N.Mckeown, V. Anantharam, and J. Walrand, "Achieving 100input-queued switch," *Proc. IEEE INFOCOM ąŕ96*, pp. 296–302, Mar 1996.

[25] L.Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," *Proc. IEEE INFOCOM ąŕ98*, pp. 533–539, Apr 1998.

[26] A.Mekkittikul and N.McKeown, "A starvation-free algorithm for achieving 100throughput in an input-queued switch," *Proceedings of ICCCN'96*, pp. 226–231, October 1996.

[27] R.E.Tarjan, "Data structures and network algorithms," *Bell Laboratories*, 1983.

[28] A. Mekkittikul and N. McKeown, "A practical scheduling algorithm to achieve 100792–799, April 1998.

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

[30] J. Dai and B. Prabhakar, "The throghput of data switches with and without speedup," *IEEE INFOCOM 2000*, pp. 556–564, March 2000.

[31] Z.Tao and S.Cheng, "A new way to share buffer — grouped input queueing in atm switching," *Proc. IEEE GLOBECOM ąŕ94*, vol. 1, pp. 475–479, 1994.

[32] A. Marsan, M.G.A.Bianco, and E.Leonardi, "Rpa: A simple, effi-cient, and flexible policy for input buffered atm switches," *EEE Commun. Letters*, vol. 1, no. 3, pp. 83–86, 1997.

[33] A. Marsan, M. A. Bianco, E. Leonardi, and L.Milia, "Quasi-optimal algorithms for input buffered atm switches," *Proc. Third IEEE Symposium on Computers and Communications ISCC qŕ98*, pp. 336–342, 1998.

[34] D. Stiliadis and A. Varma, "Providing bandwidth guarantees in an input- buffered crossbar switch," *Proc. INFOCOM qŕ95*, pp. 960–968, 1995.

[35] N.Mckeown, "The islip scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 1, no. 2, pp. 188–201, 1997.

[36] R.Z.S.Iyer and N.Mckeown, "Analysis of the parallel packet switch architecture," *IEEE/ACM Transactions on Networking*, vol. 11, no. 2, pp. 314–324, 2003.

[37] ——, "Routers with a single stage of buffering," *ACM Computer Communication Review SIGCOMM '02*, pp. 251–264, 2002.

[38] L.G.Valiant, "A scheme for fast parallel communication," *SIAM Journal on Computing*, pp. 350–361, 1982.

[39] C. Chang, D. Lee, and Y. Jou, "Load balanced birkhoff-von neumann switches," *High Performance Switching and Routing, 2001 IEEE Workshop*, pp. 276–280, 2001.

[40] I. Keslassy, R. Zhang-shen, and N. McKeown, "Maximum size matching is unstable for any packet switch," *IEEE COMMUNICATIONS LETTERS*, vol. 7, no. 10, pp. 496–498, October 2003.

[41] P. Kumar and S. Meyn, "Stability of queueing networks and scheduling policies," *IEEE Transactions on Automatic Control*, vol. 40, no. 2, pp. 251–260, February 1995.

[42] S.Y.Nam and D.K.Sung, "Decomposed corssbar switches with multiple input and output buffers," *IEEE GLOBECOM 2001*, vol. 4, pp. 2661–2665, 2001.

[43] X.Li and I.Elhanany, "Stability of a frame-based maximal weight matching algorithm with transfer speedup," *IEEE Communications Letters*, vol. 9, no. 10, pp. 942–944, 2005.

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

[45] D. Sadot and I. Elhanany, "Optical switching speed requirements for terabit/sec packet over wdm networks," *IEEE Photonic Technology Letters*, vol. 12, no. 4, pp. 440–442, April 2000.

[46] I.Keslassy, "The load-balanced router„" *Ph.D. Thesis, Stanford University*, 2004.

[47] I. Elhanany and B. Matthews, "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, 2006.

[48] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 344–357, June 1993.

[49] J.Bennett and H.Zhang, "Wf2q: Worst-case fair weighted fair queueing," *IEEE INFOCOM '96*, pp. 120–128, Mar 1996.

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

[51] S.K.A.Demers and S.Shenker, "Analysis and simulation of a fair queueing algorithm," *ACM Computer Communication Review SIGCOMM '89*, pp. 3–12, 1989.

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

Vita

Xike Li was born in Changsha, People's Republic of China, on September 17, 1975. After finishing high school in 1993, he attended Beijing University of Posts and Telecommunications, Beijing, P.R.China, where he received a Bachelor of Engineering degree in 1997. Between 1997 and 2001, he worked as a software engineer in SED electronics group co., Ltd. and Huawer Technologies co., Ltd., Shenzhen, P.R.China. In 2001, he came to study at the University of Tennessee, Knoxville, United States. He received a Master of Science degree in electrical engineering in Spring 2004 and a Doctor of Philosophy degree in computer engineering in Fall 2006, both from the department of Electrical and Computer Engineering at the University of Tennessee, Knoxville.