ON PACKET SWITCH SCHEDULING IN HIGH-SPEED DATA NETWORKS

ON PACKET SWITCH SCHEDULING IN HIGH-SPEED DATA NETWORKS

BY
AMIR GOURGY, B. ENG.(MCMASTER), M.A.SC.(WATERLOO)
MARCH 2006

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF MCMASTER UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DOCTOR OF PHILOSOPHY (2006)
(Electrical and Computer Engineering)

McMaster University
Hamilton, Ontario

TITLE:	On Packet Switch Scheduling in High-Speed Data Networks
AUTHOR:	Amir Gourgy, B. Eng.(McMaster), M.A.Sc.(Waterloo)
SUPERVISOR:	Dr. Ted H. Szymanski, Professor of Electrical and Computer Engineering, B.A.Sc., M.A.Sc., Ph.D. (University of Toronto)
NUMBER OF PAGES:	xvi,136

Abstract

There is a tremendous demand for Internet core nodes to provide quality-of-service (QoS) guarantees for multimedia services, and to provide high switching capacity that makes use of the virtually unlimited bandwidth of optical fibers. The Internet's success depends on the deployment of high-speed switches and routers that meet these two demands. We address theoretical and practical aspects of packet switch scheduling in high-speed data networks.

First, we address short-term fairness in QoS scheduling for input-queued (IQ) switches. We show that existing practical scheduling algorithms for Internet routers with IQ switches are unfair over short time scales and potentially lead to increased jitter. Subsequently, we present a scheduling policy based on credit-based fair queueing that provides better short-term fairness in QoS scheduling than existing solutions with comparable complexity. A flow-based iterative credit-based fair scheduler (iCBFS) is proposed for crossbar switches, that provides fair bandwidth distribution among flows at a fine granularity and achieves asymptotically 100% throughput, under uniform traffic. To reduce the implementation complexity of iCBFS, we present a port-based version of iCBFS that is tailored towards high-speed hardware implementation.

Second, we address the problem of fair scheduling of packets in Internet routers with IQ switches and unity speedup. Scheduling in IQ switches is formulated as *tracking* the behaviour of an output-queued (OQ) switch that provides optimal performance. We present the notion of "lag" as a performance metric that measures the difference between a packet's departure time in an IQ switch over that provided by an OQ switch. We prove that per packet mean lag is bounded for a maximum weight matching scheduling policy that uses lag values for its weights and derive a

bound on the mean lag value using a Lyapunov function technique. Furthermore, we propose a simple heuristic tracking scheduling policy and evaluate its performance by simulation.

Finally, we present a novel distributed scheduling paradigm for Internet routers with IQ switches, called Cooperative Token-Ring (CTR) that provides significant performance improvement over existing scheduling schemes with comparable complexity. In classical token-ring based scheduling for IQ switches, a separate token ring (an arbiter) is used to resolve contention for each shared resource (i.e., an output port). Although classical token-ring based scheduling achieves fairness and high throughput for uniform traffic, under non-uniform traffic the performance degrades significantly. We show that by using a simple cooperative mechanism between the otherwise non-cooperative token rings (arbiters) the performance can be significantly improved and the scheduler is able to dynamically adapt to any non-uniform traffic pattern. To provide adequate support for rate guarantees in IQ switches, we present a Weighted Cooperative Token-Ring (WCTR), a simple hierarchical scheduling mechanism. Finally, we analyze the hardware complexity introduced by the proposed CTR scheduling and describe an optimal hardware implementation for an $N \times N$ switch implementing a CTR scheduler. We show that the hardware time complexity introduced by the proposed Cooperative mechanism is $\Theta(\log N)$.

Acknowledgments

I would like to thank God who helped me finish this work and used many people to help me and guide me along the way.

I would like to express my sincere gratitude and thanks to my supervisor Dr. Ted Szymanski for his meticulous supervision of this work during the last four years. I enjoyed his unrelenting enthusiasm and passion for exploring new ideas and his unparalleled breadth and depth of knowledge in many research areas that helped me finish this work. My Ph.D. committee members Drs. Douglas Down and Terry Todd provided guidance and support.

Special thanks to all my colleagues and office mates who provided a lively working environment: Honglin Wu, Zhiwei Mao, Ying Yang, Kalyan Bhattacharyya, Houman Homayoun, Nader Fahmy, and Ehab Anis.

The administrative team of the Electrical and Computer Engineering Department deserves my gratitude for their endless help in many matters, especially Cheryl Gies and Helen Jachna.

Finally, I thank my family for their continual unconditional support to myself all the time.

I acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant and an Ontario Center of Excellence - Communications and Information Technology (OCE-CITO) Research Grant.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

There is a tremendous demand for Internet core nodes to provide quality-of-service (QoS) guarantees for multimedia services, and to provide high switching capacity that makes use of the virtually unlimited bandwidth of optical fibers. The Internet's success depends on the deployment of high-speed switches and routers that meet these two demands. This work addresses switch scheduling in high-speed data networks and proposes several techniques to provide high switching capacity and QoS guarantees.

This chapter is organized as follows: Section 1.2 provides an overview of the architecture and components of an Internet router; various switch fabric architectures are discussed in Section 1.3; the organization and contributions of this dissertation are described in Section 1.4; a list of acronyms used in this dissertation is provided in Section 1.5.

1.2 Overview of IP Router Architecture

A block diagram of typical router is shown in Figure 1.1. The blocks in Figure 1.1 can be broadly partitioned into two groups based on the functionality: datapath and control plane.

The control plane of the router include system configuration, management, and

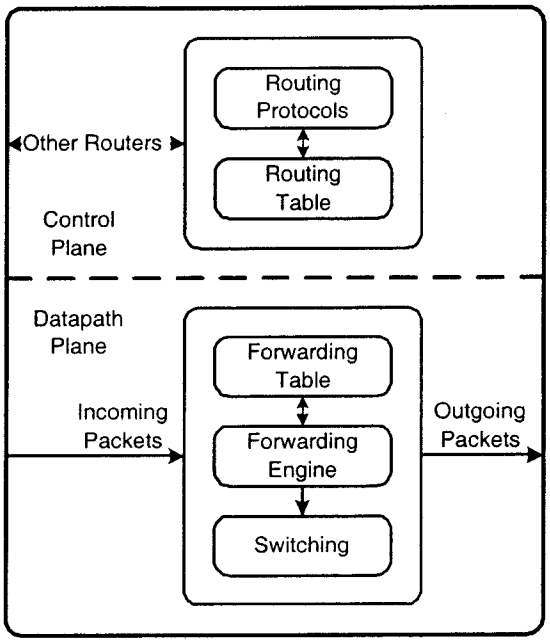


Figure 1.1: Router Block Diagram.

exchange of routing table information. These operations are performed relatively infrequently and are typically implemented in software.

The datapath operations are performed on every packet that passes the router. These functions include forwarding decisions, switching through the router's switch fabric. When a packet arrives at the forwarding engine, its destination IP address is looked up from the forwarding table (using a longest-prefix match) and the packet header is updated accordingly. Subsequently, the packet header is used to determine the packet's output port where the packet is routed through the switch fabric. An overview of switch fabric architectures is provided in Section 1.3. A scheduler is used to resolve contention among packets that are destined to the same output port in fair manner or according to their priority levels. The scheduling problem is described in detail in Chapter 2.

The packet forwarding process operates at the granularity of an entire IP packet,

which can be of variable-length, whereas switching is performed on smaller fixed-length granularity cells such that a *segmentation and reassembly* process is used. In a packet-switch, segmentation and reassembly (SAR) is the process of breaking a variable-length packet into smaller fixed-length *cells* before transmission through the switch fabric and reassembling them into the proper order at the switch's output. This segmentation and reassembly simplifies the switch's design considerably and allows the implementation of transmission rates up to several Gbps per link in VLSI hardware. We assume fixed-length cell scheduling in this work – the words *packet* and *cell* are used interchangeably for the remainder of this dissertation.

1.3 Switch Fabric Architectures

In this section we provide an overview of the main switch fabric architectures. For the purposes of this dissertation, we classify switch fabric architectures based on their queueing discipline as it is a key factor in determining the switch's performance. We describe the following switch fabric architectures: output-queued, shared memory, buffered crossbar, and input-queued.

1.3.1 Output-Queued Switch Fabric

In an output-queued (OQ) switch fabric all cells are buffered in the egress side of the switch fabric as shown in Figure 1.2, and there is a constant delay for all cells through the fabric. This queueing discipline requires that the switch fabric works N times faster than the cell rate on the line card. In addition, the buffers in egress port should support up to N writes and 1 read every time slot.

The major drawback of the OQ switch fabric is that it requires a *speedup* of N. A switch with a speedup of S can remove up to S packets from each input and deliver up to S packets to each output within a *time slot*, where a time slot is the time between packet arrivals at an input port. Unfortunately, for large or for high-speed data lines, memories with sufficient bandwidth are not available.

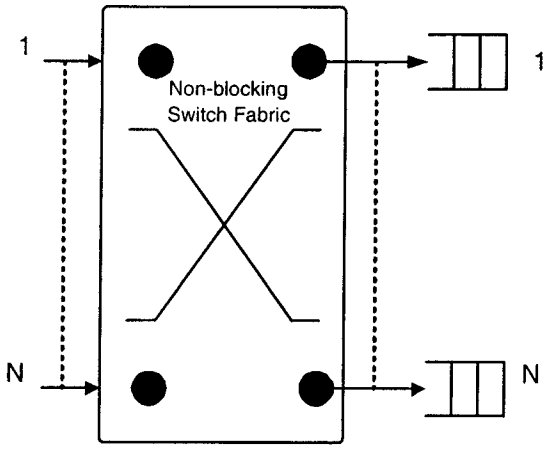


Figure 1.2: Basic architecture of an output-queued switch.

1.3.2 Shared Memory Switch Fabric

The shared memory switch fabric uses shared memory as the switch fabric such that all the inputs and output ports have access to a shared memory as shown in Figure 1.3. On the one hand, the shared memory switch fabric provides an efficient implementation of output queueing and provides optimal throughput and delay performance. On the other hand, the switch aggregate capacity ($N \times$ link speed) is limited by the memory read/write access time, within which N incoming and N outgoing cells in a time slot need to be accessed as shown in equation 1.1

$$\text{Switch Capacity} \leq \frac{\text{cell length}}{2 \times \text{memory access cycle}} \tag{1.1}$$

For instance, with a cell length of 200 bytes and a memory access cycle of 5 ns, the switch capacity is limited to 160 Gbps

1.3.3 Buffered Crossbar Switch Fabric

In the buffered crossbar architecture a distributed array of buffers is used such that a buffer is used at each crosspoint of the crossbar as shown in Figure 1.4. The objective here is to maintain output ports as busy as possible by keeping the crosspoint buffers

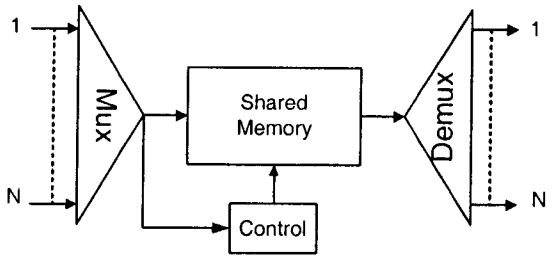


Figure 1.3: Basic architecture of an $N \times N$ shared-memory switch fabric.

non-empty. This architecture distributes contention resolution among all inputs and outputs: Independent input schedulers select packets to move from the line cards to the buffered crossbar; and independent output arbiters select packet from among all packets in the crossbar destined to the same output. This results in a simpler, more distributed implementation of $2N$ (N input schedulers, and N output schedulers) schedulers each with a complexity of $\Theta(N)$ instead of one centralized scheduler with a complexity of $\Theta(N^2)$.

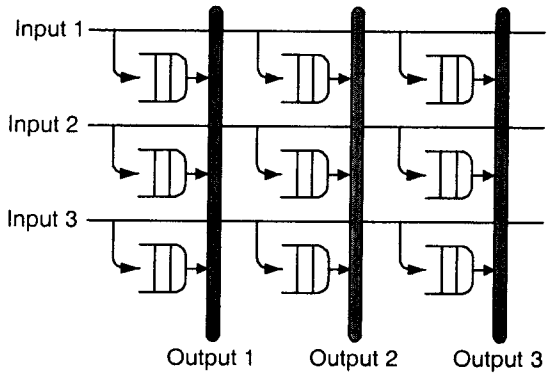


Figure 1.4: Basic architecture of a 3×3 internally buffered crossbar switch fabric.

The main drawback of the internally buffered crossbar architecture is that a separate buffer is required at each crosspoint resulting in N^2 buffers per priority class. Additionally, each crosspoint has a *fixed* amount of buffer space that is not shared among other inputs or outputs. Furthermore, this architecture requires flow control mechanisms integral to the datapath of the crossbar itself.

1.3.4 Input-Queued Switch Fabric

In the input-queued (IQ) switch arriving cells are buffered at the input before being routed to the their intended destination as shown in Figure 1.5. A crossbar is typically used as the non-blocking switch fabric.

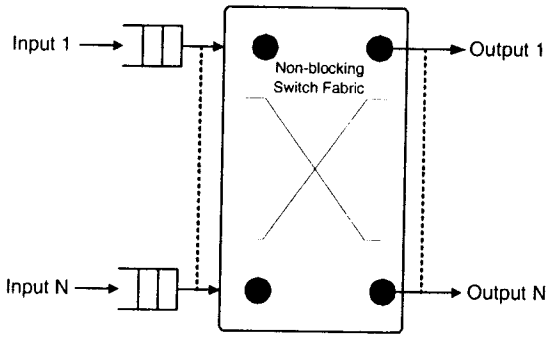


Figure 1.5: Basic architecture of an $N \times N$ input-queued switch fabric.

Unlike an OQ switch where a cell from each input can be simultaneously transmitted to the same output, the set of cells that can be transmitted from inputs to outputs in an IQ switch must satisfy the so-called *crossbar constraint*: During every time-slot, each output can only accept data from a single output, which must concurrently be transmitting data only to that output.

Input-queueing has the advantage that the bandwidth requirement on each input-queue is proportional only to the port speed and not to the number of switch ports as only one read and one write operation is required per packet cycle on each queue. For an $N \times N$ IQ switch with a port speed of B, the total bandwidth requirement is $2B$ per input queue.

A scheduler is used during each time-slot to arbitrate among the head-of-line packets of each queue to be switched to the outputs. The scheduler must satisfy the crossbar constraint, provide fairness, and achieve good performance. Scheduling in IQ switches is explored in detail in Chapter 2. In essence, designing a scalable scheduler is the main challenge of this architecture and is the main topic addressed in this dissertation.

1.4 Dissertation Organization and Contributions

This dissertation is organized as follows:

This chapter introduces the problem domain of high-speed switching networks, provides some background on switching, and summary of the main contributions of this dissertation.

Chapter 2 provides detailed background and related work on the current state-of-the-art in IQ scheduling.

In chapter 3 we present a scheduling algorithm for Internet routers with IQ switches based on credit-based fair queueing that provides better short-term fairness in QoS scheduling than existing solutions with comparable complexity. First, we present a flow-based iterative credit-based fair scheduler (iCBFS), for crossbar switches, that provides fair bandwidth distribution among flows at a fine granularity and achieves asymptotically 100% throughput, under uniform traffic. To reduce the implementation complexity of iCBFS, we present a port-based version of iCBFS that is tailored towards high-speed hardware implementation.

Chapter 4 presents a theoretical framework for evaluating the performance of IQ switches, and introduces the "lag" concept as a performance metric that measures the difference between a packet's departure time in an IQ switch over that provided by an OQ switch that provides optimal performance. We present several scheduling policies that use the "lag" as a performance metric and assess their performance using analytic and simulation methods.

In Chapter 5 we present a novel distributed scheduling paradigm for Internet routers with IQ switches, called Cooperative Token-Ring (CTR) that provides significant performance improvement over existing scheduling schemes with comparable complexity. In classical token-ring based scheduling for IQ switches, a separate token ring (an arbiter) is used to resolve contention for each shared resource (i.e., an output port). Although classical token-ring based scheduling achieves fairness and high throughput for uniform traffic, under non-uniform traffic the performance degrades significantly. We show that by using a simple cooperative mechanism between the otherwise non-cooperative token rings (arbiters) the performance can be significantly

improved and the scheduler is able to dynamically adapt to any non-uniform traffic pattern. To provide adequate support for rate guarantees in IQ switches, we present a Weighted Cooperative Token-Ring (WCTR), a simple hierarchical scheduling mechanism. Finally, we analyze the hardware complexity introduced by the proposed CTR scheduling and describe an optimal hardware implementation for an $N \times N$ switch implementing a CTR scheduler. We show that the hardware time complexity introduced by the proposed cooperative mechanism is $\Theta(\log N)$.

Chapter 6 summarizes the results and contributions of this work and provides directions for future work.

1.5 List Of Acronyms

This section lists acronyms used in this dissertation:

ATM Asynchronous Transfer Mode
BGM Bipartite Graph Matching
CMOS Complementary Metal-Oxide
CTR Cooperative Token-Ring
EiSLIP Exhaustive iSLIP
EM Exhaustive Matching
FCFS First-Come First-Served
Gbps Gigabits per second
FIFO First-In First-Out
GBps Gigabytes per second
HoL Head-of-Line
iCBFS iterative Credit-Based Fair Queueing
iDRR iterative Deficit Round Robin
IDT Ideal Departure Time
iLag Iterative Lag
IP Internet Protocol
IQ Input Queue/Queued/Queuing
iSLIP Iterative SLIP

LPF Longest Port First
LQF Longest Queue First
MWL Maximum Weighted Lag
OQ Output-Queued
PIM Parallel Iterative Matching
QoS Quality of Service
RR Round Robin
RRM Round-Robin Matching
TRV Token Request Vector
TRP Token Request Path
WCTR Weighted Cooperative Token-Ring
WDM Wavelength-Division Multiplexing
WFQ Weighted Fair Queuing
WPIM Weighted Parallel Iterative Matching
VOIP Voice Over Internet Protocol
VOQ Virtual Output Queue(d/ing)

Chapter 2

Background and Related Work

Most commercial high-performance switches and routers (e.g., CISCO 1200[Cis04], BBN [PCB+98]) employ IQ switches because the fabric and the memory of an IQ switch need to run only as fast as the line rate. This bandwidth memory requirement makes input queueing very appealing for switches with fast line rates or with a large number of ports. In this chapter we describe general background and related work on arbitration for IQ switches.

This chapter is organized as follows: We describe the head-of-line (HOL) blocking problem in IQ switches in Section 2.1 and present a known architecture technique to reduce HOL blocking in Section 2.2. A general classification of arbitration algorithms based on graph theory is described in Section 2.3. A paradigm for hardware implementation of arbitration algorithms is described in Section 2.4. We discuss randomized scheduling algorithms in Section 2.6. Finally, we discuss arbitration based on matrix decomposition and load balancing in Section 2.7.

2.1 Head-of-Line Blocking

A well known problem in pure IQ switches with first-in-first-out (FIFO) input buffers is the *head-of-line* (HOL) blocking problem. This problem occurs when cells are blocked from reaching a free output port because other cells that are ahead of it in the FIFO buffer. As shown in Figure 2.1, the cell behind the HOL cell at input port

1 is destined for an idle output port 2, but it is blocked by the HOL cell.

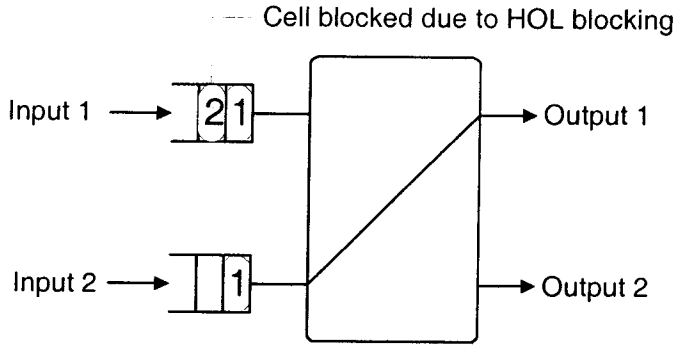


Figure 2.1: Head-of-line blocking in an IQ Switch.

Karol et al. [KHM87] have shown that FIFO input-queueing limits the switch's throughput to a theoretical maximum of merely $2 - \sqrt{2} \approx 58.6\%$ of maximum bandwidth under the assumption of uniform Bernoulli independent identically distributed (i.i.d.) traffic. For correlated arrival traffic, the throughput decreases to 50% [Li92].

One solution that has been proposed to reduce HoL blocking is to use virtual output queueing as described in the next section.

2.2 Virtual Output Queueing

Virtual output queuing (VOQ) architecture is commonly used for reducing HOL blocking such that each input maintains a separate queue for each output [TF88]. For an $N \times N$ switch, there are a total of N^2 queues such that each queue uses a FIFO scheduling policy and a scheduler is used to select a single packet from each input as shown in Figure 2.2. $Q_{i,j}$ is the queue used at input i to store packets destined for output j.

In the VOQ architecture, incoming cells are queued at the input ports and a scheduling algorithm configures the fabric during each time slot and decides which inputs will be connected to which outputs. In an $N \times N$ switch the scheduler examines the contents of N^2 virtual output queues and determines a conflict free match between

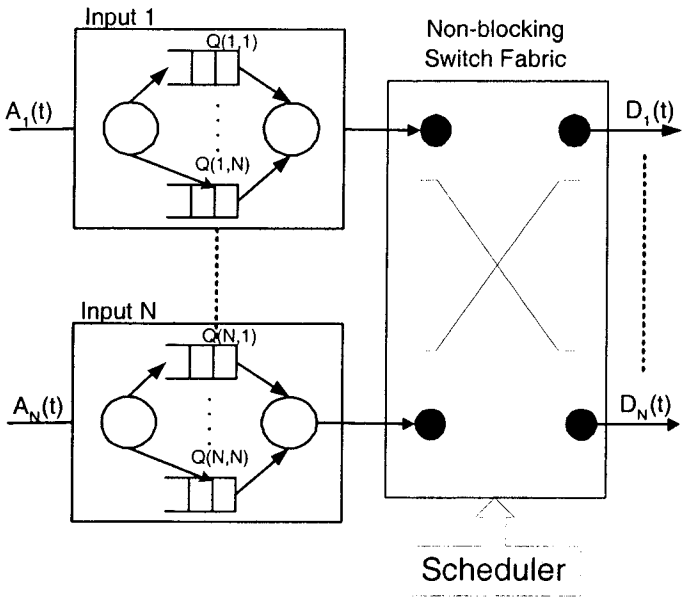


Figure 2.2: Virtual Output Queueing in an IQ Switch.

inputs and outputs.

In an IQ switch there are essentially two shared resources: the switch fabric and the outgoing link. Arriving packets are queued at the input port of the switch and they must first contend for access to the switch fabric (input contention), before contending for the outgoing link (output contention) as shown in Figure 2.3

In essence the role of the scheduler in an IQ switch is to resolve input and output contention *fairly* and *efficiently*. In the next sections we describe various techniques for resolving this contention with diverse tradeoffs between the implementation cost and the achieved performance.

2.3 Bipartite Graph Matching

In an $N \times N$ IQ switch using VOQ, the scheduler determines in every time slot a conflict free match between inputs and outputs; that is each input is matched to at most one output and conversely each output is matched to at most one input.

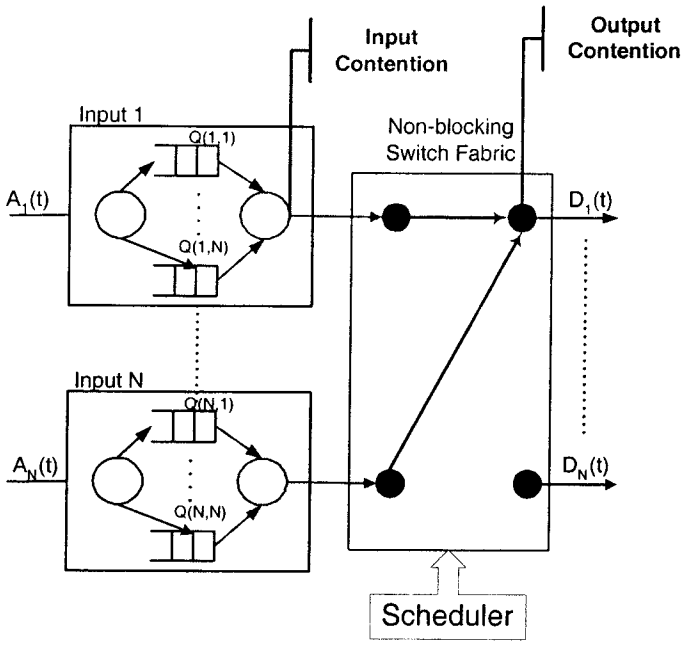


Figure 2.3: Input and Output Contention in an IQ Switch.

This scheduling problem is equivalent to finding a bipartite graph matching (BGM) [Tar83].

The graph corresponding to the scheduling problem has N source vertices that correspond to the N inputs of the switch, and N sink vertices that correspond to the outputs. An input and and output are connected by an edge if the corresponding VOQ is not empty– there are at most N^2 edges between source and sink vertices. In each time slot, a graph $G = (V, E)$ that consists of a set V of $2N$ vertices, partitioned into two sets, namely N inputs and N outputs as shown in Figure 2.4 (a). At time slot n, the set of edges E has each edge connecting vertex i from the inputs set to vertex j for which $Q_{i,j}(n) > 0$; that is, for each non-empty VOQ the graph has a corresponding edge.

A matching $\mathcal{M}$ on this graph $\mathcal{G}$ is any subset of E such that no two edges in $\mathcal{M}$ have a common vertex; that is, at most one packet is transferred from each input and at most one packet is received at each output. Specifically, let $S_{i,j}(n)$ be a binary

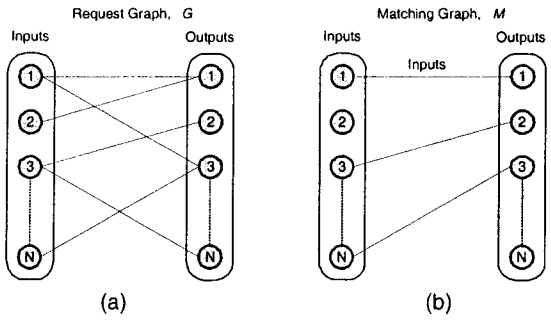


Figure 2.4: IQ Scheduling modeled as a bipartite graph matching problem: (a) Request Graph, $\mathcal{G}$. (b) Computed Matching, $\mathcal{M}$.

service indicator such that $S_{i,j}(n) = 1$ if input i is matched to output j at time slot n. A matching $\mathcal{M}$ must satisfy that $\sum_{i=1}^{N} S_{i,j}(n) \leq 1$ and $\sum_{j=1}^{N} S_{i,j}(n) \leq 1$

There are various BGM algorithms with a tradeoff between their performance and implementation complexities [ADH98]. BGM algorithms can be broadly classified into three categories: maximum size matching, maximum weighted matching, and maximal matching, which are described in the next sections.

2.3.1 Maximum Size Matching

A maximum size Matching $\mathcal{M}$ on a graph $\mathcal{G}$ is one that maximizes the number of edges in the matching; that is, $M = Max(\sum_{i,j} S_{i,j}(n))$. The sequential time complexity of the maximum size matching is $\Theta(N^{2.5})$ [Din70].

Although using maximum size matching maximizes the instantaneous throughput by transferring the maximum number of cells during each time slot, it was shown that it could lead to starvation of some queues [MMAW99] and *instability*, even under admissible traffic, and for any switch size [KM03]. A queue is said to be unstable [KM03] if after a finite time, its occupancy never returns to zero with probability one.

2.3.2 Maximum Weighted Matching

A maximum weight matching (MWM) on a bipartite graph with weighted edges is defined as a set of edges between input and output nodes with the maximum total weight among all possible sets satisfying the constraint that any input node is matched to at most one output node. At every time slot n, a weight $W_{i,j}(n)$ is associated to every edge in the request graph, $\mathcal{G}$; the maximum weighted matching finds a matching $\mathcal{M}$ that maximizes $\sum_{i,j} W_{i,j}(n)S_{i,j}(n)$ and can be found by solving an equivalent network flow problem [AMO93].

Several algorithms have been proposed based on MWM that use different functions to assign edge weights (e.g., the queue-length, the waiting time of the HOL packet, priority, or any other combination).

The sequential run time complexity of MWM is $\Theta(N^3 \log N)$[AMO93], [Tar83], [MMAW99], which makes MWM prohibitively expensive to implement in hardware. Instead, most practical algorithms are based on simple heuristics that aim at maximizing the number of connections between inputs and outputs to achieve maximal size matching as described next.

2.3.3 Maximal Size Matching

A maximal matching is defined as one in which no edges can be added to it without first removing an existing edge. The sequential time complexity of maximal matching is $\Theta(N^2)$ on a sequential model, which is lower than both maximum size and maximum weight matching. Most practical implementations of high-speed packet switches aim at achieving a maximal matching (e.g., iSLIP[McK99], WPIM[SV95], iDRR[ZB03], iFair[NB02], HSA[BDEA04]). The main distinction between the various maximal matching based algorithms is the heuristic used for selecting the edges that are added to the matching.

Most maximal matching algorithms are implemented in hardware using an iterative mechanism, which is described in Section 2.4, such that up to N inputs can be matched in each iteration, and the computation performed per iteration is $\Theta(N)$. In the worst case, N iterations are required to converge to a maximal matching; however,

in practice only a fixed number of iterations are performed.

Dai and Prabhakar proved that a speedup of 2 is *sufficient* to provide 100% throughput for any maximal matching based algorithm [DP00]. Conversely, Mneimneh et al. [MS03] proved that at *least* a speedup of two is required for maximal matching algorithm to provide 100% throughput; that is, the bound on the speed up of two is tight.

2.4 Implementing Maximal Size Matching Using Parallel Matching Algorithms

A simple paradigm that is commonly employed in implementing maximal matching is using an input arbiter at each input port to resolve input contention and an output arbiter at each output port to resolve output contention such that a maximal match is achieved by iteratively matching inputs to outputs. Specifically, two schemes can be classified under this paradigm: 2-phase, and 3-phase matching with different implementation tradeoffs. Initially, all the inputs and outputs are not matched. A 3-phase algorithm comprises the following phases:

1. Request: Each unmatched input arbiter sends a request to every output arbiter for which it has a queued cell.

2. Grant: Each unmatched output

   arbiter resolves output contention by choosing only one of the input requests and sends back a grant signal to the input port.

3. Accept: Each input arbiter resolves input contention by choosing only one of the received grants and sends back an accept signal to the corresponding output arbiter. The input and output arbiter are considered matched.

The previous phases are repeated until either a maximal matching is found or a fixed number of iterations are performed.

Anderson et al. [AOST93] proposed parallel iterative matching (PIM), a 3-phase algorithm, and used *random* selection at each input and output arbiter. Although finding a maximal matching using PIM may, in the worst case, take N iterations, it was shown that [AOST93] under *uniform i.i.d.* traffic, the algorithm converges to a maximal match in $\Theta(\log(N))$ iterations; however, for a single iteration the throughput is limited to approximately 63% for uniform i.i.d. traffic.

Iterative round-robin matching (iRRM) [MVW93] works similarly to PIM, but uses round-robin schedulers instead of random selection at both inputs and outputs. McKeown proposed iSLIP [McK99] as an improvement over PIM and iRRM. iSLIP uses rotating priority round-robin arbiters. Under uniform traffic, the pointers used in the input and output arbiters for selection (i.e., grant and accept selection) tend to point to different elements (*desynchronize*) such that each arbiter tends to make different selection from other arbiters and the largest number of inputs and outputs are matched. Consequently, under uniform Bernoulli i.i.d. traffic iSLIP arbiters adapt to a time-division multiplexing scheme, providing a perfect match and 100% throughput [McK99]. However, under non-uniform traffic, the pointers are not necessarily desynchronized and the performance potentially degrades.

In a 2-phase algorithm [Cha00] each input arbiter sends at most one request; subsequently, it receives at most one grant signal, and the accept phase is not needed; for example, dual round robin (DRR) [Li04] performs the following 2-phases:

1. Request: Each input sends an output request corresponding to the first nonempty VOQ in a fixed round-robin order, starting from the current pointer position. The pointer remains at that nonempty VOQ if the selected output is not granted in the second phase.

2. Grant: If an output receives one or more requests, it chooses the one that appears next in a fixed round-robin schedule starting from the current pointer position. The output notifies each input whether or not its request was granted. The pointer of the output arbiter is incremented to one location beyond the granted input. If there are no requests, the pointer's position does not changes.

One the one hand, a 2-phase algorithm requires less communication and is simpler to implement than a 3-phase; on the other hand, a 3-phase algorithm tends to converge to a maximal matching faster than a 2-phase algorithms. Consequently, with the same number of iterations, a 3-phase algorithm usually provides a better performance. For simplicity, we refer to all scheduling schemes based on either 2-phase or 3-phase matching paradigm as ΠRGA.

Figure 2.5 shows a typical interconnection of $2N$ arbiters to implement a 3-phase matching for an $N \times N$ switch. Each arbiter is usually implemented using a rotating round-robin priority encoder that operates in $\Theta(\log N)$ time.

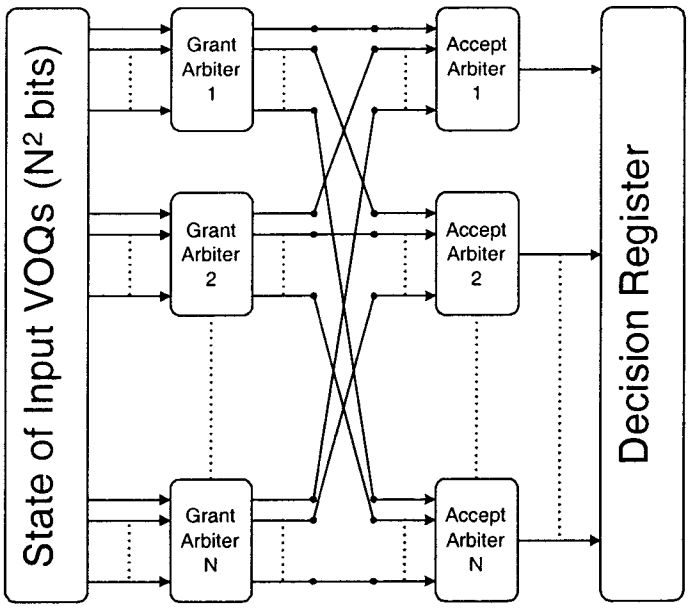


Figure 2.5: Hardware Interconnection Structure of 3-Phase Matching.

2.5 Exhaustive Matching

Li et al. [LPC02] proposed coupling ΠRGA paradigm with *exhaustive matching*(EM). In EM, after an input is matched to an output using ΠRGA, a VOQ is served continuously until it becomes empty; that is, EM attempts to achieve high throughput by

maintaining the current match between inputs and outputs as long as possible, and performing a match between the unmatched input and output ports when needed. This scheme tends to perform well under non-uniform bursty traffic, and has the potential of amortizing the cost of arbitration time over multiple time slots. Simulations results for different variations of exhaustive matching algorithms are reported in [KC03] using a 2-phase and 3-phase matching where it was shown that exhaustive iSLIP (EiSLIP) produces the best results among various proposed exhaustive matching algorithms (e.g., exhaustive dual round-robin matching, exhaustive PIM, etc.).

2.6 Randomized Scheduling Algorithms

Randomized scheduling algorithms have been proposed for IQ switches [Tas98] in an attempt to simplify the scheduling problem and provide fairness. The basic idea of randomized scheduling is to select the *best* matching from a set of random matches. The best matching is defined to be the matching with the maximum weight. Various weight functions have been proposed and analyzed in the literature [GPS03].

Goudreau et al. [GKR00] proposed using a randomized technique that addresses the problem of low throughput under non-uniform traffic for ΠRGA schedulers, the *Shakeup* scheduling policy. In the shakeup scheduling policy, after a maximal matching is computed (say using any ΠRGA scheduler) each unmatched input is allowed to force a match for itself *randomly* even though an existing match has to be knocked off. The argument for this scheme is that by randomly breaking matches, adding new ones the algorithm will escape the "local minima" of a maximal matching and probabilistically converges to a *maximum size match.* Although theoretically sound, no analysis has been made on the number of iterations required for converging, and its implementation feasibility is unknown. Furthermore, given the randomized nature of the algorithm, it is not clear how it can be extended to provide rate guarantees. In other words, Shakeup attempts to find the global maximum, but the feasibility of its implementation in a real system is unknown because it takes more iterations to converge, mostly because all the selections are done randomly.

2.7 Birkhoff von Neumann Switches

Chang et al. [CCH01] formulated IQ scheduling as matrix decomposition that can be performed using Birkhoff von Neumann decomposition, which decomposes a doubly substochastic matrix into a convex combination of (sub)permutation matrixes. This decomposition is based on the assumption that an arrival rate matrix is given that specifies the arrival rates between every input-output pair.

The off-line sequential computational complexity to compute the permutation matrices is $\Theta(N^{4.5})$ and the number of permutation matrices is $\Theta(N^2)$. Subsequently, the on-line sequential computational complexity to schedule the permutation matrices is $\Theta(\log N)$.

Although this scheme provides 100% throughput with unity speedup, it requires: explicit knowledge of all the arrival rates; the need to store all the computed permutation matrices, which does not scale for large N; and the need to perform complex calculations when arrival rates change. In addition, because the decomposition algorithm uses long term traffic arrival statistics, it does not adapt too well to dynamic traffic fluctuations. Although other decomposition methods have been proposed that reduce the number of permutation matrices, in general, they do not provide good throughput; for example, the throughput for the decomposition scheme in [KKLS03] is $\Theta(1/\log N)$, which tends to 0 for large N.

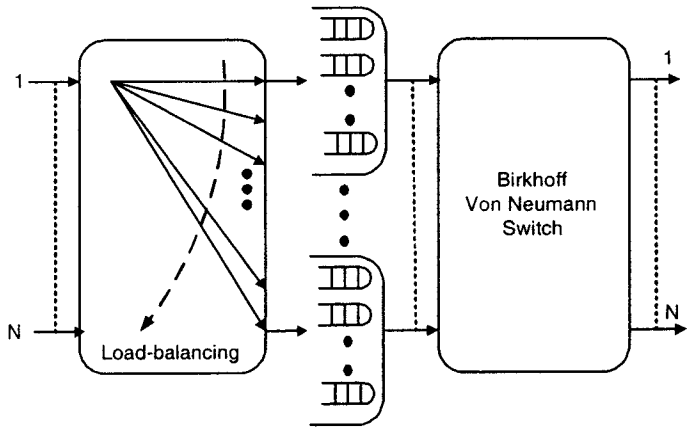


Figure 2.6: Architecture of load-balanced Birkhoff-von Neumann switch.

Load-balanced Birkhoff-von Neumann [CLJ02] switches attempt to trivialize the arbitration process. A load-balanced routes consists of two stages as shown in Figure 2.6. First, a load balanced stage spreads arriving packets equally among line cards. Then, a forwarding stage transfers packets from the line cards to their intended destination. In addition to the delay incurred by using a 2-stage switch, the main drawback of this architecture is that packets can be missequenced, which may require complicated hardware implementation and non-scalable computation overhead. Providing a scalable solution that can simultaneously provide QoS support and solve the packet missequencing problem is the major difficulty in the load-balanced router architecture.

In this chapter we discussed background and related work on arbitration for IQ switches. We described the head-of-line blocking problem in pure IQ switches and showed how the virtual output queueing architecture combined with a matching algorithm could reduce HOL blocking. We presented a classification of matching into maximum weighted, maximum size, and maximal size matching and compared their performance tradeoffs. Finally, we discussed other scheduling approaches that includes exhaustive matching, randomized scheduling, and Birkhoff-von Neumann switches.

Chapter 3

Credit-based Fair Scheduling in Input-Queued Switches

We present a novel QoS scheduling algorithm for Internet routers with IQ switches based on credit-based fair queueing. We present a flow-based iterative credit-based fair scheduler (iCBFS), for crossbar switches, that provides fair bandwidth distribution among flows at a fine granularity and achieves asymptotically 100% throughput, under uniform traffic. To reduce the implementation complexity of iCBFS, we present a port-based version of iCBFS that is tailored towards high-speed hardware implementation. We show by simulation that iCBFS provides better fairness than existing schedulers in the literature, with comparable hardware complexity.

3.1 Introduction

Although several practical scheduling algorithms such as iDRR [ZB03], WiSLIP [McK99], and WPIM [SV95] (described in the next section) have been proposed for IQ switches to provide QoS guarantees, these algorithms provide bandwidth guarantees over coarse granularity (i.e., a frame) and exhibit unfairness over short time scales. Specifically, these schemes are fair only over timescales longer than a *frame size*, where the frame size is one round-robin of service over all backlogged flows in the switch such that all backlogged flows are served in proportion to their reservations. Over timescales less than a frame, these schedulers do not serve flows in proportion to their reservations and flows can be served in any arbitrary order. Although the

aggregate bandwidth received by a flow over the entire frame is in proportion to its reservation, within a frame time some flows may not get any service until the very end of the frame and bandwidth distribution over the frame time is nonuniform. Furthermore, as the switch size increases, the number of queues in the switch increases and the frame size becomes larger; thereby, this unfairness leads to increased jitter, which is undesirable for multimedia services like VOIP. It is this problem that our proposed scheduler solves. We emphasize that this problem can not be solved by using a smaller frame size because the frame size is limited by the resolution of the minimum allocatable fraction of bandwidth per flow; for example, consider a future core router with link speeds of 100 Gbps. For a flow to reserve only 10 Mbps, or 0.01 percent of the link capacity, the frame size needs to be at least 10000 time slots.

Bensaou et al. [BTC01] have proposed credit-based fair queueing for OQ switches. In this chapter, we propose a scheduling algorithm for IQ switches based on credit-based-fair-queueing [BTC01], called iterative credit based fair scheduling (iCBFS). Our simulation results show that iCBFS provides fair bandwidth distribution among flows bandwidth at a fine granularity, and solves the unfairness for timescales smaller than a frame size; thereby our algorithm provides better short-term fairness than existing schemes, with comparable hardware complexity. In addition, iCBFS achieves asymptotically 100% throughput, under uniform traffic. Note that the short-term fairness problem addressed by iCBFS is orthogonal to the CTR presented in Chapter 5. In essence, the iCBFS scheme presented in this chapter could be combined with the WCTR presented in Section 5.8 to provide rate guarantees and short-term fairness.

This chapter is organized as follows. Section 3.2 provides a review of related work on scheduling for IQ switches. Section 3.3 discusses fairness in IQ schedulers, presents our proposed flow-based scheduler (iCBFS), and compares its performance to other scheduling schemes. In section 3.5, we propose a port-based version of (iCBFS) that is tailored towards efficient high-speed hardware implementation. Finally, section 3.7 concludes this chapter and summarizes our contributions.

3.2 Related Work on QoS Scheduling in IQ Switches

McKeown [McK99] proposed "weighted iSLIP" (WiSLIP) as a variation of iSLIP that can allocate bandwidth nonuniformly to different inputs. The bandwidth from input i to output j is given by the ratio $f_{ij} = \frac{n_{ij}}{d_{ij}}$, where n_{ij} is reservation for the $input_i$–$output_j$ pair, and d_{ij} is the aggregate reservation for output j. Instead of each arbiter maintaining an ordered circular list $S = 1, \ldots, N$ as in iSLIP, the list is expanded in WiSLIP at output j to the ordered circular list $S_j = 1, \ldots, W_j$, where W_j=lowest common multiple d_{ij} and input i appears $\frac{n_{ij}}{d_{ij}} \times W_j$; that is, the size of the circular changes based on the reservation values.

Stiliadis [SV95] proposed weighted PIM (WPIM) that allocates output bandwidth among inputs based on reservations made during an admission control phase. In WPIM, the time axis is divided into frames with a fixed number of slots per frame (e.g., a frame is typically 1000 slots [SV95]). The reservation unit is slot/frame. Consequently, WPIM provides bandwidth guarantee at a coarse granularity of a frame size.

Ni and Bhuyan [NB02] proposed a fair scheduling algorithm for IQ switches called iFS, which is based on virtual time. In iFS, each output link maintains a fair queueing engine, which assigns a virtual time to every incoming packet based on bandwidth reservation of the packet's flow. The incoming packet is then queued in a FIFO input buffer on a per flow basis. The algorithm then executes a maximal matching scheme based on virtual time, where only the grant and accept stages are executed.

On the one hand, by using virtual-time stamps for every incoming packet, iFS [NB02] can honour bandwidth reservations at a very fine granularity better than most existing schemes; on the other hand, the cost of this algorithm is the increased complexity in implementing N virtual-time based fair queueing engines.

A major problem with virtual-time-based approaches is the time stamp overflow. Because time stamp is an increasing function of the time that depends on a common virtual clock, which in turns reflects the value of the time tag of previously served packets, the virtual clock cannot be reinitialized to zero until the system is completely empty and all sessions are idle, which although statistically finite can be extremely

long, given that most real-communication traffic exhibits self-similar patterns. This may easily cause an overflow in the time tag unless special hardware algorithms are used [CJGL99]. Floating-point units are usually used in computing the virtual-time stamp. In addition, virtual-time-based approaches require that packets be sorted according to their time tags by the fair queueing engine. In iFS, every incoming packet needs to be assigned a virtual time-stamp and inserted into a sorted list. Therefore, for practical implementation of iFS, a very high-speed fair queueing hardware engine needs to be designed to compute virtual time-stamps in floating point, perform sorting, and be able to process up to packets during each time slot. These requirements are expensive to implement in hardware.

To overcome the complexity of using a virtual-time-based fair-queuing engine at each output, and assigning a virtual time-stamp to each incoming packet, Zhang and Bhuyan [ZB03] proposed iDRR, a ΠRGA scheduling scheme based on deficit round-robin. In iDRR, each input and output maintain a circular list such that inputs and outputs are matched in round-robin based on a quota value assigned by deficit-round-robin engines, in proportion to their reservations. Each matched input-output pair may transfer packets until it uses its uses its available quota or there are no more packets to transfer. A port-based version of iDRR, called iPDRR was also proposed in [ZB03] along with its hardware implementation.

3.3 A Flow-based Fair Scheduling Algorithm

First, a definition of fairness in IQ switch scheduling is presented. Second, we describe the architecture of our proposed flow-based iterative credit-based fair scheduler (iCBFS). Third, we present iCBFS algorithm in detail. Fourth, we evaluate the performance of iCBFS using various traffic models, and compare its fairness to WiSLIP, WPIM, and iDRR.

3.3.1 Definition of Fair Scheduling

We assume a work-conserving IQ switch, and use a definition of fairness similar to [NB02] and [ZB03]. Let $flow_k(i,j)$ denote the kth flow from input i to output j with

bandwidth reservation S_k, and $W_k(t_1, t_2]$ be the amount of $flow_k(i, j)$ traffic served in the interval $(t_1, t_2]$. Two flows $flow_M(i_1, j_1)$ and $flow_N(i_2, j_2)$ are in contention if $i_1 = i_2$ or $j_1 = j_2$; that is, in an IQ switch, there are essentially two shared resources: the crossbar, where flows at each input contend (input contention); and the bandwidth of each outgoing link, where flows destined to the same output link contend (output contention). For any two backlogged flows $flow_M(i_1, j_1)$ and $flow_N(i_2, j_2)$ that are in contention, a scheduling scheme is ideally fair in $(t_1, t_2]$ if $\frac{W_M(t_1,t_2]}{S_M} = \frac{W_N(t_1,t_2]}{S_N}$

That is, contending flows are served in proportion to their reservations. This definition of fairness, of course, holds only in an idealized fluid flow network. When the network is more realistic and serves the traffic flows by a nonnegligible quantum of variable size (packet by packet), the definition of fairness can be written as

$$\left|\frac{W_M(t_1, t_2]}{S_M} - \frac{W_N(t_1, t_2]}{S_N}\right| \leq B$$

where B is a bound that gives a measure of fairness, also called fairness index [Gol94]. The smaller the fairness index, the fairer is the scheduling algorithm.

3.3.2 Architecture of iCBFS Switch

The basic architecture of iCBFS is shown in Figure 3.1; for each output link we maintain a credit-based fair queueing (CBFQ) arbiter and a separate queue is used for each flow at the input ports. The scheduling algorithm is based on a ΠRGA policy.

The basic idea of iCBFS is to assign each flow a counter that gets incremented in proportion to the flow's reservation such that when the counter reaches a certain threshold value, its corresponding flow is flagged as a *candidate*, and is allowed to transmit a packet across the switch; subsequently, the counter is decremented after transmission. These counters are maintained by the CBFQ arbiters and are used to fairly resolve output contention as described in the next section.

In addition to the counters used by CBFQ arbiters, each input arbiter tracks the aggregate reservation from its port to all outputs, and uses another set of counters to track the aggregate number of packets transmitted to each output. These input

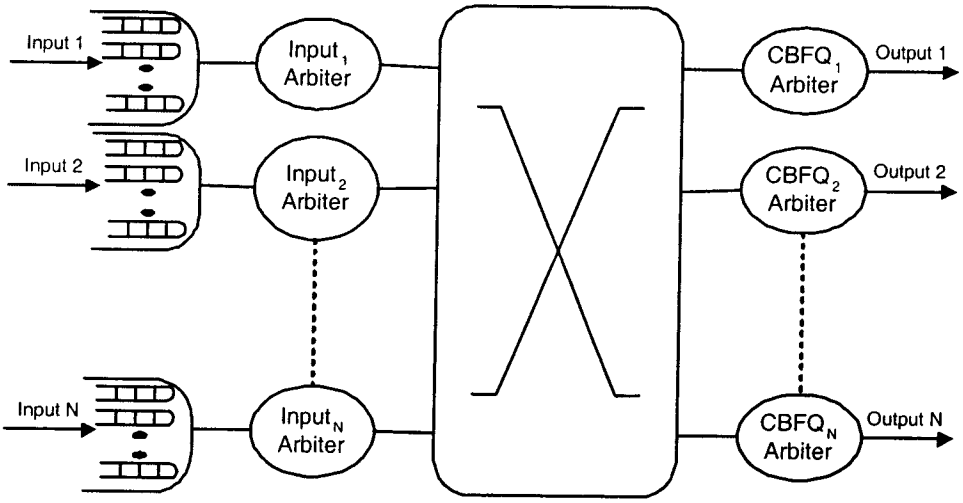


Figure 3.1: Architecture of a flow-based credit-based fair queueing switch.

arbiters' counters are used to fairly resolve input contention as described in the next section. All counters can be updated independently in parallel, which suits efficient hardware implementation.

Let the average packet arrival rate at input i for output j be λ_{ij}. The incoming traffic is called admissible if $\sum_{i=1}^{N} \lambda_{ij} < 1$, and $\sum_{j=1}^{N} \lambda_{ij} < 1$, We assume that flows' reservations are admissible.

3.3.3 Description of iCBFS algorithm

In an $N \times N$ switch, for each $flow_k(i,j)$ going from input i to output j, the input arbiter i uses a separate queue q_{jk}, and the $CBFQ_j$ arbiter maintains a counter K_{jk} and a bandwidth share S_{jk}. Let $Q_j = q_{j1}, q_{j1}, \ldots, q_{jJ}$ be the set of queues for flows $1, \ldots, J$ that are destined to output j, with bandwidth shares $S_{j1} \geq S_{j2} \geq \ldots \geq S_{jJ}$. Initially, all counters are set to zero. Each $CBFQ_j$ engine updates the counters as follows: $K_{jk} = K_{jk} + \frac{S_{jk}}{S_{j1}}$; that is, the backlogged queue with the largest share, at each CBFQ engine, is chosen as the reference queue to calculate a pro-rated share of bandwidth each backlogged queue should receive. K_{jk} is the accumulated credit for $flow_k$ destined to output j. Each $flow_k(i,j)$ with $K_{jk} \geq 1$ and $|q_{jk}| > 0$ is marked

as candidate and can be used during the iterative matching phase as described next.

In addition to the counters used by CBFQ arbiters, each input arbiter i uses a set of N counters $C_i = C_{i1}, C_{i2}, \ldots, C_{iN}$ such that C_{ij} indicates the current available quota of the $input_i$–$output_j$ pair. The quota is the number of reserved slots per frame for each $input_i$–$output_j$ pair. Let R_{ij} represent aggregate reserved bandwidth from $input_i$ to $output_j$. Each input arbiter i then assigns a quota to the counter values $C_{i1}, C_{i2}, \ldots, C_{iN}$ such that C_{ij} indicates the current available quota of the $input_i$–$output_j$ pair. These quotas can be either statically or dynamically reconfigured. In the static approach, a fixed minimum quota value (q_{min}) is assigned to the minimum possible aggregate reservation R_{min}. Subsequently, each aggregate reservation C_{ij} is assigned a quota $\frac{R_{ij}}{R_{min}q_{min}}$. In the dynamic approach, the value of q_{min} can be dynamically calculated based on the current flow with minimum reservation and the quotas of all other flows are calculated accordingly.

Initially, all inputs and outputs are unmatched. Then in each iteration:

1. Request: Each unmatched input sends a request to every output for which it has a queued packet.

2. Grant: If an unmatched output receives any requests, it chooses any candidate flow that belongs to an unmatched input and send a grant to this flow at its corresponding input. Note that counters are updated if there are no candidate flows for any of the requests.

3. Accept: If an unmatched input receives any grants, it chooses the flow with the largest quota for its counter. Note that selecting the flow with the largest quota resolves input contention in fair and simple manner.

In each time slot, for every selected flow, the switch transfers a packet of its head-of-line (HOL) queue. The input arbiter decrements the quota by 1 and the output arbiter decrements the flow's counter value by 1. The previous algorithm executes until either no more matches can be made or for a fixed number of iterations.

To circumvent flows from overusing or underusing their reservations, we require all quotas and counters be reinitialized after some period of time. For simplicity, we assumed a fixed frame size (e.g., 1000 slots) after which all the counters are initialized.

3.4 Simulation Results

First, the switch is set such that different flows have different reservations and the throughput per flow is measured to evaluate the fairness of the scheduler. Second, the switch setting is such that all flows have equal reservations and the performance is measured for a 16×16 switch under uniform Bernoulli i.d.d. traffic. The number of iterations was fixed to 4. The performance of iCBFS is compared to WiSLIP, iDRR, and WPIM.

3.4.1 QoS Traffic Model

To illustrate the fairness of iCBFS in bandwidth allocation, a 4×4 switch was simulated such that each input has four flows, each going to a different output with a different bandwidth reservation. Let $f_k(i,j)$ represent flow k from input port i to output port j. In the simulated switch, $f_1(0,0)$, $f_2(1,0)$, $f_3(2,0)$, $f_4(3,0)$ have reserved 10, 20, 30, and 40 percent of the bandwidth, respectively, but they always maintain the same actual arrival rate. Other flows have a load of 5 percent each. This traffic model has been used in [NB02] and [SV95]. We used equivalent switch settings for iCBFS, iDRR, and WPIM with equivalent frame size of 1000 slots. Figures 3.2, 3.3, 3.4, 3.5 shows the throughput per flow using iCBFS, iDRR, WiSLIP, and WPIM, respectively, after 750 time slots. The value of 750 represents 75 percent of the frame size and was chosen to illustrate the short-term unfairness problem present in other schemes and the superiority of iCBFS in solving this problem.

3.4.1.1 ICBFS vs. iDRR

Both iCBFS and iDRR were simulated with $q_min = 50$ slots and $r_{min} = 5\%$ with static counter initialization after 1000 slots. Although iDRR [ZB03] avoids the complexity of the virtual-time approach used in iFS, it does so at the expense of other performance metrics such as delay and fairness. iDRR possesses all the deficiencies inherent in deficit-round-robin service, namely that it is fair only over time scales longer than frame, and it has unbounded delay (the delay depends on local switch settings that can be arbitrarily large; see [GVC96] p. 3). As shown in Figure 3.3, at the rightmost

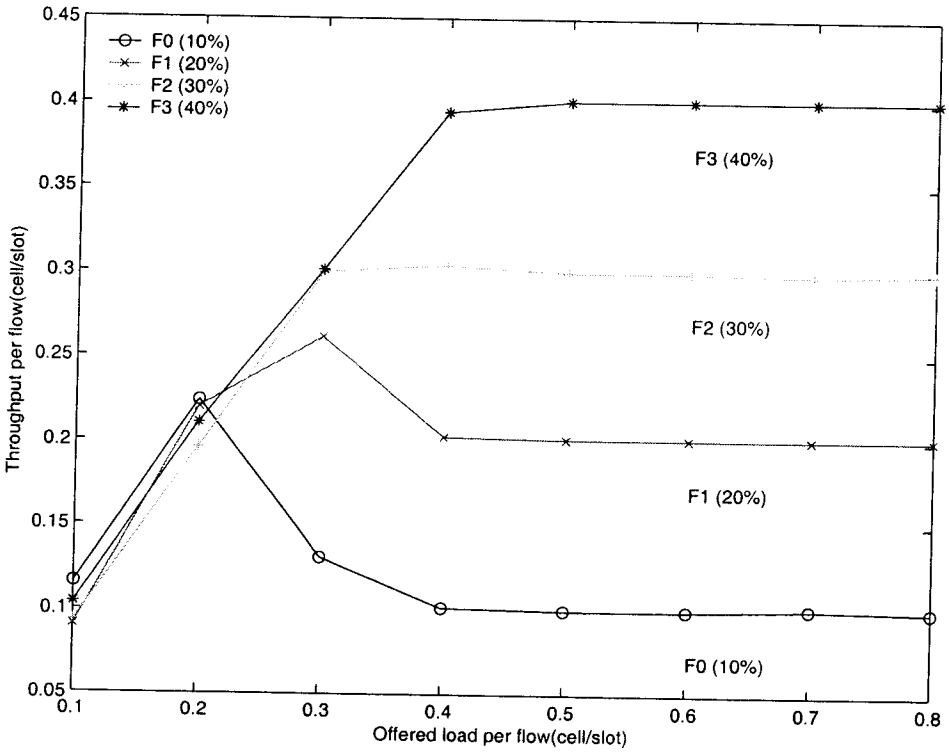


Figure 3.2: Throughput per flow using iCBFS.

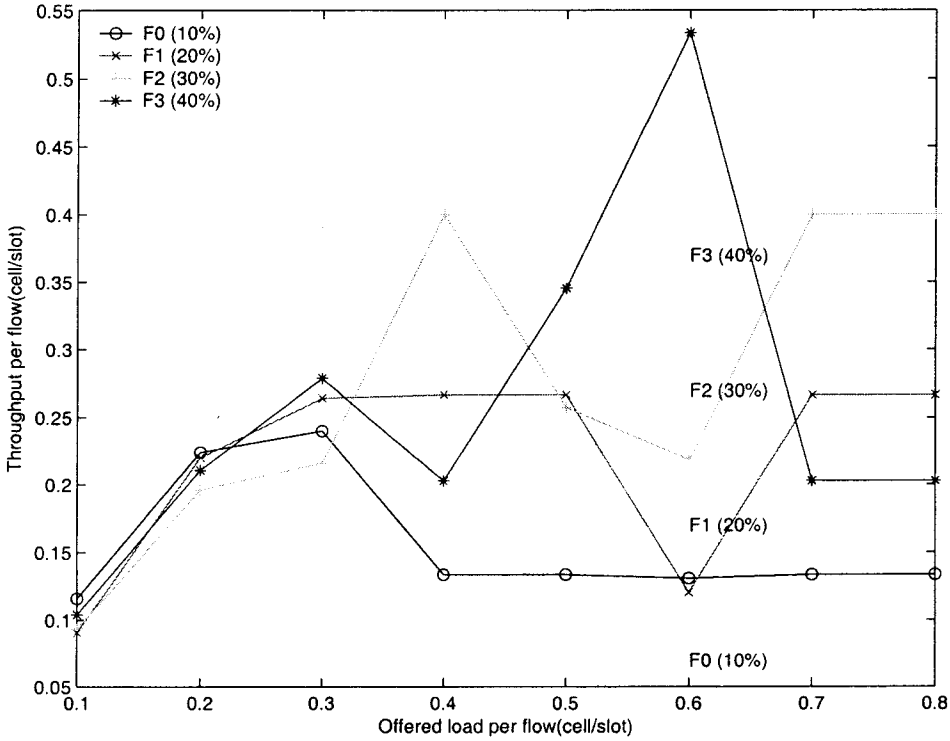


Figure 3.3: Throughput per flow using iDRR.

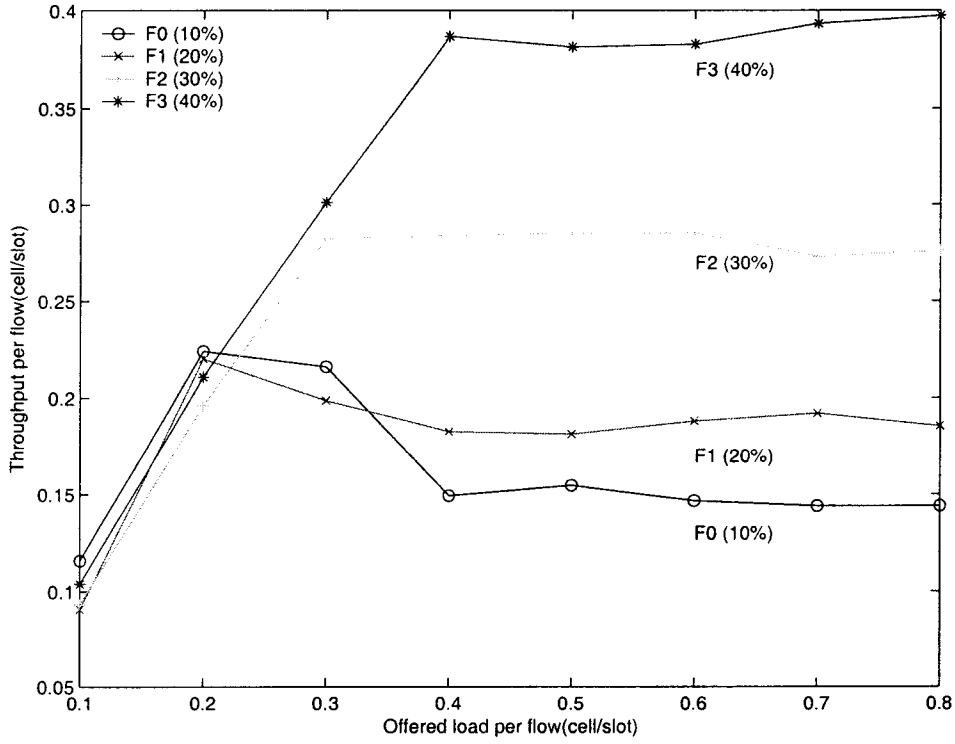


Figure 3.4: Throughput per flow using WiSLIP.

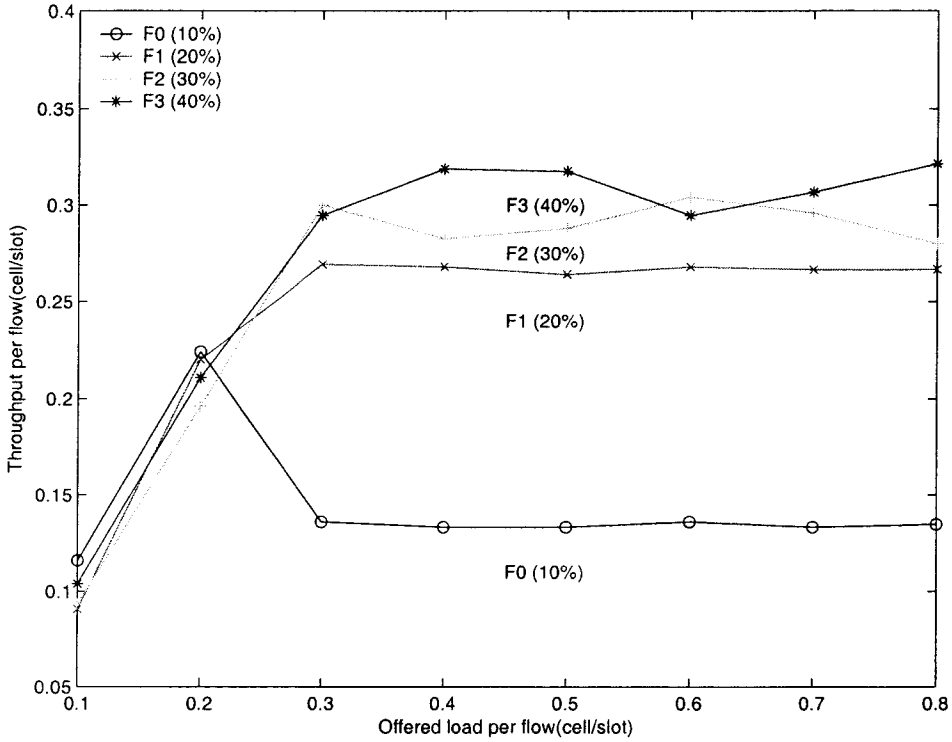


Figure 3.5: Throughput per flow using WPIM.

point of the graph, $f_1(0,0)$, $f_2(1,0)$, $f_3(2,0)$, $f_4(3,0)$ and receive 13, 26, 40, and 20 percent of the bandwidth, respectively. Specifically, $f_4(3,0)$ receives only half of its reserved bandwidth share leading to a large delay and jitter. In contrast, iCBFS is able to precisely allocate the bandwidth among the flows in proportion to their reservation.

3.4.1.2 IPCBFS vs. WiSLIP

As shown in Figure 3.4 at the rightmost point of the graph, WiSLIP does not precisely allocate bandwidth among flows in proportion to their reservations; $f_1(0,0)$ receives 15 percent of the bandwidth instead of its reserved 10 percent. Consequently, both $f_2(1,0)$ and $f_3(2,0)$ receive only 18% and 27% instead of 20% and 30%, respectively. In contrast, iCBFS is able to precisely allocate the bandwidth among the flows in proportion to their reservation. We identify the unfairness in iSLIP and its variant WiSLIP as caused by the simple operation of the rotating round-robin priority arbiters–the output arbiters do not track precisely how much bandwidth each input port uses. Specifically, iSLIP and all its variants [McK99] use simple rotating round-robin priority arbiters at each output arbiter with a pointer g_i to the current highest priority input of the round-robin schedule. This pointer g_i is only incremented (modulo N) if, and only if, the grant signal is accepted in the first iteration of the algorithm. For all subsequent iterations, the pointer is not updated even if a granted input is accepted. Although this scheme elegantly eliminates starvation in both iSLIP and its variants, it leads to impreciseness in tracking the bandwidth allocated to each input port (see [McK99] for a detailed explanation regarding the pointer update and the starvation problem). In addition, as the switch size increases the number of elements at each output arbiter's circular list increases and these elements can be positioned in any order. Consequently, the time required to serve all elements in the list will increase and the short-term unfairness will manifest itself clearly.

3.4.1.3 IPCBFS vs. WPIM

Although WPIM is fair over a time scale larger than the frame size (typically 1000 slots [SV95]), it is unfair over shorter time scales. As shown in Figure 3.5 at the

rightmost point of the graph,$f_1(0,0)$, $f_2(1,0)$, $f_3(2,0)$, and $f_4(3,0)$ receive 13, 26, 28, and 32 percent of the bandwidth, respectively. This unfairness is caused by the uniform random selection used at the output arbiters. In essence, all flows with available credit are treated equally until their credit is used up. Consequently, flows with higher bandwidth reservations than others receive their differential bandwidth share only at the end of a frame, whereas iCBFS distributes this differential bandwidth share uniformly over the entire time scale.

When all flows use their reserved credits, WPIM reduces to PIM and all unreserved bandwidth is distributed equally among all inputs [SV95], whereas iCBFS distributes unreserved bandwidth among all inputs in proportion to their reservations.

In summary, iCBFS provides fair bandwidth among flows in proportion to their reservations. iCBFS provides significantly better fairness than WiSLIP, WPIM, and iDRR over time scales less than a frame size. We emphasize that as the switch size increases, the frame size required to serve all the input ports increases proportionally and the short-term unfairness problem manifests itself clearly in increased jitter. The simple case of a 4×4 switch was only used to simplify the presentation. In addition, as the link speed increases the frame size would also increase.

3.4.2 Uniform Traffic

In addition to providing fair bandwidth among flows in proportion to their reservations, we evaluated the performance of iCBFS when all flows have equal reservations. Figure 3.6 shows the average delay of iCBFS compared to iSLIP, WPIM, and iDRR under uniform i.i.d. Bernoulli traffic. Similar to other scheduling schemes, iCBFS is capable of achieving asymptotically 100% throughput under uniform traffic. However, this traffic model is not realistic for Internet routers, which are usually non-uniform.

3.4.3 ON/OFF Markov-Modulated Arrivals

Figure 3.7. shows the average delay for iCBFS compared to iDRR, WiSLIP, and WPIM under an ON/OFF Markov Modulated Process with geometric burst size of 16. This traffic model is described in detail in [For04].

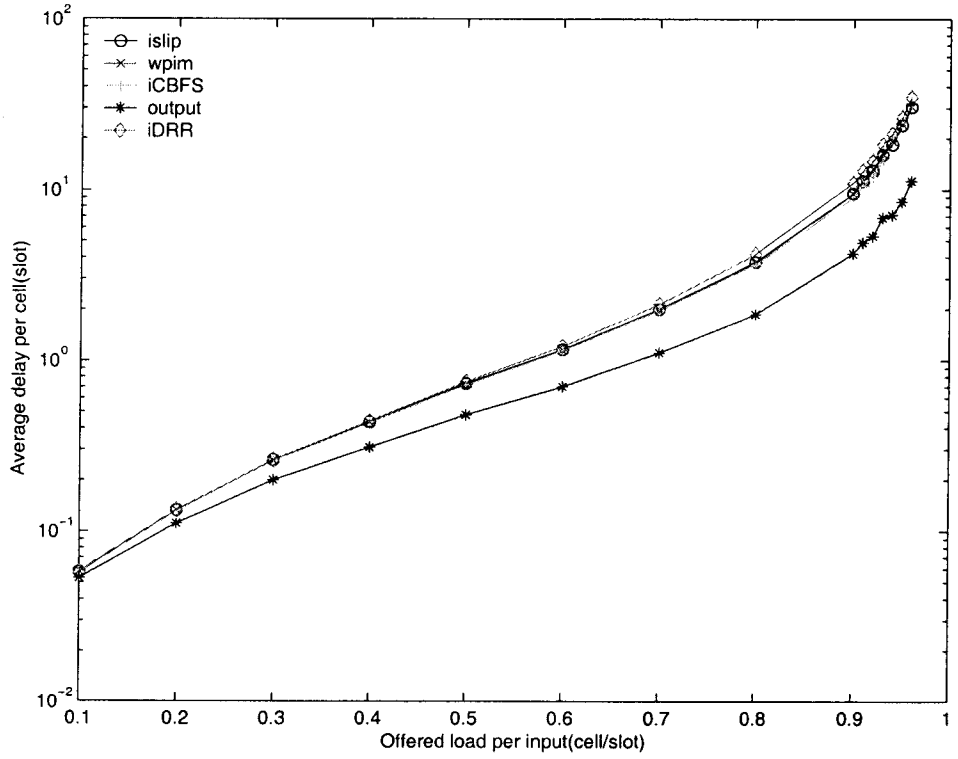


Figure 3.6: Average Delay of iCBFS, iSLIP, WPIM, and Output-Queueing under uniform Bernoulli i.i.d. Traffic.

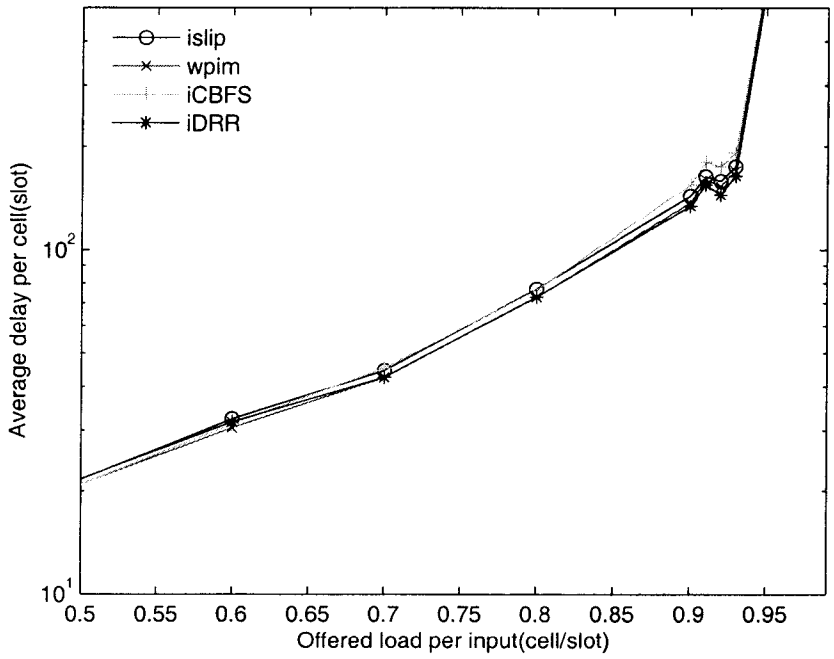


Figure 3.7: Average Delay under 2-state Markov-modulated arrivals with average burst size of 16.

As shown in Figure 3.7, the average delay for iCBFS is almost identical to iDRR, WiSLIP and WPIM.

3.5 A Port-based Fair Scheduling Algorithm

There is a trade-off in the design of high-speed switches between fairness among flows and simplicity of hardware design required for high-speed implementation. On the one hand, flow-based-scheduling guarantees fairness among flows by isolating non-conforming flows and provides bandwidth guarantees to individual flows; on the other hand, it makes hardware design relatively complex, and does not scale well as the number of flows grows. Port-based scheduling [SV95] allows simple hardware implementation at the cost of a coarse granularity of bandwidth guarantee. Rather than tracking *individual* flows at each input port, a port-based scheduler tracks the *aggregate* bandwidth reservation at each input port. Consequently, the complexity of a port-based scheduler is proportional to the switch size instead of the number of flows, which can be significantly larger. Thus, port-based scheduling can reduce the complexity of the scheduler considerably.

We propose to divide scheduling into two layers: CBFQ per virtual queue at the input side (labelled $V_{CBFQ}(i,j)$ for packets at input port i destined to output j), and a port-based scheduler, $PCBFQ_j$, at each output port j. $V_{CBFQ}(i,j)$ can be implemented in software using dynamic RAM (DRAM), and $PCBFQ_j$ can be easily implemented in hardware. Intuitively, using this hierarchical scheduling scheme, the complexity of the original $CBFQ_j$ engine at each output j is distributed among all the input ports and the $PCBFQ_j$ only deals at the abstraction of port-based scheduling; thus simplifying the design considerably.

The $PCBFQ_j$, at each output port j, maintains a counter K_{ij} and a bandwidth share S_{ij} for the aggregate bandwidth reservation from input i to output j. Similar to iCBFS, each input arbiter i uses a set of counters C_{ij} to track the aggregate quota for each $input_i$–$output_j$ pair. A VOQ_{ij} becomes candidate if $K_{ij} \geq 1$. The port-based of iCBFS, called iPCBFS would execute the request, grant, and accept stages as described in the previous section.

Note that the simulation results for iPCBFS are identical to the simulation results for iCBFS described Section 3.4.

3.6 Complexity of iCBFS

We assume a CRCW PRAM model, and estimate the complexity of iCBFS scheduling for an $N \times N$ switch. In the iCBFS algorithm, each time an output arbiter selects a flow to send a grant signal, all the operations executed are $\Theta(1)$ time. Similarly, each time an input arbiter selects an input to send the accept signal, all the operations executed are $\Theta(1)$ time.

The priority sort of the flows to select the flow with the largest share at each output arbiter changes only at the burst level timescale. That is, each flow's share is allocated upon the admission of a new flow and does not change during its lifetime. Consequently, the sorting consists only of extracting the pre-ordered list of active flows from a static list. Note that iDRR also maintains a pre-ordered list of active flows such that the flow with the smallest reservation is always used in calculating the quota for other flows. During the grant stage of iCBFS, the counter values do not need to be sorted according to their values. Consequently, we only need to compare each counter's value to 1. We point that all the counters' update and comparison operations can be implemented using integers.

In iCBFS, each output arbiter needs to maintain a counter for each flow, whereas in iPCBFS the number of counters is fixed and equals N.

Similar to all algorithms based on ΠRGA paradigm, both iCBFS, and iPCBFS may require up to N iterations in the worst case and an average of $\Theta(\log N)$ iterations for uniform traffic.

3.7 Conclusion

We proposed iCBFS, a flow-based fair scheduling algorithm for Internet routers with IQ switches. We showed through simulation that iCBFS can fairly allocate bandwidth in proportion to flows' reservations and provide considerably better fairness over short-time scales compared to all other schemes; thereby, iCBFS reduces the

jitter and delay for multimedia services like VOIP and video-on-demand. In addition, the algorithm achieves 100% throughput for uniform traffic. To simplify the implementation complexity of iCBFS, we proposed a port-based version of iCBFS, iPCBFS, which is simpler to implement in hardware.

Chapter 4

On Tracking the Behaviour of an Output-Queued Switch

We address the problem of fair scheduling of packets in Internet routers with IQ switches and unity speedup. Scheduling in IQ switches is formulated as *tracking* the behaviour of an OQ switch that provides optimal performance. We present the notion of "lag" as a performance metric that measures the difference between a packet's departure time in an IQ switch over that provided by an OQ switch. We prove that per packet mean lag is bounded for a maximum weight matching scheduling policy that uses lag values for its weights and derive a bound on the mean lag value using a Lyapunov function technique. Furthermore, we propose a simple heuristic tracking scheduling policy and evaluate its performance by simulation.

4.1 Introduction

The Internet's success depends on the deployment of high-speed switches that provide QoS guarantees for multimedia services, and high switching capacity that makes use of the virtually unlimited bandwidth of optical fibers.

On the one hand, the demand of QoS guarantees can be met using OQ switches, which provides optimal throughput. In addition, much research effort has been devoted to packet scheduling at output ports to support fair bandwidth sharing that provides delay bounds for regulated traffic(e.g., weighted fair queueing (WFQ) [PG93]). However, output queueing for an $N \times N$ switch requires the switching fabric and

memory to run up to N times faster than the line rate; unfortunately, for large N or for high-speed data lines, memories with sufficient bandwidth are not available. On the other hand, the fabric and the memory of an IQ switch need only to run as fast as the line rate. This property makes input queueing very appealing for switches with fast line rates or with a large number of ports. One method that has been proposed to reduce HOL blocking is to increase the speedup of a switch (See Section 1.3.1.).

A theoretical result [CGMP99] established that an $N \times N$ combined input-and output-queued (CIOQ) switch with a speedup of two could exactly *emulate* an $N \times N$ OQ switch for any traffic pattern of input cells. Emulation occurs at every time instance if, under identical inputs both systems produce identical departures. Unfortunately, the complexity of the scheduling algorithm presented in [CGMP99] renders OQ switch emulation infeasible (see [KPCS99], [MRS03] for a discussion of the complexity). The speedup requirement translates to a smaller time available for the execution of the arbitration algorithm. In a hardware implementation, reduction of the available time by a factor of two poses a substantial problem, although the difference does not seem significant asymptotically; it translates to a requirement of doubling the operating frequency of the arbiter, which might not be practically achievable. The tradeoff between the delay and speedup in a CIOQ switch has been analyzed in [GLPS04]. Furthermore, Minkenberg [Min02] has shown that exact emulation of an OQ using a CIOQ switch is possible only if the CIOQ switch has infinite output buffers.

Most commercial high-performance switches and routers (e.g., CISCO 1200 [Cis04], BBN [PCB+98], Lucent Cajun [Luc04] family, or Avici TSR45000 [Avi04]) use IQ switches. Most of these high-speed switches are built around a crossbar switch that is configured using a centralized scheduler designed to provide high throughput.

We consider scheduling policies in an IQ-crossbar switch with a unity speedup. Given that an IQ switch requires at least a speedup of two to exactly emulate an OQ switch [CGMP99], an IQ scheduling policy with a unity speedup can not exactly emulate the behaviour of an OQ switch, under all possible traffic patterns. Consequently, we formulate scheduling in an IQ switch as the problem of tracking an OQ switch. We propose the "lag" as a performance metric that measures the difference

between a packet's departure time in an IQ switch over that provided by an OQ switch. We present an IQ scheduling policy with unity speedup for which the lag is bounded and derive a bound on the mean lag value per packet. Furthermore, we propose a simple heuristic tracking scheduling policy and evaluate its performance by simulation. Although in this chapter we describe the case of tracking an OQ switch implementing only a FIFO scheduling policy, our results can be easily extended for other nonanticipative (decisions do not depend on future arrivals) scheduling policies.

This chapter is organized as follows. Section 4.2 formulates scheduling in an IQ switch with unity speedup as tracking the behaviour of an OQ switch. Section 4.3 provides motivation for tracking the behaviour of an OQ switch and discusses related work. In Section 4.5, we present two scheduling policies for tracking the behaviour of an OQ switch. First, we present a scheduling policy called *maximum weighted lag* (MWL). We prove that the mean lag value is bounded for MWL and derive an upper bound on its value using a Lyapunov function technique. The MWL scheduling policy has a high implementation cost, but serves as a solid base for developing other practical scheduling policies that approximate its performance. Consequently, we present a simpler heuristic tracking policy that can be readily implemented in hardware. The performance of the proposed scheduling policies is evaluated by simulation in Section 4.6. Section 4.7 provides our conclusions.

4.2 Problem Formulation

We consider an $N \times N$ OQ switch that uses scheduling policy Π_{OQ} and an IQ switch with unity speedup that uses scheduling policy Π_{IQ}. For an $N \times N$ switch, we use the following notational conventions: i an input, $1 \leq i \leq N$; j an output, $1 \leq j \leq N$; $Q_{i,j}$ is the VOQ at input i and buffers cells destined for output j; $HOL_{i,j}$ is the head-of-line cell at $Q_{i,j}$.

Let the average cell arrival rate at input i for output j be λ_{ij} . We assume that incoming traffic is admissible; that is, $\sum_{i=1}^{N} \lambda_{ij} < 1$, and $\sum_{j=1}^{N} \lambda_{ij} < 1$. The arrival process is identical to both switches. The goal is to find a scheduling policy Π_{IQ} that tracks the behaviour of the OQ switch as close as possible, where we define what

tracking means more precisely after introducing some definitions. Given that an IQ switch requires at least a speedup of two to exactly emulate an OQ switch [CGMP99], a scheduling policy for an IQ switch with a unity speedup can not exactly emulate the behaviour of an OQ switch, under all possible traffic patterns. In general, cells arriving to the IQ switch implementing Π_{IQ} will depart at some later time than the OQ switch implementing Π_{OQ}. Consequently, we say that an IQ switch implementing Π_{IQ} lags the behaviour of the OQ switch implementing Π_{OQ} .

4.2.1 Definition of Terms

Here we make precise some of the terminology used throughout this chapter.

Definition 1. *Arrival Rate Matrix (λ): $\lambda \equiv [\lambda_{ij}]$, where the arrival process is assumed to be admissible and stationary; that is, $\sum_{i=1}^{N} \lambda_{ij} < 1$, $\sum_{j=1}^{N} \lambda_{ij} < 1$, $\lambda_{ij} \geq 0$ and associated arrival rate vector $\underline{\lambda} \equiv (\lambda_{1,1}, \ldots, \lambda_{1,N}, \ldots, \lambda_{N,1}, \ldots, \lambda_{N,N})^T$.*

Definition 2. *Ideal departure time (IDT): The ideal departure time for a cell c, $IDT(c)$, is the time slot at which c will depart from an OQ switch using Π_{OQ}.*

Definition 3. *Actual departure time (ADT): The actual departure time (ADT) for a cell c, $ADT(c)$, is the time slot at which c departs from the switch under consideration (i.e., IQ implementing Π_{IQ}).*

Definition 4. *Cell Lag (CL): The cell lag for a cell c, $CL(c)$, is the difference between the ideal departure time and the actual departure time. Precisely,*

$$CL(c) \equiv \begin{cases} ADT(c) - IDT(c) & ADT(c) > IDT(c) \\ 0 & otherwise \end{cases} \tag{4.1}$$

In addition, we define the cell lag for a cell c given the current time slot n, $CL(c,n)$, as the difference between the ideal departure time and the current time slot. Precisely,

$$CL(c,n) \equiv \begin{cases} n - IDT(c) & n > IDT(c) \\ 0 & otherwise \end{cases}$$

The goal of a scheduling policy can be characterized by any statistical metric that attempts to minimize the cell lag; for example, in Section 4.5.1 we present a scheduling policy that minimizes the mean lag value per packet.

Note that according to equation (4.1) the lag is nonnegative and generally a cell's ADT is greater than its IDT, however, a cell may occasionally depart from an IQ

switch earlier than an OQ switch; for example, consider a 2×2 switch at a specific time slot such that the two most lagging cells for its outputs (e.g., outputs 1 and 2) reside at the same input port (e.g., input 1). Because the scheduling policy can transfer at most one cell from each input port (e.g., input 1), another cell with an IDT in the future can be selected from the other input port (e.g., input 2) to improve the throughput.

4.3 Motivation and Related Work

In an OQ switch arriving packets are immediately available at the outgoing link. Consequently, the only shared resource in an OQ switch is the outgoing link for which packets contend for access (output contention). In an IQ with switch there are essentially two shared resources: the switch fabric and the outgoing link. Arriving packets are queued at the input port of the switch and they must first contend for access to the switch fabric, before contending for the outgoing link (see Section 2.2).

In an IQ switch packets are queued at the input port of the switch and they must first contend for access to the switch fabric (input contention), before contending for the outgoing link; that is, in an IQ switch, there are two shared resources: the switch fabric and the outgoing link.

All present IQ scheduling policies resolve input and output contention using heuristics such as using a round-robin scheme at both the input and output to solve the contention fairly [McK99], or using the packet's age (i.e., time in the switch) to resolve contention [MMAW99]. All these schemes can be seen as an approximation to the ideal case of an OQ switch, where all of the outgoing links are independent and packets are served independently in each outgoing link; that is, by tracking the behaviour of an OQ switch and minimizing the lag, we automatically resolve input and output contention in a fair manner and eliminate any starvation problem of inputs that other scheduling policies have to carefully handle.

We emphasize that significant research effort (e.g., [PG93], [Cru91], [PG94]) has been done in developing scheduling policies for ideal servers that provide bounded latency, jitter, and end-to-end delay for traffic flows. Unfortunately, the Internet

does not consist only of ideal servers, but rather of heterogeneous servers (i.e., non ideal IQ and CIOQ servers, and ideal OQ servers). By tracking the behaviour of an ideal server, we approximate its behaviour as close as possible and attempt to bound the performance difference between the ideal server and an IQ switch.

Tabatabaee et al. [TGT01] consider the related problem of packetizing arbitrary fluid policies in an $N \times N$ crossbar switch using FIFO virtual output queues. They define trackable fluid policies such that for each pair of input and output ports, at each time step, the cumulative number of packets sent between these ports differs from the cumulative fluid scheduled between these ports by less than 1. They prove that a tracking policy always exists for the special case of a 2×2 switch, provide an example for a 3×3 switch where a nonanticipative tracking policy does not exist, and propose several heuristics for packetizing fluid policies on general $N \times N$ switches. Rosenblum et al. [RGT04] further extend the results in [TGT01] by relaxing the tracking constraint such that the cumulative difference in the number of packets sent using the fluid and packetized policies can be more than one packet. Our work differs from [TGT01] and [RGT04] in that we track the precise packet departure sequence in an OQ switch rather than the aggregate rate provided by a fluid scheduling policy in an IQ switch, which does not necessarily track an OQ switch; for two scheduling policies to provide the same service rate they need to serve only the same number of packets per link, rather than tracking the precise packet departure order, which can be different between the two scheduling policies. This issue is discussed in more detail in Section 4.5.1.

4.4 Computing the Ideal Departure Time

We consider the case of $\Pi_{OQ} = FIFO$. The architecture of our IQ switch is shown in Figure 4.1. We use virtual output queueing (VOQ) at each input port of the switch and a crossbar as the switching fabric.

For $\Pi_{OQ} = FIFO$, arriving cells at the IQ switch can be immediately assigned an IDT using a simple parallel prefix circuit [Szy97](i.e., a ranker circuit). Let $N_j(n)$ be the number of cells in the OQ switch destined to output j at time slot n. The IQ

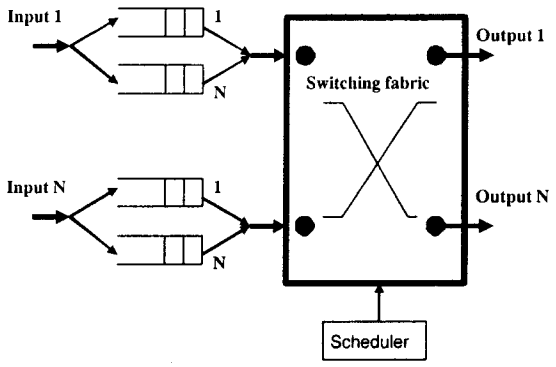


Figure 4.1: Logical structure of an input-queued switch.

switch uses N rankers such that each ranker calculates the number of cells present in the OQ switch being tracked. At the beginning of each time slot, n, the number of packets in the OQ switch is computed as follows:

$$N_j(n) \equiv \begin{cases} N_j(n-1) - 1 & N_j(n-1) > 0 \\ 0 & N_j(n-1) = 0 \end{cases}$$

Note that the subtraction of one in the previous equation accounts for one (cell/time slot) departure in the OQ switch. For every new cell c arriving at time slot n destined to output j, ranker j assigns a numeric rank (from $1 \ldots N$) in a linear order [1] to packets arriving for output port j. The IDT of each cell is equal to its numeric rank plus $N_j(n-1)$, and $N_j(n-1)$ is updated accordingly. The complexity of computing the $IDT(c)$ in hardware using a parallel prefix computation is $\Theta(\log N)$ depth and $\Theta(N)$ circuit size, expressed in terms of binary operators [Szy97].

4.5 Tracking Scheduling Policies

In this section we present two tracking scheduling policies. First, we present the maximum weighted lag scheduling policy and prove that its per packet mean lag is bounded and derive a bound on the mean lag value using a Lyapunov function

[1] We investigated diverse ordering schemes (e.g., round-robin, linear, etc) for assigning IDT to simultaneous cell arrivals destined to the same output and found it to have an insignificant effect on the results, when the queues had infinite capacity.

technique. Second, we propose a heuristic tracking scheduling policy called iLag based on maximal matching that is simple to implement in hardware.

4.5.1 Maximum Weighted Lag Scheduling Policy

Maximum weighted lag (MWL) is based on the implementation of a maximum bipartite weight-matching algorithm (See Section 2.3.2). At every time slot n, we associate a weight $W_{i,j}$ to every $Q_{i,j}$ such that $W_{i,j} = CL(HOL_{i,j}, n)$; that is, $W_{i,j}$ is the lag of an HOL packet in $Q_{i,j}$. The maximum weighted lag scheduling policy finds a matching $\mathcal{M}$ that maximizes $\sum_{(i,j)\in M} W_{i,j}$ and can be found by solving an equivalent network flow problem [AMO93]. The sequential run time complexity of MWM is $\Theta(N^3 \log N)$[AMO93].

Previous work on MWM considered only the weight to be either some function of the occupancy of the VOQs (i.e., number of packets in each VOQ) or the waiting time of the cell at the head of line of each VOQ (e.g., [MMAW99], [KM01], [DP00], [LMNM01a], and [LMNM03]). Consequently, these algorithms do not necessarily track the behaviour of an OQ switch and a cell's departure times may deviate from the ideal case under non-uniform traffic. In addition, using the occupancy of the VOQs as the edge weight can lead to starvation of certain inputs [MMAW99].

Because MWL computes the matching with the maximum possible total weight during every time slot, it aims at minimizing the mean lag (μ_{lag}). Although this algorithm is too complex to implement in practice, it serves as a reference model for which other approximation algorithms are developed.

The stability of maximum weighted matching scheduling policies is a well studied problem in the literature. McKeown et al. [MMAW99] proved the stability of longest queue first (LQF) and oldest cell first (OCF) maximum weight matching for all admissible i.i.d. arrival processes using a Lyapunov function technique; Dai and Prabhakar [DP00] extended the results to prove the stability of a maximum weight matching algorithm under any admissible arrival processes using fluid model techniques.

Although the results for the fluid model technique established in [DP00] could easily be used to prove the stability of MWL, it can not be further extended to derive a bound on the expected lag value. Consequently, we use a Lyapunov function

technique that allows us to derive a bound on the expected lag value as described next.

Theorem 1. *A FIFO tracking policy that uses the maximum weighted lag as the scheduling policy is stable (achieves* 100% *throughput) for all admissible i.i.d. arrival processes.*

The proof of Theorem 1 for the stability of MWL is an adaptation of the proof for stability of Oldest Cell First scheduling presented in [MMAW99]. The proof uses substantially the same techniques to first develop a discrete time Markov chain reflecting the lag of a cell. The proof then identifies a quadratic Lyapunov function which establishes the existence of a negative drift in the Markov chain for sufficiently large states. The existence of the negative drift implies the stability of the Markov chain, using a result of Kumar and Meyn [KM95]. The stability of the lag implies the stability of the queue occupancy. The main differences in the proofs are as follows. The definition of a cell's weight is changed from the cell's age to the cell's lag, which is equal to the cell age minus a positive term reflecting the cell's ideal departure time. Lemma's 7, 8, 9 and 10 in [MMAW99] are modified to reflect the new cell weights. We present a model of the system and use it to establish the proof of Theorem 1 next.

Model

The arrival process at each input port i is assumed to be a i.i.d. discrete-time stationary ergodic process of fixed size cells. At the beginning of each slot, either zero or one cell arrives at each input port. Virtual output queueing is used such that when a cell arrives at time slot n for output j at input i , it is placed in queue $Q_{i,j}$.

Definition 5. *Let* $\underline{Q}(n)$ *be the occupancy vector at time slot n such that*

$$\underline{Q}(n) \equiv (Q_{1,1}(n), \ldots, Q_{1,N}(n), \ldots, Q_{N,1}(n), \ldots, Q_{N,N}(n))^T.$$

Definition 6. *Let* $\lambda_{min} \equiv min(\lambda_{i,j}, 1 \leq i, j \leq N)$.

Definition 7. *Let* $C_{i,j}(n)$ *denote the HOL cell of* $Q_{i,j}$ *at time slot n.*

Definition 8. *Let* $\underline{\tau}(n)$ *be the interarrival time vector such that*

$$\underline{\tau}(n) \equiv (\tau_{1,1}(n), \ldots, \tau_{1,N}(n), \ldots, \tau_{N,1}(n), \ldots, \tau_{N,N}(n))^T.$$

where $\tau_{i,j}(n)$ *is the interarrival time between* $C_{i,j}(n)$ *and the cell behind it in* $Q_{i,j}$ *([MMAW99], appendix B, definition 2).*

Definition 9. *Let $A(n)$ be the arrival matrix representing the arrivals into each queue at time slot n, $A(n) \equiv [A_{i,j}(n)]$ where*

$$A_{i,j}(n) \equiv \begin{cases} 1 & \text{if an arrival occurs at } Q_{i,j} \text{ at time slot } n \\ 0 & \text{otherwise} \end{cases}$$

and the associated arrival vector is

$$\underline{A}(n) \equiv (A_{1,1}(n), \ldots, A_{1,N}(n), \ldots, A_{N,1}(n), \ldots, A_{N,N}(n))^T.$$

([MMAW99], appendix A, definition 2)

Definition 10. *Let $S(n)$ be the service matrix indicating which queues are served during slot n, $S(n) \equiv [S_{i,j}(n)]$ where*

$$S_{i,j}(n) \equiv \begin{cases} 1 & \text{if } Q_{i,j} \text{ is served at time slot } n \\ 0 & \text{otherwise} \end{cases}$$

and $S(n) \in S$, the set of service matrices[2]. Note that $S(n)$ is a permutation matrix; that is, $\sum_{i=1}^{N} S_{ij} = \sum_{j=1}^{N} S_{ij} = 1$. We define the associated service vector $\underline{S}(n) \equiv (S_{1,1}(n), \ldots, S_{1,N}(n), \ldots, S_{N,N}(n))^T$.

Definition 11. *Let $\underline{L}(n)$ be the lag vector at time slot n such that*

$$\underline{L}(n) \equiv (L_{1,1}(n), \ldots, L_{1,N}(n), \ldots, L_{N,1}(n), \ldots, L_{N,N}(n))^T,$$

where $L_{i,j}(n)$ is the lag of $C_{i,j}(n)$ (cell at HOL of $Q_{i,j}$ at time slot n). (Recall that the lag is the difference between the ideal departure time and the current time, also note that all elements in the lag vector are nonnegative.)

Definition 12. *Let $L_{max} \equiv max(L_{i,j}, 1 \leq i, j \leq N)$.*

Definition 13. *Let T be a positive-semidefinite diagonal matrix whose diagonal elements are $\lambda_{1,1}, \ldots, \lambda_{1,N}, \ldots, \lambda_{N,1}, \ldots, \lambda_{N,N}$.*

Definition 14. *$[\underline{a} \odot \underline{b} \odot \underline{c}]$ denotes a vector in which each element is a product of the corresponding elements of the vectors: $\underline{a}$, $\underline{b}$, and $\underline{c}$, i.e., $a_{i,j}b_{i,j}c_{i,j}$.*

Definition 15. *Let $\underline{1}$ denote a column vector of dimension N^2 whose elements are all ones.*

[2]This definition of the "service" matrix is a permutation matrix, which includes the case where an empty queue is served.

Definition 16. *Let* $\underline{D}(n, n+\triangle n)$ *be the aggregate arrival vector for each output port during the time interval* $[n, n+\triangle n]$

$$\underline{D}(n, n+\triangle n) = (D_1(n, n+\triangle n), \ldots, D_N(n, n+\triangle n))^T,$$

where $D_j(n, n+\triangle n)$ *represents the aggregate number of cells that arrived to the switch during the time interval* $[n, n+\triangle n]$ *destined to output* j*. Note that the dimension of the vector* $\underline{D}(n, n+\triangle n)$ *is* N*, whereas most previously defined vectors have dimension* N^2*, consequently, we define the following vector:*

$$\begin{aligned}\underline{Z}(n, n+\triangle n) \equiv \Big(&D_1(n, n+\triangle n), \ldots, D_N(n, n+\triangle n), \ldots, \\ &D_1(n, n+\triangle n), \ldots, D_N(n, n+\triangle n), \ldots, \\ &D_1(n, n+\triangle n), \ldots, D_N(n, n+\triangle n)\Big)^T\end{aligned}$$

i.e., the vector $\underline{Z}(n, n+\triangle n)$ *is the vector* $\underline{D}(n, n+\triangle n)$ *written out* N *times.*

Definition 17. *The* approximate *Lag next-state vector, which does not consider the case of an empty queue is given by:* $\underline{\widetilde{L}}(n+1) \equiv \underline{L}(n) + \underline{1} - \left[\underline{S}(n) \odot [\underline{\tau}(n) + \underline{Z}(\underline{\tau}(n))]\right]$

Explanation: The above equation describes the evolution of the lag vector. In the above equation, if $Q_{i,j}$ is not serviced at slot n then its corresponding $S_{i,j}$ element in $\underline{S}(n)$ is zero and the corresponding term in $\underline{S}(n) \odot [\underline{\tau}(n) + \underline{Z}(\underline{\tau}(n))]$ cancels out. In this case the lag increases by 1. Alternatively, if the HOL cell at $Q_{i,j}$ is serviced at time slot n, then we need to calculate the lag of the cell following it in the queue. We consider two subcases:

CASE A: There were no packet arrivals to the switch destined to output j during the interarrival period between the HOL cell at $Q_{i,j}$ and the cell following it (i.e., $Z_{i,j}(\tau_{ij})$ is zero). In this case, the corresponding element for $Q_{i,j}$ in $\underline{S}(n)$ is 1 and $Z_{i,j}(\tau_{i,j})$ is zero. Therefore,

$$L_{i,j}(n+1) = L_{i,j}(n) + 1 - \tau_{i,j}(n),$$

i.e., the new lag is the old lag minus the interarrival time between the two cells.

CASE B: There were arrivals during the interarrival period between the HOL cell in $Q_{i,j}$ and the cell following it. In this case, all cells that arrived during this interarrival period should depart from the switch (or be selected to be transferred

across the switch by the scheduler) before the new HOL cell at $Q_{i,j}$, so the new lag is given by:

$$L_{i,j}(n+1) \equiv L_{i,j}(n) + 1 - \tau_{i,j}(n) - Z_{i,j}(\tau_{i,j}(n)).$$

The following facts are used in the proof of the stability of the lag vector.

Fact 1. *For all i, j, n an interarrival time $\tau_{i,j}(n)$ is independent of the lag $L_{i,j}(n)$. This fact is true because we are assuming an i.i.d. traffic model.* .

Fact 2. $\tau_{i,j}(n) \geq 1$ *because there is at most one arrival per time slot, so the arrival times of any two consecutive cells must be at least one slot apart.*

Fact 3. *For all i, j, n $(\lambda_{i,j} = 0) \Rightarrow (\|Q_{i,j}\| = 0) \Rightarrow (L_{i,j}(n) = 0)$; that is, any queue whose arrival rate is zero is empty and consequently has a zero lag.*

Proof of Theorem 1. We prove the stability of the lag vector, which implies the stability of the queue occupancy. Recall that the lag is defined in terms of the total occupancy of packets in the switch destined to an output port.

The following Lemma is adapted from [MMAW99], Lemma 7.

Lemma 1. *$\underline{L}^T(n)\underline{\lambda} - \underline{L}^T(n)\underline{S}^*(n) \leq 0$, $\forall \underline{L}(n), \underline{\lambda}$ where $\underline{S}^*(n)$ is such that $\underline{L}^T(n)\underline{S}^*(n) = max(\underline{L}^T(n)\underline{S}(n))$ (Note that $\underline{S}^*(n)$ is the service vector selected by the maximum weighted lag scheduling policy at time slot n.)*

Proof. Identical to the proof of [MMAW99], Lemma 2. □

The following Lemma is adapted from [MMAW99], Lemma 8 and is simplified for an $N \times N$ switch rather than an $N \times M$ switch.

Lemma 2. *For all $\underline{\lambda} \leq (1-\beta)\underline{\lambda}_m$ (the inequality is interpreted componentwise), $0 < \beta < 1$, where $\underline{\lambda}_m$ is any rate vector such that $\|\underline{\lambda}_m\|^2 = N$, there exists $0 < \varepsilon < 1$ such that*

$$E\big[\widetilde{\underline{L}}^T(n+1)T\widetilde{\underline{L}}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)\big] \leq \varepsilon\|\underline{L}(n)\| + K.$$

Proof. By expansion

$$\begin{aligned}\widetilde{\underline{L}}^T(n+1)T\widetilde{\underline{L}}(n+1) &= \underline{L}^T(n)T\underline{L}(n) + 2\underline{L}^T(n)\underline{\lambda} \\ &\quad - 2\underline{L}^T(n)\Big[\underline{S}^*(n)\odot\underline{\tau}(n)\odot\underline{\lambda}\Big] - 2\underline{L}^T(n)\Big[\underline{S}^*(n)\odot\underline{Z}(\underline{\tau}(n))\odot\underline{\lambda}\Big] \\ &\quad + \sum_{i,j}\lambda_{i,j} - 2\sum_{i,j}S^*_{i,j}(n)\tau_{i,j}(n)\lambda_{i,j} \\ &\quad - 2\sum_{i,j}S^*_{i,j}(n)Z_{i,j}(\tau_{i,j}(n))\lambda_{i,j} + \sum_{i,j}S^*_{i,j}(n)\tau^2_{i,j}(n)\lambda_{i,j} \\ &\quad + 2\sum_{i,j}S^*_{i,j}(n)Z_{i,j}(\tau_{i,j}(n)) + \sum_{i,j}S^*_{i,j}(n)Z^2_{i,j}(\tau_{i,j}(n))\lambda_{i,j}.\end{aligned}$$

Subtracting $\underline{L}^T(n)T\underline{L}(n)$ from both sides and taking the expected value and observing that the expected value of τ is $\frac{1}{\lambda}$,

$$\begin{aligned}&E[\widetilde{\underline{L}}^T(n+1)T\widetilde{\underline{L}}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)] \\ &= 2\underline{L}^T(n)\underline{\lambda} - 2\underline{L}^T(n)\underline{S}^*(n) - 2\underline{L}^T(n)\Big(\underline{S}^*(n)\odot\underline{Z}(\underline{\tau})\odot\underline{\lambda}\Big) \\ &\quad + \sum_{i,j}\lambda_{i,j} - 2\sum_{i,j}S^*_{i,j}(n) - 2\sum_{i,j}S^*_{i,j}(n)E[Z_{i,j}(\tau_{i,j})\lambda_{i,j}] \\ &\quad + \sum_{i,j}\frac{S^*_{i,j}(n)}{\lambda_{i,j}} + 2\sum_{i,j}S^*_{i,j}(n)E[Z_{i,j}(\tau_{i,j})] \\ &\quad + \sum_{i,j}S^*_{i,j}(n)E[Z^2_{i,j}(\tau_{i,j})\lambda_{i,j}].\end{aligned} \tag{4.2}$$

We make use of the following properties to simplify equation (4.2) and establish Lemma 2:
(a) $\sum_{i,j}\lambda_{i,j} < N$; (from the admissibility constraints)
(b) $\sum_{i,j}S^*_{i,j}(n) \geq 0$; (from the scheduling algorithm properties) so, this term can be ignored in equation (4.2) because it has a negative sign.
(c) $\underline{L}^T(n)\Big(\underline{S}^*(n)\odot\underline{Z}(\underline{\tau})\odot\underline{\lambda}\Big) \geq 0$; (because each element in this term is non-negative; observe that this term has a negative sign in equation (4.2) so it can be ignored)
(d) $\sum_{i,j}S^*_{i,j}(n)E[Z_{i,j}(\tau_{i,j})\lambda_{i,j}] \geq 0$; (because each element in this term is non-negative; observe that this term has a negative sign in equation (4.2) so it can be ignored) Also, note that the following positive terms in equation (4.2) are bounded:

$$\begin{aligned}
&\sum_{i,j} S^*_{i,j}(n)/\lambda_{i,j} \leq \psi < \infty, \\
&\sum_{i,j} S^*_{i,j}(n) E[Z^2_{i,j}(\tau_{i,j})\lambda_{i,j}] \leq \alpha < \infty \\
&\sum_{i,j} S^*_{i,j}(n) E[Z_{i,j}(\tau_{i,j})] \leq \gamma < \infty
\end{aligned} \tag{4.3}$$

From equation (4.2), properties (a) through (d), and equation (4.3), we obtain

$$\begin{aligned}
&E[\underline{\tilde{L}}^T(n+1)T\underline{\tilde{L}}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)] \\
&\leq 2\underline{L}^T(n)\underline{\lambda} - 2\underline{L}^T(n)\underline{S}^*(n) + N + \psi + 2\alpha + \gamma.
\end{aligned} \tag{4.4}$$

Using Lemma 2, we obtain:

$$\begin{aligned}
&\underline{L}^T(n)T\underline{L}(n) \leq -\beta \underline{L}^T(n)\underline{\lambda}_m \\
&\underline{L}^T(n)\underline{\lambda} - \underline{L}^T(n)\underline{S}^*(n) \leq -\beta \|\underline{L}^T(n)\| \|\underline{\lambda}_m\| \cos(\theta)
\end{aligned} \tag{4.5}$$

where θ is the angle between $\underline{L}^T(n)$ and $\underline{\lambda}_m$.

We now show that $\cos(\theta) > \delta$ for some $\delta > 0$ whenever $\underline{L}^T(n) \neq 0$ using the same approach as in [MMAW99], equations (16)-(18). This is included here for completeness and is simplified for an $N \times N$ switch rather than an $N \times M$ switch.

We do this by contradiction: suppose that $\cos(\theta) = 0$, i.e., $\underline{L}^T(n)$ and $\underline{\lambda}_m$ are orthogonal. This can only occur if $\underline{L}^T(n) = 0$, or if for some i, j, both $\lambda_{i,j} = 0$ and $L_{i,j}(n) > 0$, which is not possible: for $Q_{i,j}$ to have a lag greater than zero, $\lambda_{i,j}$ must be greater than zero. Therefore, $\cos(\theta) > 0$ unless $\underline{L}^T(n) = 0$. Now we show that $\cos(\theta) > \delta$ for some $\delta > 0$. Because $\lambda_{i,j} > 0$ wherever $L_{i,j}(n) > 0$, and because $\|\underline{\lambda}\|^2 < N$

$$\cos(\theta) = \frac{\underline{L}^T(n)\underline{\lambda}}{\|\underline{L}(n)\|\|\underline{\lambda}\|} \geq \frac{L_{max}(n)\lambda_{min}}{\|\underline{L}(n)\|\sqrt{N}}.$$

Also, $\|\underline{L}(n)\| \leq N L_{max}(n)$, and so $\cos(\theta)$ is bounded below by

$$\cos(\theta) \geq \frac{\lambda_{min}}{N^{\frac{3}{2}}}. \tag{4.6}$$

Substituting equation (4.6) in equation (4.5) we get

$$\cos(\theta) = \frac{\underline{L}^T(n)\lambda}{\|\underline{L}(n)\|\|\lambda\|} \geq \frac{L_{max}(n)\lambda_{min}}{\|\underline{L}(n)\|\sqrt{N}}$$

$$E[\widetilde{\underline{L}}^T(n+1)T\widetilde{\underline{L}}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)] \leq -2\varepsilon\|\underline{L}(n)\| + K \tag{4.7}$$

where $\varepsilon = 2\beta\frac{\lambda_{min}}{N}$ and $K = \psi + N + 2\alpha + \gamma$. □

The following Lemma is adapted from [MMAW99] Lemma 9 and is simplified for an $N \times N$ switch rather than an $N \times M$ switch.

Lemma 3. *For all $\underline{\lambda} \leq (1-\beta)\underline{\lambda}_m$ (the equation is interpreted componentwise), $0 < \beta < 1$, where $\underline{\lambda}_m$ is any rate vector such that $\|\underline{\lambda}_m\| = N$, there exists $0 < \varepsilon < 1$ such that*

$$E\left[\underline{L}^T(n+1)T\underline{L}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)\right] \leq \varepsilon\|\underline{L}(n)\| + K.$$

Observe that the difference between Lemmas 2 and 3 is that Lemma 2 uses the approximate next state vector, whereas Lemma 3 uses the exact next state vector. The approximate next state vector assumes that each VOQ always has a packet. The exact next state vector takes the empty queue case into account.

The proof of this Lemma is similar to the proof of Lemma 9 in [1], and is included here for completeness.

Proof.

$$L_{i,j}(n+1) = \begin{cases} \widetilde{L}_{i,j}(n+1) & \widetilde{L}_{i,j}(n+1) \geq 0 \\ 0 & \widetilde{L}_{i,j}(n+1) < 0 \end{cases} \tag{4.8}$$

The fact that T is a positive-semidefinite matrix together with equation (4.8) imply that for all n

$$\underline{L}^T(n+1)T\underline{L}(n+1) \leq \widetilde{\underline{L}}^T(n+1)T\widetilde{\underline{L}}(n+1).$$

Therefore,

$$E\left[\underline{L}^T(n+1)T\underline{L}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)\right] \leq E\left[\widetilde{\underline{L}}^T(n+1)T\widetilde{\underline{L}}(n+1)|\underline{L}(n)\right].$$

This proves the Lemma. □

Lemma 4. *There exists a quadratic Lyapunov function $V(\underline{L}(n))$ such that*

$$E\left[V(\underline{L}(n+1)) - V(\underline{L}(n))|\underline{L}(n)\right] \leq -\varepsilon\|\underline{L}(n)\| + K$$

where $K, \varepsilon > 0$.

Proof. From Lemma 3, $V(\underline{L}(n)) = \underline{L}^T(n)T\underline{L}(n)$, $\varepsilon = 2\beta\frac{\lambda_{min}}{N}$, and $K = \psi + N + 2\alpha + \gamma$. □

Theorem 2. *Under Maximum Weighted Lag, the expectation of the lag values are bounded for all n under all admissible and independent arrival processes, i.e.,* $\forall n, E[\|\underline{L}(n)\|] < \infty$.

Proof. $V(\underline{L}(n)) = \underline{L}^T(n)T\underline{L}(n)$ is a quadratic Lyapunov function and according to the arguments in [KM95], it follows that the expectation of the lag values is bounded for all n under the maximum weighted lag scheduling policy. □

Theorem 3. *Under the MWL scheduling policy, the expectation of the queue occupancy is bounded for all n under all admissible and independent arrival process, i.e.,* $\forall n, E[\|\underline{Q}(n)\|] < \infty$.

Proof. That stability of the lag values implies the stability of the per packet additional waiting in the IQ switch using the MWL scheduling policy over that provided by the OQ switch being tracked. Given the traffic admissibility constraints, each packet's delay in the OQ switch being tracked is finite. Consequently, the total delay provided by the IQ switch using MWL is bounded. Therefore, all the queue occupancies in the IQ switch under MWL are bounded for all n. □

Different weight functions lead to different bounds on the average queue size (cell delay) with varying performance; for example, in [KM01] it is shown that all maximum weight matching scheduling policies with weight equal to the queue size raised to some positive α, $\|Q_{i,j}\|^\alpha$, are stable. However, it is shown through simulation that under a specific arrival pattern the average cell delay is smaller when $\alpha = 0.5$ than for all higher values of α. A methodology for deriving bounds on the cell delay and queue size is described in [LMNM03]. In [MMAW99] it was shown that Longest Queue First could potentially lead to starvation. Longest Port First (LPF) was proposed in [MM98] and was shown by simulation to provide better performance than LQF and OCF, but it is possible to construct a traffic pattern that leads to starvation for LPF [Mek98]. All the previous results are applicable to stability in a single node (switch). The problem of scheduling a network of IQ switches is considered in [AZ03] and it is shown that both the LQF and LPF scheduling policies can be unstable for a fixed traffic pattern in a simple network of eight IQ switches.

We establish a bound on the mean lag value using the techniques developed in [LMNM03]; the following definitions are needed for bound result in Theorem 4:

Definition 18. *L_1 Norm: Given a vector $\underline{Z} \in \mathbb{R}^{N^2}$, the norm $\|\underline{Z}\|_1$ is defined as:*

$$\|\underline{Z}\|_1 = \sum_{k=1}^{N^2} |z_k|.$$

Definition 19. *Input-Output Norm: Given a vector $\underline{Z} \in \mathbb{R}^{N^2}$, $\underline{Z} = \left\{z_k, k = Ni + j, i, j = 1, \ldots, N\right\}$, the norm $\|\underline{Z}\|_{IO}$ is defined as:*

$$\|\underline{Z}\|_{IO} = \max_{j=1,\ldots,N} \left\{ \sum_{k=1}^{N} |z_{Nk+j}|, \sum_{l=1}^{N} |z_{Nj+l}| \right\}$$

$\|\underline{Z}\|_{IO}$ takes the maximum of the sum of quantities related to all the queues referring either to the same input or to the same output; for example, the traffic arrival vector is admissible if and only if $\|\underline{\lambda}\|_{IO} < 1$.

Definition 20. *Let $\underline{L}(n)$ be the lag vector at time slot n such that*

$$\underline{L}(n) \equiv (L_{1,1}(n), \ldots, L_{1,N}(n), \ldots, L_{N,1}(n), \ldots, L_{N,N}(n))^T,$$

where $L_{i,j}(n)$ is the lag of $C_{i,j}(n)$ (cell at HOL of $Q_{i,j}$ at time slot n).

Theorem 4. *A bound on the mean lag, $E[\|\underline{L}(n)\|_1]$, using a maximum weighted Lag scheduling policy under any admissible i.i.d. arrival process is given by:*

$$E[\|\underline{L}(n)\|_1] \leq \frac{N^3 + 3N^2\|\underline{\lambda}\|_1}{2(1 - \|\underline{\lambda}\|_{IO})}.$$

Proof:

Definition 21. *Given a vector $\underline{Z} \in \mathbb{R}^{N^2}$, the second order norm $\|\underline{Z}\|_2$ is defined as:*

$$\|\underline{Z}\|_2 = \sqrt{\sum_{k=1}^{N^2} (Z_k)^2}$$

Definition 22. *The unit vector parallel to $\underline{Z}$ is denoted by $\hat{\underline{Z}}$, and is defined as:*

$$\hat{\underline{Z}} = \frac{\underline{Z}}{\|\underline{Z}\|_1}$$

To proceed we need the following theorem due to Leonardi et al. [LMNM03],Theorem 3.6, which is presented here in a form appropriate for the problem under consideration.

Theorem 5 ([LMNM03],Theorem 3.6). *Given a system of queues whose evolution is described by a Discrete Time Markov Chain (DTMC) with state vector* $Y_n \in \mathbb{N}^M$ *, whose state space* H *is a subset of the Cartesian product of a denumerable state space* H_L *and a finite state space* H_K*, and for which all the polynomial moments of lag distributions are finite, if a lower bounded polynomial function* $V(\underline{L}(n)),V : \mathbb{N}^N \to R$*, can be found, such that* $E\Big[V(\underline{L}(n)) \mid Y_n\Big] < \infty$ *and there exist two positive real numbers* $\epsilon \in \mathbb{R}^+$ *and* $B \in \mathbb{R}^+$*, such that*

$$E\Big[V(\underline{L}(n+1)) - V(\underline{L}(n)) \mid Y_n\Big] \leq -\epsilon f\Big(\|\underline{L}(n)\|\Big) \qquad \forall Y_n : \|\underline{L}(n)\| > B, \tag{4.9}$$

where $f(x)$ *is a continuous function in* $\mathbb{R}^+$*,then*

$$\begin{aligned}\lim_{n\to\infty} E\Big[f\Big(\|\underline{L}(n)\|\Big)\Big] \leq \lim_{n\to\infty} E\Big[f\Big(\|\underline{L}(n)\|\Big) \\ + \frac{V(\underline{L}(n+1)) - V(\underline{L}(n))}{\epsilon} | Y_n \in H_B\Big] \times P[Y_n \in H_B]\end{aligned} \tag{4.10}$$

Note that for MWL $Y(n) = (\underline{A}(n), \underline{L}(n), \underline{\tau}(n))$ is an appropriate DTMC and all the polynomial moments of the lag distribution are finite by Theorem 3.5 of [LMNM03], which is included here for completeness.

Theorem 6 ([LMNM03],Theorem 3.5). *Given a system of queues whose evolution is described by a DTMC with state vector* $Y_n \in \mathbb{N}^M$*, whose state space* H *is a subset of the Cartesian product of a denumerable state space* H_L *and a finite state space* H_K*, if a lower bounded function* $V(\underline{L}(n)$*, called Lyapunov function,* $V(\underline{L}(n)),V : \mathbb{N}^N \to R$*, can be found, such that* $E\Big[V(\underline{L}(n)) \mid Y_n\Big] < \infty$*,* $\forall Y_n$ *and there exists* $\epsilon \in \mathbb{R}^+$ *and* $B \in \mathbb{R}^+$ *such that*

$$E\Big[V(\underline{L}(n+1)) - V(\underline{L}(n)) \mid Y_n\Big] \leq -\epsilon\|\underline{L}(n)\| \qquad \forall Y_n : \|\underline{L}(n)\| > B,$$

then the system of queues is strongly stable. In addition, if there exists symmetric copositive matrix $Z \in \mathbb{R}^{N\times N}$ *defining the Lyapunov function* $V(\underline{L}(n)) = \underline{L}(n)Z\underline{L}^T(n)$*, then all the polynomial moments of the queue lengths distribution are finite.*

The proof of Theorem 4 consists of two steps. First, we find a lower bound on ϵ in equation (4.9). The second step is to use equation (4.10) to derive the bound on $E[\|\underline{L}(n)\|_1]$.

Using equation (4.10) with $f(\|\underline{L}(n)\|) = \|\underline{L}(n)\|_1$ and $V(\underline{L}(n)) = \underline{L}^T(n)T\underline{L}(n)$

$$-\frac{E\left[\underline{L}^T(n+1)T\underline{L}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)\right]}{\|\underline{L}(n)\|_1} \geq \epsilon \quad \forall \underline{L}(n) : \|\underline{L}(n)\|_1 > B \quad (4.11)$$

for some $B > 0$. The function at the left hand side of equation (4.11) admits a limit for $\|\underline{L}(n)\|_1 \to \infty$ which depends on the direction of the vector $\underline{L}(n)$. Let ϵ_{max} be the smallest value for this limit, i.e.

$$\epsilon_{max} = \liminf_{\|\underline{L}(n)\|_1 \to \infty} -\frac{E\left[\underline{L}^T(n+1)T\underline{L}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)\right]}{\|\underline{L}(n)\|_1}.$$

Substituting equation (4.2) in the above equation and observing that all the terms in the numerator that do not contain $\underline{L}(n)$ will go to zero upon dividing by $\|\underline{L}(n)\|_1 \to \infty$, we get

$$\epsilon_{max} = \liminf_{\|\underline{L}(n)\|_1 \to \infty} -\frac{2\underline{L}^T(n)\underline{\lambda} - 2\underline{L}^T(n)\underline{S}^*(n) - 2\underline{L}^T(n)\Big(\underline{S}^*(n) \odot \underline{Z}(\underline{\tau}) \odot \underline{\lambda}\Big)}{\|\underline{L}(n)\|_1}.$$

Rearranging the terms we get:

$$\epsilon_{max} = \liminf_{\|\underline{L}(n)\|_1 \to \infty} \frac{2\underline{L}^T(n)\underline{S}^*(n) + 2\underline{L}^T(n)\Big(\underline{S}^*(n) \odot \underline{Z}(\underline{\tau}) \odot \underline{\lambda}\Big) - 2\underline{L}^T(n)\underline{\lambda}}{\|\underline{L}(n)\|_1}.$$

Taking 2 as a common factor and rearranging the terms we get:

$$\epsilon_{max} = 2 \liminf_{\|\underline{L}(n)\|_1 \to \infty} \Big(\frac{\underline{L}^T(n)\underline{S}^*(n) - \underline{L}^T(n)\underline{\lambda}}{\|\underline{L}(n)\|_1} + \frac{\underline{L}^T(n)\Big(\underline{S}^*(n) \odot \underline{Z}(\underline{\tau}) \odot \underline{\lambda}\Big)}{\|\underline{L}(n)\|_1} \Big). \quad (4.12)$$

We make use of the following proposition, which was proved in [LMNM03] (Proposition A.1) and is included here for completeness.

Proposition 1. *For any nonnull normalized vector* $\hat{\underline{Z}}(n) \in \mathbb{R}^{+N^2}$*:*

$$\underline{S}^*(n)\hat{\underline{Z}}^T(n) \geq \frac{1}{N}$$

Applying Proposition 1 to the second term in equation (4.12) we get:

$$\frac{\underline{L}^T(n)\Big(\underline{S}^*(n)\odot\underline{Z}(\tau)\odot\underline{\lambda}\Big)}{\|\underline{L}(n)\|_1}\geq\frac{\underline{Z}(\tau)\cdot\underline{\lambda}}{N}\geq 0.$$

Now, we use a technique from [LMNM03](pg. 542 and 543) to bound the following term:

$$\frac{\underline{L}^T(n)\underline{S}^*(n)-\underline{L}^T(n)\underline{\lambda}}{\|\underline{L}(n)\|_1}.$$

Consider the vector $\underline{U}(n)=E[\underline{A}(n)]+(1-\|\underline{\lambda}\|_{IO})\underline{S}^*(n)$. It is straightforward to prove that $\|\underline{U}(n)\|_{IO}\leq 1$. Also, the fact that the system is stable implies $E[\underline{A}(n)]=E[\underline{S}^*(n)]=\underline{\lambda}$. Thus,

$$\frac{\underline{S}^*(n)\underline{L}^T(n)-\underline{U}(n)\underline{L}^T(n)}{\|\underline{L}(n)\|_1}=\frac{\underline{S}^*(n)\underline{L}^T(n)-\underline{\lambda}\odot\underline{L}^T(n)-(1-\|\underline{\lambda}\|_{IO})\underline{S}^*(n)\underline{L}^T(n)}{\|\underline{L}(n)\|_1}\geq 0$$

and from Lemma 1 we have

$$\frac{\underline{S}^*(n)\underline{L}^T(n)-\underline{\lambda}\underline{L}^T(n)}{\|\underline{L}(n)\|_1}\geq\frac{(1-\|\underline{\lambda}\|_{IO})\underline{S}^*(n)\underline{L}^T(n)}{\|\underline{L}(n)\|_1}.$$

Applying Proposition 1 we get:

$$\frac{\underline{S}^*(n)\underline{L}^T(n)-\underline{\lambda}\underline{L}^T(n)}{\|\underline{L}(n)\|_1}\geq\frac{(1-\|\underline{\lambda}\|_{IO})}{N}. \tag{4.13}$$

Substituting equations (4.5.1) and (4.13) in equation (4.10), we get:

$$\epsilon_{max}\geq\frac{2}{N}(1-\|\underline{\lambda}\|_{IO}). \tag{4.14}$$

The next step is to evaluate equation (4.10):

$$\lim_{n\to\infty}E\Big[f\Big(\|\underline{L}(n)\|\Big)\Big]\leq\lim_{n\to\infty}E\Big[f\Big(\|\underline{L}(n)\|\Big)+\frac{V(\underline{L}(n+1))-V(\underline{L}(n))}{\epsilon}|Y_n\in H_B\Big]\times P[Y_n\in H_B]$$

Evaluating the term $E[V(\underline{L}(n+1))-V(\underline{L}(n))|Y_n\in H_B]$ appearing in equation (4.9)

and using the result from equation (4.2)

$$\begin{aligned}
&E[V(\underline{L}(n+1)) - V(\underline{L}(n))|Y_n \in H_B] \\
&= E[\underline{L}^T(n+1)T\underline{L}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)] \\
&= 2\underline{L}^T(n)\underline{\lambda} - 2\underline{L}^T(n)\underline{S}^*(n) - 2\underline{L}^T(n)\Big(\underline{S}^*(n) \odot \underline{Z}(\underline{\tau}) \odot \underline{\lambda}\Big) \\
&\quad + \sum_{i,j} \lambda_{i,j} - 2E\Big[\sum_{i,j} S^*_{i,j}(n)\Big] - 2E\Big[\sum_{i,j} S^*_{i,j}(n) Z_{i,j}(\tau_{i,j})\lambda_{i,j}\Big] \\
&\quad + E\Big[\sum_{i,j} \frac{S^*_{i,j}(n)}{\lambda_{i,j}}\Big] + 2E\Big[\sum_{i,j} S^*_{i,j}(n) Z_{i,j}(\tau_{i,j})\Big] \\
&\quad + E\Big[\sum_{i,j} S^*_{i,j}(n) Z^2_{i,j}(\tau_{i,j})\lambda_{i,j}\Big].
\end{aligned}$$

and using the result of equation (4.5.1) and equation (4.13) we get

$$\begin{aligned}
&E[\underline{L}^T(n+1)T\underline{L}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)] \leq \\
&\quad \epsilon_{max}\|\underline{L}(n)\|_1 + \sum_{i,j} \lambda_{i,j} - 2E\Big[\sum_{i,j} S^*_{i,j}(n)\Big] - 2E\Big[\sum_{i,j} S^*_{i,j}(n) Z_{i,j}(\tau_{i,j})\lambda_{i,j}\Big] \\
&\quad + E\Big[\sum_{i,j} \frac{S^*_{i,j}(n)}{\lambda_{i,j}}\Big] + 2E\Big[\sum_{i,j} S^*_{i,j}(n) Z_{i,j}(\tau_{i,j})\Big] \\
&\quad + E\Big[\sum_{i,j} S^*_{i,j}(n) Z^2_{i,j}(\tau_{i,j})\lambda_{i,j}\Big].
\end{aligned}$$

From stability we have $E[\underline{S}(n)] = E[\underline{\lambda}]$, $E[\underline{S}^T(n)\underline{S}(n)] = \|\underline{\lambda}\|_1$, and $E[\underline{\lambda}^T\underline{S}(n)] = E[\underline{\lambda}^T]E[\underline{S}(n)] = \|\underline{\lambda}\|_2^2$. Also, $E[S_{i,j}] = \lambda_{i,j}$; so, $E\Big[\sum_{i,j} \frac{S^*_{i,j}(n)}{\lambda_{i,j}}\Big] = N^2$ because we are summing over N^2 elements and each element is 1. Similarly, $E\Big[\sum_{i,j} S^*_{i,j}(n) Z_{i,j}(\tau_{i,j})\Big] \leq N\|\underline{\lambda}\|_1$

$$\begin{aligned}
&E[\underline{L}^T(n+1)T\underline{L}(n+1) - \underline{L}^T(n)T\underline{L}(n)|\underline{L}(n)] \leq \\
&\quad \epsilon_{max}\|\underline{L}(n)\|_1 - \|\underline{\lambda}\|_1 - 2E\Big[\sum_{i,j} S^*_{i,j}(n) Z_{i,j}(\tau_{i,j})\lambda_{i,j}\Big] \\
&\quad + N^2 + 2N\|\underline{\lambda}\|_1 + E\Big[\sum_{i,j} S^*_{i,j}(n) Z^2_{i,j}(\tau_{i,j})\lambda_{i,j}\Big].
\end{aligned} \tag{4.15}$$

Substituting equation (4.15) in equation (4.10) we get:

$$E\Big[\|\underline{L}(n)\|_1\Big] \leq E\Big[\|\underline{L}(n)\|_1 + \tfrac{V(\underline{L}(n+1))-V(\underline{L}(n))}{\epsilon}|\underline{L}(n)\Big]$$

$$E\Big[\|\underline{L}(n)\|_1\Big] \leq E\Big[\|\underline{L}(n)\|_1\Big(1-\tfrac{\epsilon_{max}}{\epsilon}\Big)\Big] + \frac{N^2+2N\|\underline{\lambda}\|_1+E\Big[\sum_{i,j}Z_{i,j}^2(\tau_{i,j})\lambda_{i,j}^2\Big]-\|\underline{\lambda}\|_1-2E\Big[\sum_{i,j}Z_{i,j}(\tau_{i,j})\lambda_{i,j}^2\Big]}{\epsilon}.$$

If we set $\epsilon = \epsilon_{max}$ we get:

$$\begin{aligned}
E\Big[\|\underline{L}(n)\|_1\Big] &\leq \frac{N^2+2N\|\underline{\lambda}\|_1+E\Big[\sum_{i,j}Z_{i,j}^2(\tau_{i,j})\lambda_{i,j}^2\Big]-\|\underline{\lambda}\|_1-2E\Big[\sum_{i,j}Z_{i,j}(\tau_{i,j})\lambda_{i,j}^2\Big]}{\epsilon_{max}} \\
&\leq \frac{N^2+2N\|\underline{\lambda}\|_1+E\Big[\sum_{i,j}Z_{i,j}^2(\tau_{i,j})\lambda_{i,j}^2\Big]-\|\underline{\lambda}\|_1}{\epsilon_{max}} \\
&\leq \frac{N^2+2N\|\underline{\lambda}\|_1+N\|\underline{\lambda}\|_1}{\frac{2}{N}(1-\|\underline{\lambda}\|_{IO})} \\
&\leq \frac{N^2+3N\|\underline{\lambda}\|_1}{\frac{2}{N}(1-\|\underline{\lambda}\|_{IO})} \\
&\leq \frac{N^3+3N^2\|\underline{\lambda}\|_1}{2(1-\|\underline{\lambda}\|_{IO})}.
\end{aligned}$$

We emphasize that the bound in Theorem 4 is a much stronger property than bounding the average packet delay in an IQ switch over that in an OQ switch. Not only does Theorem 2 provide a bound on the additional mean delay for all packets departing an IQ switch using MWL over an OQ switch, it also applies to any individual packet departing the IQ switch. Specifically, Theorem 4 provides a bound on the difference between the precise packet departure sequence from an IQ using MWL over that provided by an OQ switch; for example, consider an IQ scheduling policy that periodically serves the same number of packets per output port as an OQ switch over a time interval larger than the corresponding time interval in an OQ switch. For all admissible traffic, this behaviour would imply a bounded per packet average delay compared to an OQ switch, but it does not imply the property of Theorem 4. This behaviour occurs because each packet's departure order could be different from the IQ scheduling policy compared to the OQ scheduling policy; the key difference lies in

the lag definition such that a packet departing ahead of its time would have a zero lag. Observe that if a negative lag was allowed then the mean lag value becomes the additional mean delay in an IQ switch over that in an OQ switch as packets departing ahead of their IDT (negative lag) would offset packets departing after their IDT (positive lag). Furthermore, bounding the mean delay in an IQ switch over that in an OQ switch requires only knowledge about the average service rate per output port in both switches rather than the precise packet departure sequence from each switch.

4.5.2 Iterative Lag Scheduling Policy

Iterative lag (iLag) is a simple heuristic based on maximal matching (see Section 2.3.3). iLag can be implemented using an arbiter at each input and output port using a request-grant-accept paradigm. Initially all input and output arbiters are unmatched, then in each iteration:

1. Request: Each unmatched input sends a request to every unmatched output for which it has a queued cell.

2. Grant: If an unmatched output receives any requests, it chooses the request with the most lagging cell and sends a grant to this input.

3. Accept: If an unmatched input receives any grants, it chooses the grant for its most lagging cell and sends an accept signal to this output.

The input and output arbiter are considered matched. The algorithm executes until either no more matches can be made or a fixed number of iterations are performed. The hardware implementation of iLag comprises the hardware to compute the IDTs in an OQ switch, the hardware to select the maximum lagging cells at each output arbiter, and the hardware at each input arbiter to select the maximum lagging cell and perform the accept step.

4.6 Simulation Results

The average cell delay and $E[\|\underline{L}\|_1]$ of MWL and iLag are evaluated by simulation for a 16×16 switch and compared to LPF [MM98], LQF [MMAW99], iSLIP [McK99]

and PIM [AOST93]. All simulations were performed with 99% confidence and 1% accuracy. iLag, iSLIP, and PIM were executed with 4 iterations. Bernoulli and bursty traffic distributions are used for performance evaluation.

4.6.1 Bernoulli Traffic Distribution

For Bernoulli i.i.d. distribution, we use three traffic models: uniform, log diagonal, and diagonal arrival pattern.

1. Uniform: $\lambda_{i,j} = \frac{\rho}{N} \quad \forall i, j$, where $N = 16$ is the size of the switch.

2. LogDiagonal: $\lambda_{i,j} = 2\lambda_{i,|j+1|}$, and $\sum_i \lambda_{i,j} = \rho$; for example, the distribution of the load at input 1 across all outputs is $\lambda_{i,j} = \frac{2^{N-j}\rho}{2^N-1}$. This arrival pattern is more skewed than uniform loading.

3. Diagonal: $\lambda_{i,j} = 2\rho/3$, $\lambda_{i,|i+1|} = \rho/3$ $\forall i$, and $\lambda_{i,j} = 0$ for all other i and j. This is very skewed loading and is more difficult to schedule than uniform loading.

As shown in Figure 4.2, MWL provides the lowest $E[\|\underline{L}\|_1]$ compared to other maximum weight matching schemes under uniform Bernoulli arrivals, although all maximum weight matching schemes have almost the same average cell delay as shown in Figure 4.3. The same trend occurs for iLag compared to iSLIP and PIM.

Similarly, under log diagonal traffic, MWL provides the lowest $E[\|\underline{L}\|_1]$ as shown in Figure 4.4, whereas the delay of all maximum weighted matching scheduling policies is almost identical as shown in Figure 4.5.

The same trend occurs for diagonal traffic as shown in Figures 4.6 and 4.7.

4.6.2 Bursty Traffic Distribution

Internet traffic is bursty in nature [CB97]. We considered an ON/OFF Markov Modulated Process with geometric burst size of 16. This traffic model is described in detail in [For04].

The value of $E[\|\underline{L}\|_1]$ is generally higher under bursty traffic than under a Bernoulli traffic distribution. As shown in Figure 4.8, MWL achieves the lowest lag compared to other maximum weighted matching policies, whereas their delays are almost identical

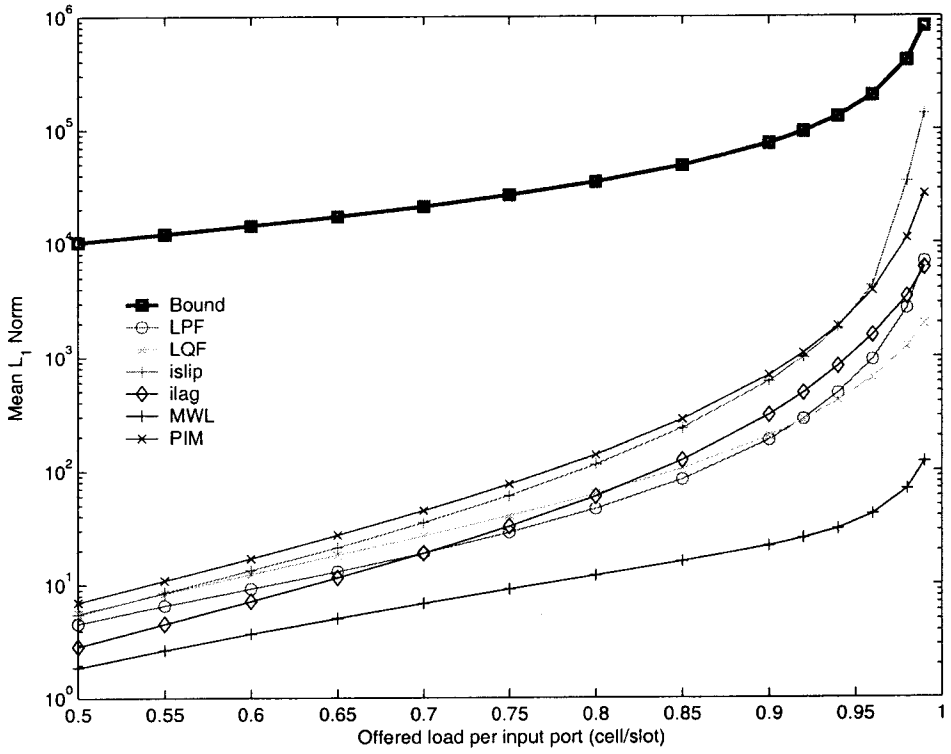


Figure 4.2: $E[\|\underline{L}\|_1]$ versus offered load for uniform Bernoulli i.i.d. traffic.

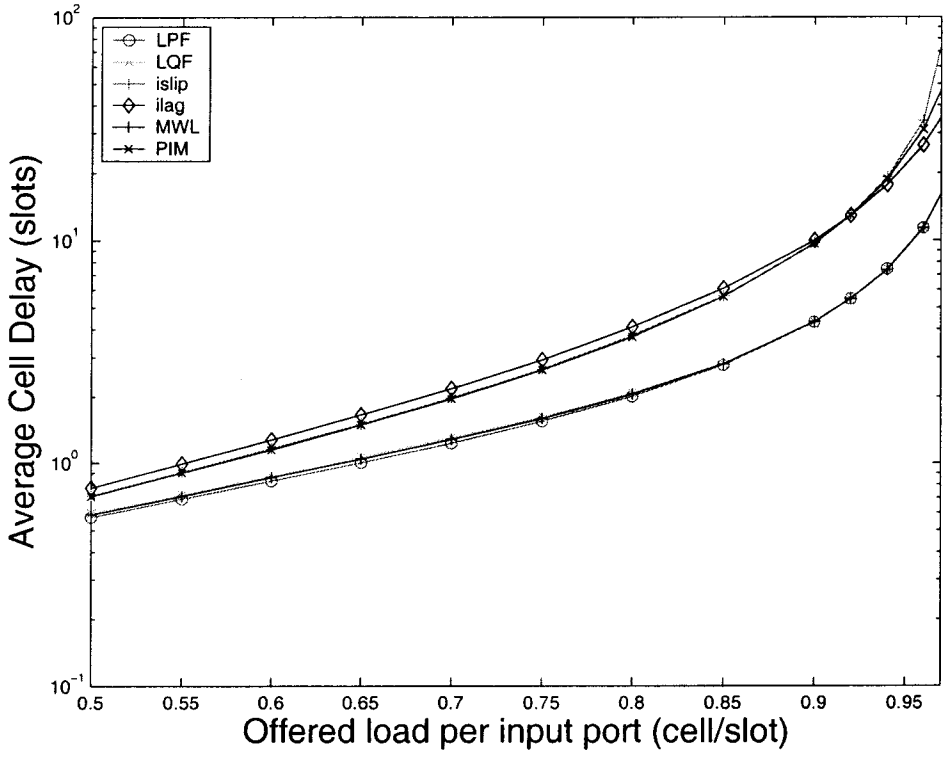


Figure 4.3: Average cell delay versus offered load for uniform Bernoulli i.i.d. traffic.

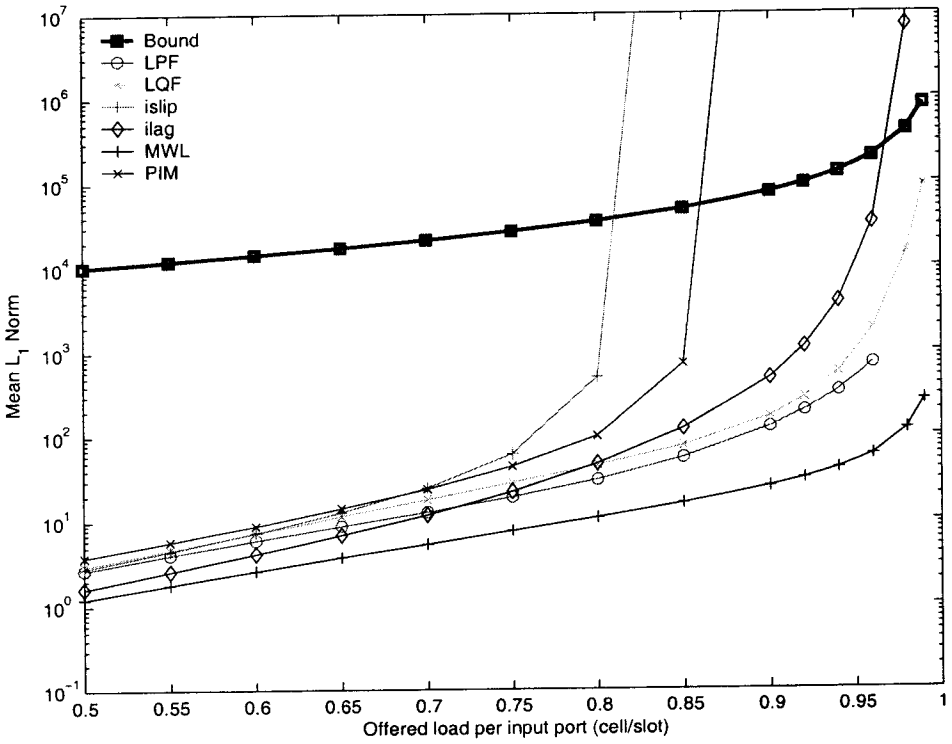


Figure 4.4: $E[\|\underline{L}\|_1]$ versus offered load for log diagonal traffic.

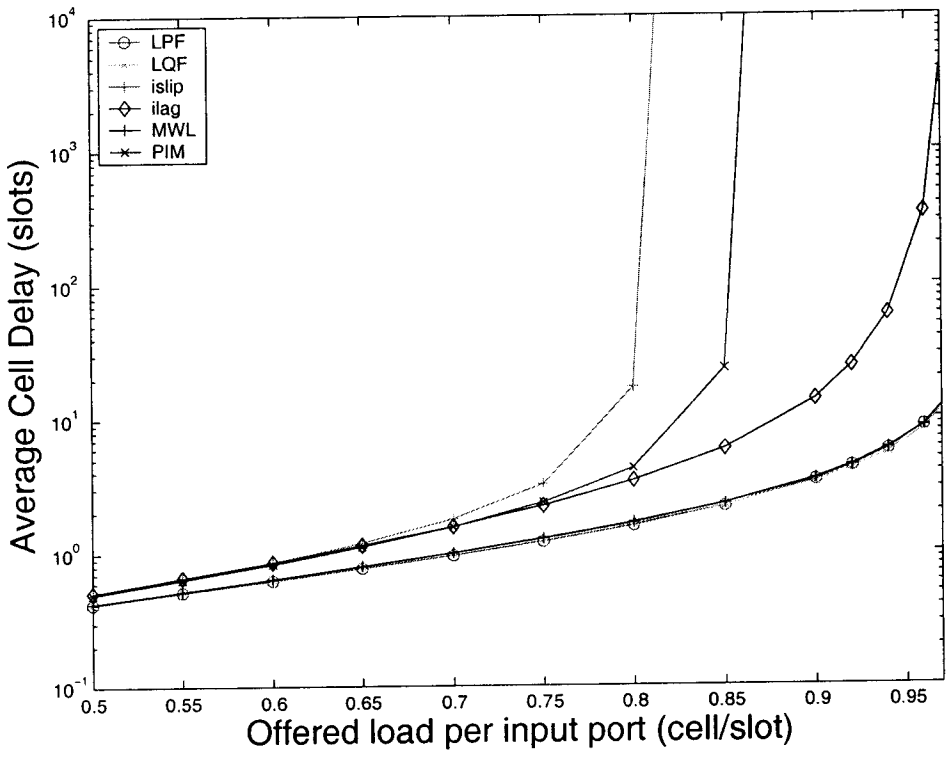


Figure 4.5: Average cell delay versus offered load for log diagonal traffic.

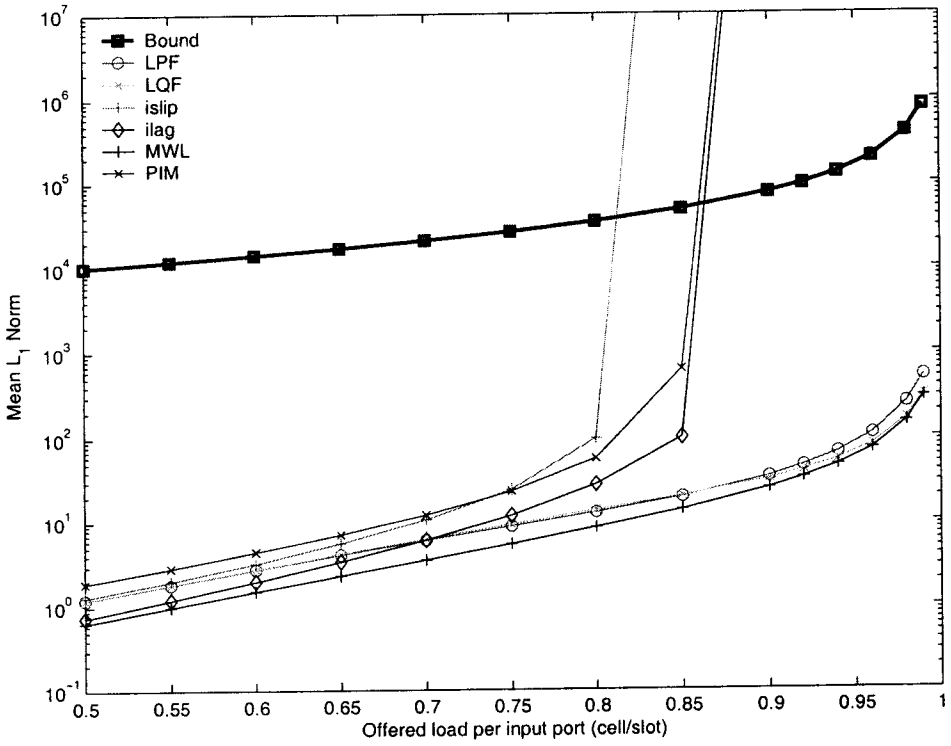


Figure 4.6: $E[\|\underline{L}\|_1]$ versus offered load for diagonal traffic.

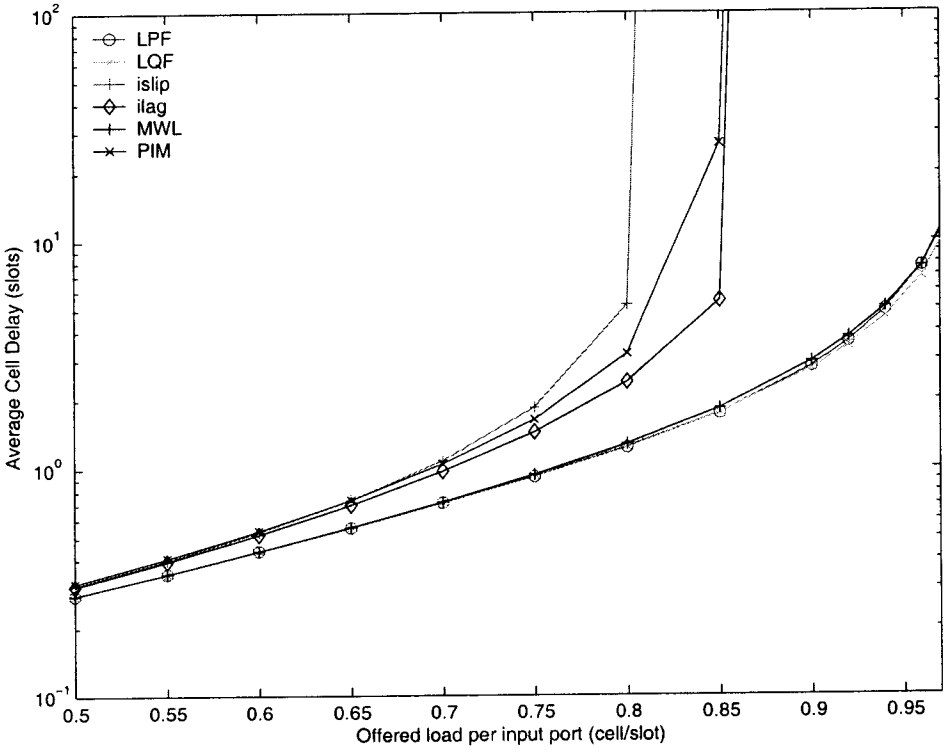


Figure 4.7: Average cell delay versus offered load for diagonal traffic.

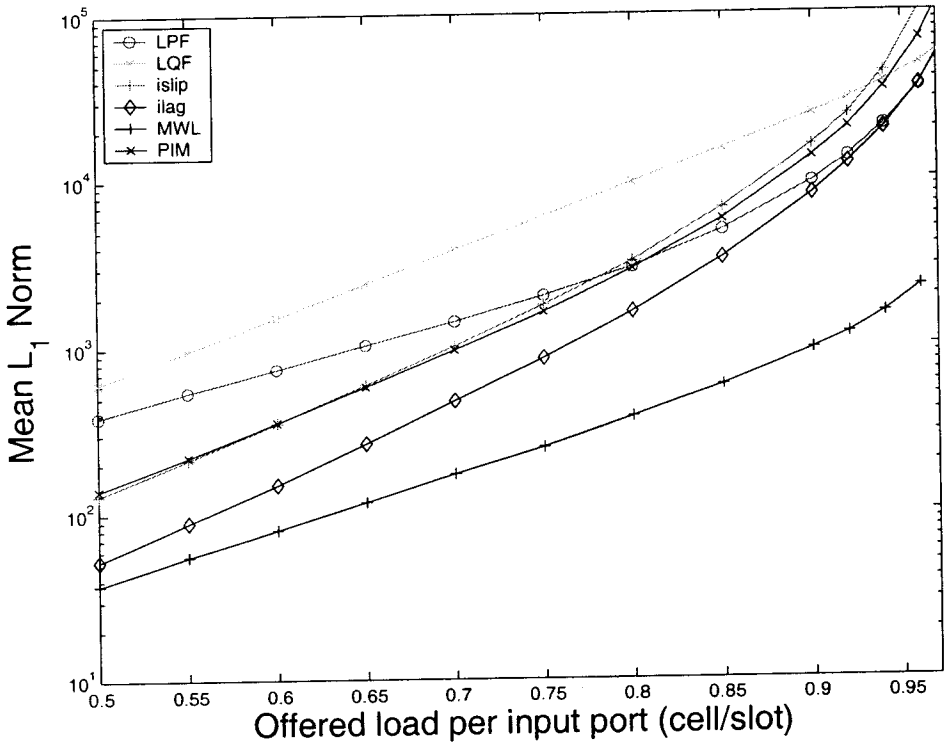


Figure 4.8: $E[\|\underline{L}\|_1]$ versus offered load for bursty traffic.

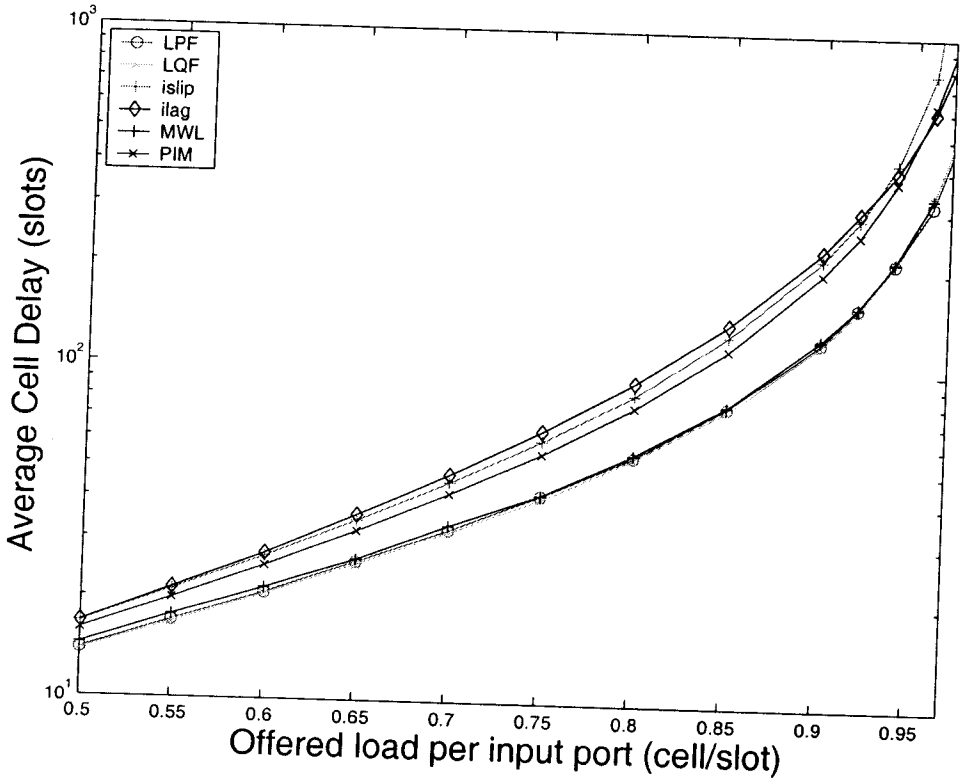


Figure 4.9: Average cell delay versus offered load for bursty traffic.

as shown in Figure 4.9. Similarly, iLag achieves the smallest lag compared to iSLIP and PIM.

4.7 Conclusion

IQ switches are commercially used in most Internet routers due to their capability of operating at high line speeds with a lower memory bandwidth requirement than OQ switches. In this chapter, we addressed fair scheduling in Internet routers with IQ switches. We formulated switch scheduling in an IQ switch with unity speedup as tracking the behaviour of an OQ switch. By tracking the behaviour of an OQ switch, an IQ switch resolves input and output contention fairly, eliminates any starvation of inputs, and approximates the behaviour of an OQ switch as close as possible. We introduced the lag as a performance metric that measures the difference between a packet's departure time in an IQ switch compared to an OQ switch. We proved that per packet lag is bounded for a maximum weighted matching scheduling policy that uses lag values for its weights and derived a bound on the mean lag value using a Lyapunov function technique. Finally, we proposed a simple heuristic tracking scheduling policy and evaluated its performance by simulation.

Chapter 5

Cooperative Token-Ring Scheduling

In this chapter we present a novel distributed scheduling paradigm for Internet routers with IQ switches, called cooperative token-ring (CTR) that provides significant performance improvement over existing scheduling schemes with comparable complexity. In classical token-ring based scheduling for IQ switches, a separate token-ring (an arbiter) is used to resolve contention for each shared resource (e.g., an output port). Although classical token-ring based scheduling achieves fairness and high throughput for uniform traffic, under non-uniform traffic the performance degrades significantly. We show that by using a simple cooperative mechanism between the otherwise non-cooperative token-rings (arbiters) the performance can be significantly improved and the scheduler is able to dynamically adapt to non-uniform traffic patterns. In addition, our proposed CTR scheduling policy potentially amortizes the cost of arbitration time over multiple time slots, such that tokens are exchanged only on as-needed basis. The proposed cooperative mechanism is conceptually simple and is supported by experimental results. To provide adequate support for rate guarantees in IQ switches, we present a weighted cooperative token-ring (WCTR), a simple hierarchical scheduling mechanism. Finally, we analyze the hardware complexity introduced by cooperative mechanism and describe an optimal hardware implementation with time complexity of $\Theta(\log N)$ and circuit size of $\Theta(N \log N)$ per node.

5.1 Introduction and Related Work

Most commercial high-performance switches and routers (e.g., CISCO 1200[Cis04], BBN [PCB+98]) employ IQ switches because an IQ switch requires its fabric and memory to run only as fast as the line rate, which makes IQ very appealing for switches with fast line rates and/or with a large number of ports. A VOQ architecture and a crossbar as the switch fabric are typically used such that a scheduling algorithm configures the crossbar during each time slot and decides which inputs are connected to which outputs.

Most practical schedulers are based on simple heuristics, which are readily implemented in hardware, that aim at maximizing the number of connections between inputs and outputs and achieving a maximal match using a ΠRGA scheduler (See Section 2.4) like PIM [AOST93], WPIM [SV95], iSLIP [McK99], etc. Most maximal matching based scheduling algorithms perform well under uniform traffic, but the performance degrades under non-uniform traffic; for example, iSLIP uses rotating round-robin priority arbiters at the inputs and outputs such that under uniform traffic, the pointers used in the input and output arbiters for selection tend to point to different elements (*desynchronize*) and each arbiter tends to make a different selection from other arbiters and the largest number of inputs and outputs are matched. Consequently, under uniform Bernoulli i.i.d. traffic iSLIP arbiters adapt to a time-division multiplexing scheme, providing a perfect match and 100% throughput. However, under non-uniform traffic, the pointers are not necessarily desynchronized and the performance potentially degrades – Chang et al. [CLJ02] showed using a pathological traffic pattern for a 3×3 switch how iSLIP can get trapped in "bad modes" such that the throughput is limited to 66.67%.

To cope with degrading performance under non-uniform traffic, without increasing the scheduler's complexity, Li et al. [LPC02] proposed coupling the ΠRGA paradigm with *exhaustive matching*(EM) (see Section 2.5). Specifically, it was shown [LPC02] that exhaustive iSLIP (EiSLIP) produces the best results compared to several proposed exhaustive scheduling algorithms and performs better than non-exhaustive matching algorithms, under some non-uniform traffic patterns.

Load-balanced Birkhoff-von Neumann [CLJ02] switches (see Section 2.7) address the problem of scheduling non-uniform traffic using a two stage scheduler: a load balancing stage followed by a second scheduling stage that essentially operates on uniform traffic. The main drawback of this architecture is that packets can be missequenced, which may require complicated hardware implementation and non-scalable computation overhead. Furthermore, providing a scalable solution that simultaneously provides QoS support and solves the packet missequencing problem is a major difficulty in the load-balanced router architecture.

5.2 Problem Addressed

In summary, it is a challenge to find a scheduling scheme for IQ switches that meets the following requirements:

1. Provides high throughput for both uniform and non-uniform traffic.
2. Provides rate guarantees for QoS traffic and proportional bandwidth sharing.
3. Is readily implemented in hardware: most practical schedulers are iterative with hardware time complexity of $\Theta(\log N)$ per iteration, where N is the size of the switch; usually $\log(N)$ iterations are used in practice.

In this chapter we address the previous issues and present a solution that meets all these requirements. We emphasize that almost all practical scheduling schemes in the literature can provide high throughput under uniform traffic; however, under non-uniform traffic, the throughput usually degrades significantly.

This chapter is organized as follows. Section 5.3 provides an overview of the proposed cooperative token-ring scheduling policy. In Section 5.4, we present all the algorithmic details of the proposed CTR scheduler. We present a parallel implementation of CTR based on ΠRGA paradigm in Section 5.5. The performance of CTR, for best-effort traffic, is evaluated by simulation in Section 5.6. In section 5.7, we examine the fairness of the proposed CTR scheduler. We propose a two-level hierarchical scheduler, weighted CTR scheduler, which supports rate-guarantees and

proportional bandwidth sharing in Section 5.8. In Section 5.9, we provide an optimal hardware implementation for the proposed cooperative mechanism. Several detailed examples of CTR scheduling policy are presented in Section 5.10. Finally, Section 5.11 provides our conclusions.

5.3 Overview of Cooperative Token-Ring Scheduling

In this section we informally describe the cooperative token-ring (CTR) scheduling policy. The goal is to provide an intuitive understanding of the concept rather than to list the algorithmic details, which are given in Section 5.4.

Consider the system shown in Figure 5.1 (a) There are a set of four users (nodes) that are alphabetically labeled A, B, C, and D. There are four resources, which are represented by the tokens T_1, T_2, T_3, and T_4. These tokens rotate clockwise in the ring and could be acquired by any of the nodes subject to the constraint that each node acquires at most one resource simultaneously. Each node maintains a separate queue for each token that represents backlogged work for that resource. We assume that time is slotted such that token-arbitration is performed during each time-slot where each node may acquire or release an acquired token. At the end of token-arbitration each node may be matched to at most one token and consumes an element from the corresponding queue. Consider the configuration shown in Figure 5.1(a) where each of the four nodes has backlogged queues for some resources, which are represented by the rectangle boxes outside the ring: node A requires tokens T_1 and T_2; node B requires tokens T_1; node C requires tokens T_3 and T_4; and node D requires token T_4. The initial token(s) position(s) are as shown in Figure 5.1(a): T_1 is at node D; T_3 and T_4 are at node B; and T_2 is at node C.

In classical token-ring scheduling each node makes an independent token-selection decision obliviously of the state of other nodes; for example, given the initial state shown in Figure Figure 5.1 (a) each node could acquire the first available token to result in the matching state shown in Figure 5.1 (b) where: node A acquires token T_1, and node C acquires token T_4. The resource utilization in this example using classical

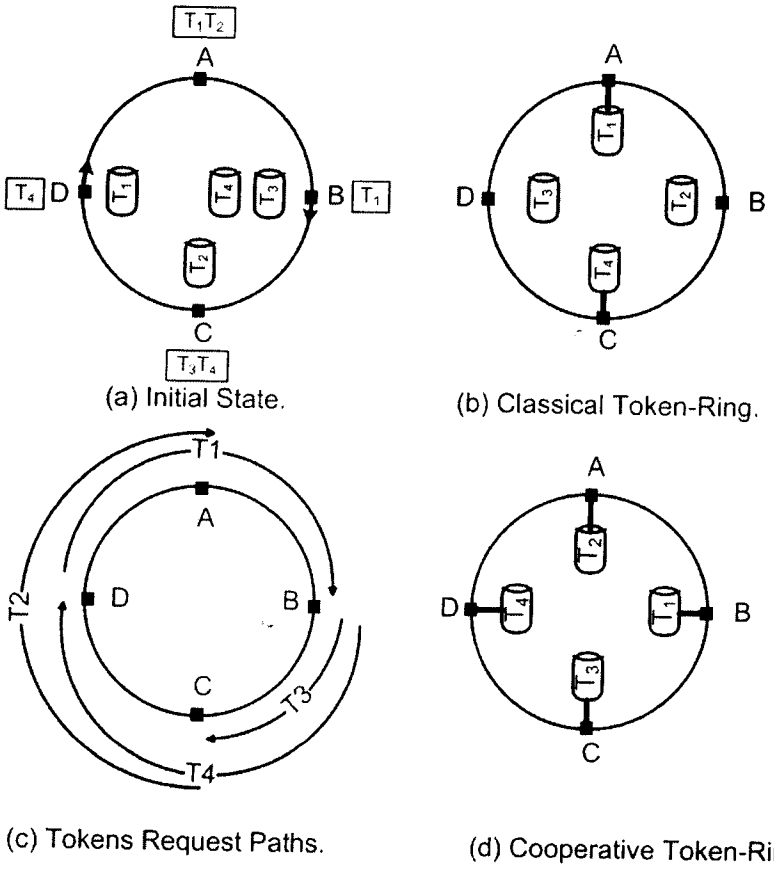


Figure 5.1: Scheduling Using Classical Token-Ring and Cooperative Token-Ring.

token-ring scheduling is 50%. Note that nodes B and D do not require tokens T_2 and T_3.

CTR is an iterative scheme such that each iteration comprises two phases: computing a request path for each token, and token propagation/selection. The main idea of CTR is to create a guided path along the ring for each token from its current position to the last node in the ring that requires that token and is not matched to any other token. The guided paths for the initial token configuration in Figure 5.1 (a) are pictorially shown in Figure 5.1 (c); for example, the guided path for token T_1 starts at node D and ends at node B, which is the last node that is not matched to any other token and requires token T_1. The value of each token request path at each node is a Boolean variable that indicates whether this token is requested by some other nodes along path. Subsequently, tokens propagate through the ring such that each node uses the token request paths to decide whether to acquire, swap, or release a token to improve the overall resource utilization. Specifically, token propagation/selection is performed at each node to achieve two goals:

1. Attempt to improve the node's resource utilization. If the node is not matched (has not acquired any token yet) then it acquires the first available token that it needs regardless of whether this token is requested by other nodes along the path.

2. Attempt to improve the overall resource utilization of the ring by swapping its acquired token for another unrequested token. This swapping is performed using the token request paths' information that have been previously computed.

After computing the guided paths in Figure 5.1 (c), tokens propagate in the ring. When node A receives token T_1, it acquires it. When tokens T_3, and T_4 arrive at node C, node C acquires T_3 because T_4 is requested by some other node along the path as indicated by the request path for T_3. Subsequently, token T_4 propagates along the ring and is acquired by node D. When T_2 arrives at node A, which had previously acquired T_1, node A swaps T_1 for T_2 and T_1 propagates to node B, where it gets acquired. Note that A performs the swapping because it knows that T_1 is requested

by some other node along the ring. The final state is shown in Figure 5.1 (d), where each node is matched to a token and the resource utilization is 100%.

In essence, the main difference between traditional token-ring and cooperative token-ring is that each node in traditional token-ring scheduling considers only its own resource utilization, whereas in CTR the token-selection at each node additionally *cooperates* with other nodes in the ring to improve the overall resource utilization.

5.4 Description of Cooperative Token-Ring Scheduler

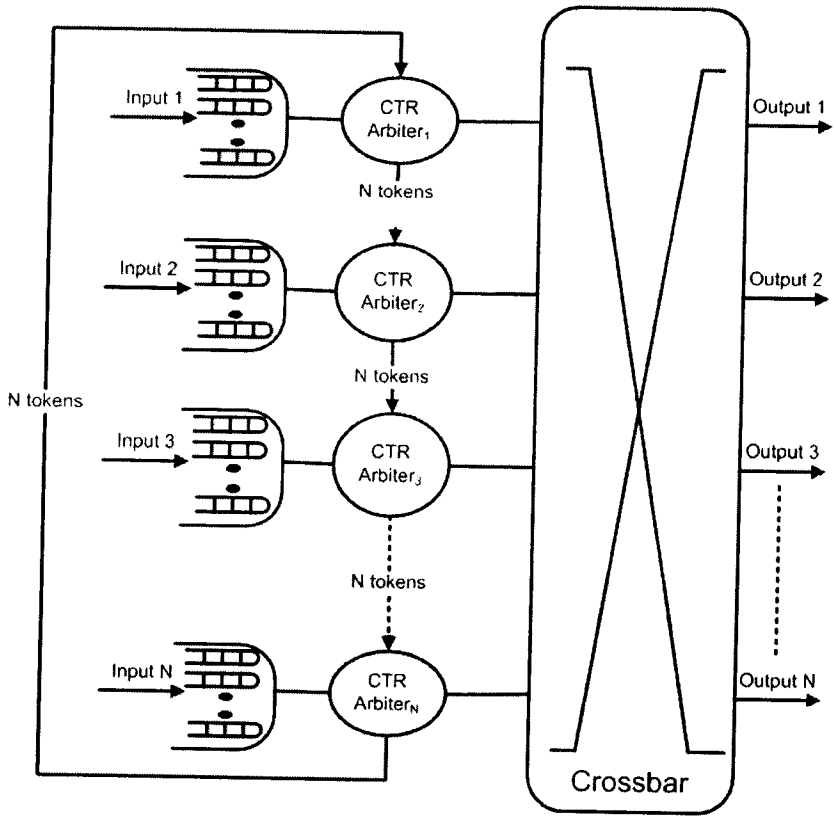


Figure 5.2: Architecture of Cooperative Token-Ring Switch.

The basic architecture of a cooperative token-ring switch is shown in 5.2. There

are N tokens in the ring that correspond to the N outputs of the switch such that each CTR arbiter is allowed to acquire at most one token. When input i is matched to output j in a time slot it is allowed to transmit a packet to output j during that time slot.

CTR is an iterative algorithm such that each iteration comprises two phases:

1. Computing the tokens request paths phase: In this phase a token request vector (TRV) is computed for *each* input that represents which tokens are requested by other unmatched inputs. Each element in the TRV vector is a binary value such that a true value for $TRV(i)$ indicates that token i is requested by some other unmatched input along the ring. Computing the TRV is described in Section 5.4.1 and its hardware complexity is examined in Section 5.9.1.

2. Token propagation/selection phase: In this phase tokens propagate through the ring and each CTR arbiter selects tokens based on its VOQ status and TRV.

Here we make precise some terminology used for the remainder of this chapter. We adopt a matrix representation to represent the switch's state. We use the following standard notations:

$+$ denotes standard Boolean OR operation.

$\times$ denotes standard Boolean AND operation.

1 denotes true and 0 denotes false.

$\overline{A}$ denotes Boolean NOT operator applied to the Boolean parameter A.

Definition 23. *The VOQ state matrix* **VOQ**. *$VOQ_{i,j}$ is set to one if the virtual output queue at input i for output j is nonempty, and is set to zero otherwise.*

Definition 24. *The matched matrix* **M**. *$M_{i,j}$ is set to one if input i is currently matched to output j.*

Definition 25. *The request matrix* **R**. *$R_{i,j}$ is set to zero if input i is matched and is set to $VOQ_{i,j}$, otherwise. The request matrix is used for computing the token request paths.*

Definition 26. *The token position* **TP**. *$TP_{i,j}$ is set to one if the token for output j is currently at input i and is set to zero, otherwise. Note that multiple tokens can be at the same input, and also when token j is at node i does not necessarily imply that input i is matched to output j.*

Definition 27. *The token request paths matrix* **TRP**. *Let k be position of the token j in the ring: Then $TRP_{i,j}$ is set to one if there exists an input y between (circular wise) input i (exclusive) and k (exclusive) that is not matched and $VOQ_{y,j} = 1$. Observe that each row in* **TRP** *represents the TRV for the corresponding input; i.e., row i is the TRV for input i. The goal of the first phase of CTR scheduling is to compute the* **TRP** *matrix.*

Our design strategy is to have a communication structure that is feasible to implement in hardware and that could be used to iteratively improve the throughput of the switch such that a tradeoff could be made between the number of iterations performed and the achieved throughput. Coincidentally, we would like the communication mechanism to be as concise as possible and reflect the dynamic nature of the traffic conditions such that the scheduler is able to dynamically adapt to time-varying traffic, which manifests itself in the status of VOQs, such that little or no exchange of tokens is performed between the different arbiters when the status of VOQs do not change and more communication is performed when the status of the VOQ change and arbiters become unmatched that could potentially be matched. Unequivocally, the TRV at each input can be viewed as forming guided paths for the tokens to reach their intended destinations that lead to an overall performance improvement. Computing the token request paths and token-selection phases are described in detail in Sections 5.4.1 and 5.4.2, respectively.

5.4.1 Computing the Tokens Request Paths Phase

The TRV computed at each input represents the set of tokens that are requested by other unmatched inputs along the ring and is used by each CTR arbiter during the token selection phase as described in Section 5.4.2. Each element in the TRV is a binary value that is set to true if there is an unmatched arbiter along the ring that requests this token.

The same algorithm is used to compute the token request path for each token in the ring (i.e., a column in the **TRP** matrix). To simply the notation, we focus on computing the token-request path along one ring and drop the second subscript.

Assume a token-ring with N nodes and N tokens. Let $|k| = (k \ mod \ (N+1))$.

The value of TRP at input module i is given by:

$$TRP_i = R_{|i+1|} + \sum_{j=i+2}^{j=i+N-1} R_{|j|} \prod_{k=i+1}^{k=j-1} \overline{TP}_{|k|} \tag{5.1}$$

Various implementation schemes could be used to compute **TRP** based on equation (5.1). One possible implementation scheme is to exploit the ring structure and send the requests in opposite direction of token propagation such that each node computes its TRP bit and either propagates or stops the request based on the token's position. The time complexity using this technique is $\Theta(N)$. In section 5.9, we describe how a binary tree structure could be used to evaluate equation (5.1) in $\Theta(\log N)$ time. We emphasize that computing TRP requires simple boolean operations that is readily implementable in hardware.

5.4.2 Token Propagation/Selection Phase

Each CTR arbiter performs token-selection using its computed TRV. We say that a token is requested if its corresponding element in the TRV is true and is unrequested otherwise. A precondition for acquiring token by an input is that the corresponding VOQ is nonempty.

Each CTR arbiter selects tokens according to the following rules:

$\mathcal{R}1$ An input that is not matched acquires the first available token regardless of whether this token is requested or not – this ensures that the matching converges.

$\mathcal{R}2$ Acquiring an unrequested token is prioritized over acquiring a requested token.

$\mathcal{R}3$ Swapping an acquired token with an unrequested token, when possible.

$\mathcal{R}4$ An input arbiter that still has backlogged packets for its acquired token, can hold its acquired token for more than one time slot.

The Prioritization according to [$\mathcal{R}2$] is done to provide other unmatched inputs the chance to acquire the requested token and improve the overall throughput.

[$\mathcal{R}3$] allows two cases for token-swapping: swapping a requested token for a non-requested token as in [$\mathcal{R}2$]; and swapping a non-requested token for *another* unrequested token. Swapping between unrequested tokens allows the breaking of cyclic dependencies; for example consider a token-ring with three nodes: A, B, and C such that C is not matched and requests a token that is acquired by B. In turn, B would relinquish its acquired token only if it acquires the token that is acquired by node A. According to Definition 25, node B can not send a request for the token acquired by node A because B is already matched; however, node A would swap its acquired token according to [$\mathcal{R}3$], which in turn would be acquired by B in exchange for the token required by node C to achieve 100% utilization. A detailed example that shows how swapping unrequested tokens could break a cyclic dependency is provided in Section 5.10.5.

[$\mathcal{R}4$] is based on the observation that the state of the VOQs changes slightly between time slots. So, rather than starting each matching from scratch at the beginning of each time slot, [$\mathcal{R}4$] attempts to improve over the matching computed from the previous time slot.

There are various mechanisms for implementing a CTR scheduler with implementation tradeoffs. We emphasize our description so far has been only a logical description: any hardware implementation that logically implements the CTR scheduler could be used; for example, in Section 5.5, we describe how CTR could be physically implemented using a ΠRGA paradigm, which is typically employed in high-speed IQ switch implementation [GM99].

5.5 Parallel Implementation of CTR

In this section we present a parallel implementation of cooperative token-ring scheduler that is tailored towards high-speed implementation with a hardware time complexity of $\Theta(\log N)$ per iteration based on ΠRGA paradigm.

At the beginning of each iteration the TRV is computed for each input as described in Section 5.4.1 and the Token Propagation/Selection Phase is performed using ΠRGA paradigm as described next.

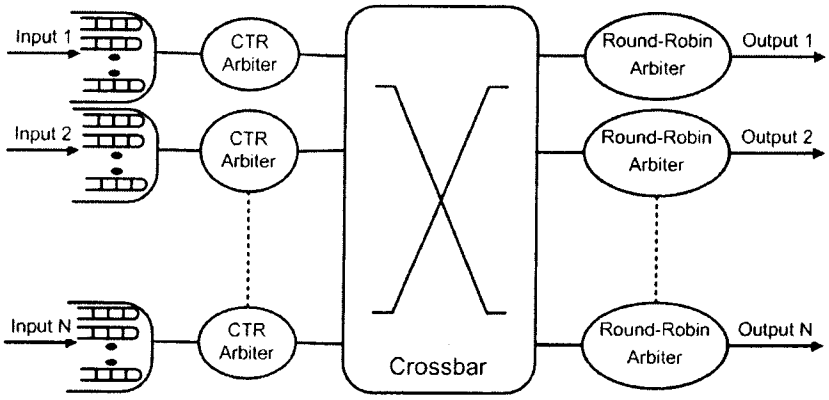


Figure 5.3: Parallel Implementation of Cooperative Token-Ring.

As shown in Figure 5.3, a round-robin arbiter is used at each output port, and a CTR arbiter is used at each input. The round-robin arbiter at each output implements the token-ring for the corresponding output, whereas the CTR arbiter implements the token-selection described in Section 5.4.2; the CTR arbiter implements the request and accept phase, and the round-robin arbiter implements the grant phase of the ΠRGA paradigm. Each CTR arbiter and round-robin arbiters uses a rotating round-robin priority encoder as described next. Specifically, each iteration of the CTR algorithm comprises the following steps:

1. Compute the TRV for each input as described in Section 5.4.1.

2. Request Step: Each *unmatched* CTR arbiter sends a request to every output arbiter for which it has a queued cell, whereas each *matched* input sends a request to every *unrequested* and unmatched output [1] for which it has a queued cell. Recall that an unrequested output is one for which the corresponding element in TRV is false.

3. Grant Step: If an unmatched output arbiter receives any requests, it chooses the one that appears next in fixed round-robin fashion starting with the highest priority element. The output notifies each input whether or not its request was

[1] Technically, it is irrelevant whether an input sends a request to a matched output because per definition a matched output ignores the requests it receives, but it helps simplify our presentation.

granted. The pointer to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the granted input.

4. Accept-step: Each CTR arbiter selects one of the grant signals and sends an accept signal to the corresponding output arbiter. Selecting a grant signal follows [$\mathcal{R}1$] - [$\mathcal{R}3$] described in Section 5.4.2, which are are reiterated here for completeness. There are two cases:

   (a) Unmatched Input: Select a grant for an unrequested output (i.e., the corresponding bit in TRV is zero), if possible; otherwise, select a grant for a requested output and send the accept signal starting with the highest priority element. The corresponding input and output are considered matched. The round-robin pointer is incremented (modulo N) to one location beyond the accepted output.

   (b) Matched Input: Per definition, the received grants are for unrequested outputs (requests were sent only for unrequested outputs in the Request Step) and the output selects from among these grants in a round-robin fashion starting with highest priority element – the CTR arbiter uses a rotation round-robin priority. The input resets (breaks) its previously matched output and sends an accept signal (is matched) to the selected output. The round-robin pointer is incremented (modulo N) to one location beyond the accepted output.

Most ΠRGA typically converge to a maximal matching (See Section 2.3.3) after $\log N$ iterations, on average [McK99], although this convergence has never been formally proven in the literature for any of the deterministic ΠRGA schemes (those schemes that do not use random selection). The only established formal convergence in the literature is for PIM, which uses random selection at both the inputs and outputs and was shown to converge to a maximal matching after $\log N$ iterations, on average, under Bernoulli i.i.d. traffic [AOST93]. In the worst case a ΠRGA requires N iterations – every iteration matches only a single input to an output [AOST93]. Similar to other ΠRGA, CTR typically converges to a maximal matching after $\log N$

iterations on average; we conjecture that CTR converges to maximum size matching (See Section) after N iterations; however, establishing an analytic proof of this convergence is an area of future work.

5.6 Simulation Results for Best Effort Traffic

In this section, we evaluate the performance of CTR, iSLIP, EiSLIP, Dual Round-Robin (DRR), and PIM for a 16×16 switch with four iterations. All simulations were performed with 99% confidence and 1% accuracy; that is, the simulations were run until the relative width of the confidence interval equals 1% with probability $\geq$ 99%. The simulations were implemented using a Java testbed (30KLOC) and were executed on an IBM Blade Center; the confidence interval calculations were done using the batch method with a batch size of 1000 time slots and were calculated only after the system reached steady state (i.e., arrival rate equals departure rate). On average each simulation run required four hours of execution time. We evaluate the performance for both Bernoulli traffic distributions and various bursty traffic models.

5.6.1 Bernoulli Traffic Distribution

We use various traffic models recommended by the switching fabric benchmarking group [For04]. The following arrival patterns are used with Bernoulli traffic distribution. Note that ρ denotes the normalized load such all inputs are equally loaded, and N is the switch size.

1. Uniform: $\lambda_{i,j} = \rho/N \ \forall i, j$.

2. Diagonal: $\lambda_{i,j} = 2\rho/3$, $\lambda_{i,|i+1|} = \rho/3 \ \forall i$, and $\lambda_{i,j} = 0$ for all other i and j. This is very skewed loading and is more difficult to schedule than uniform loading.

3. Logdiagonal: $\lambda_{i,j} = 2\lambda_{i,|j+1|}$, and $\sum_i \lambda_{i,j} = \rho$; for example, the distribution of the load at input 1 across outputs is $\lambda_{i,j} = \frac{2^{N-j}\rho}{2^N-1}$: This type of load is more balanced than diagonal loading, but more skewed than uniform loading.

Figures 5.4, 5.5, and 5.6 show the average delay under uniform, log diagonal and diagonal traffic, respectively. In addition to providing the best performance under the three arrival patterns, the improvement achieved by the proposed CTR scheduling policy manifests itself clearly as the arrival pattern becomes more skewed: under uniform arrivals, all schemes can support up to 100% traffic load and CTR provides the lowest delay; as the arrival pattern becomes more skewed under logdiagonal traffic, only CTR is able to provide almost 100% throughput for traffic load larger than 90%; finally, under the diagonal arrival pattern, which is the most skewed arrival pattern, the breakpoint at which both iSLIP and EiSLIP can not handle the traffic load moves further to the left and only CTR is able to provide 100% throughput for traffic loads larger than 85%.

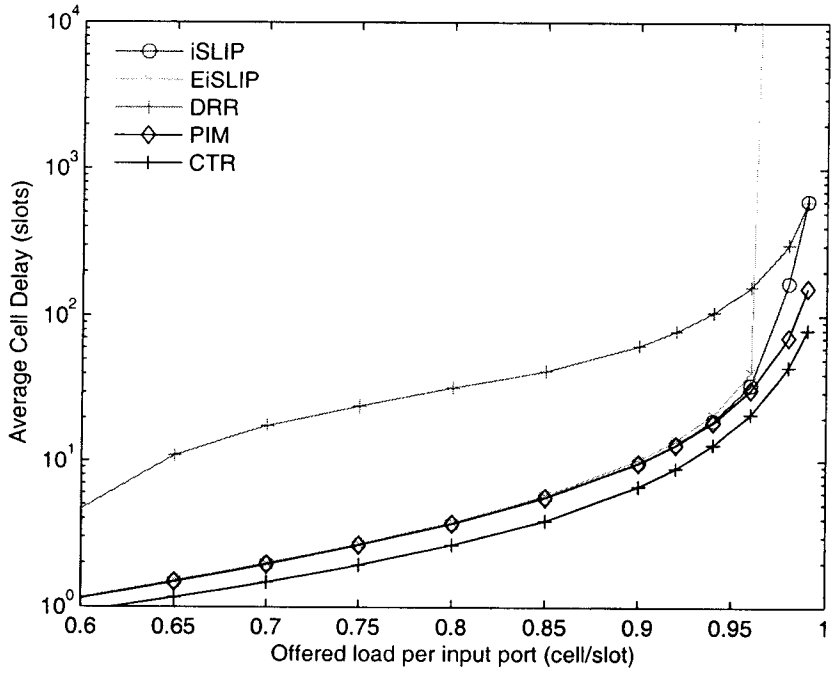


Figure 5.4: Performance of CTR, iSLIP, DRR, PIM, and EiSLIP for uniform traffic.

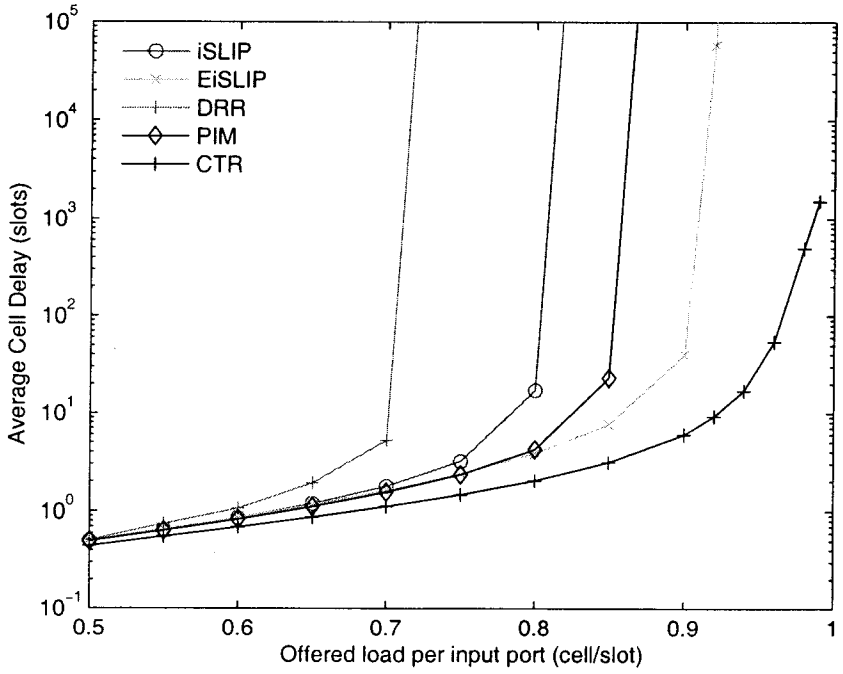


Figure 5.5: Performance of CTR, iSLIP, and EiSLIP for Log Diagonal traffic.

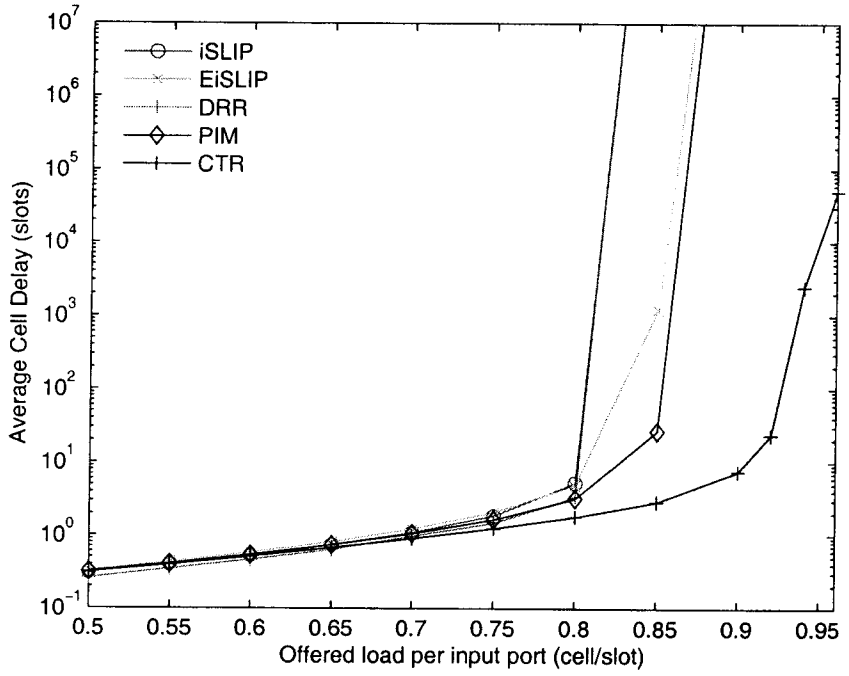


Figure 5.6: Performance of CTR, iSLIP, DRR, PIM, and EiSLIP for Diagonal traffic.

5.6.2 Simulation as a function of the switch size

Figure 5.7 shows the average latency imposed by a CTR scheduler as a function of offered load for switches with 4, 8, 16, and 32 ports for Bernoulli uniform traffic with $\log(N)$ iterations. The performance is almost identical for the various switch sizes.

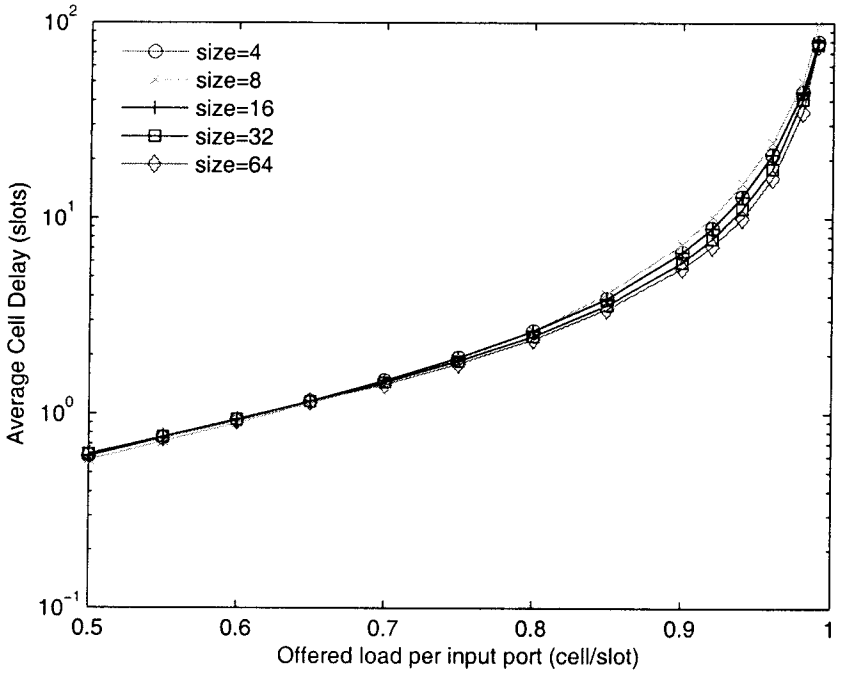


Figure 5.7: The performance of CTR as a function of switch size for uniform i.i.d. Bernoulli arrivals.

5.6.3 Bursty Traffic Distribution

Because real-internet traffic is bursty [CB97], bursty traffic models were considered in our simulations. Specifically, a bursty traffic with fixed burst size, and an ON/OFF Markov Modulated Process with a geometrically distributed burst size [For04] were used.

5.6.4 Uniform Bursty Traffic

In this traffic model packets come as a burst of length N (i.e., 16) such that packets within the same burst are destined to the same output. For every Nth time slot, the probability that there is a burst arriving at a particular input port is ρ, and the probability that there are no packet arrivals in these N slots is $1-\rho$. The destination of the N packets within a burst is chosen uniformly from among the N output ports.

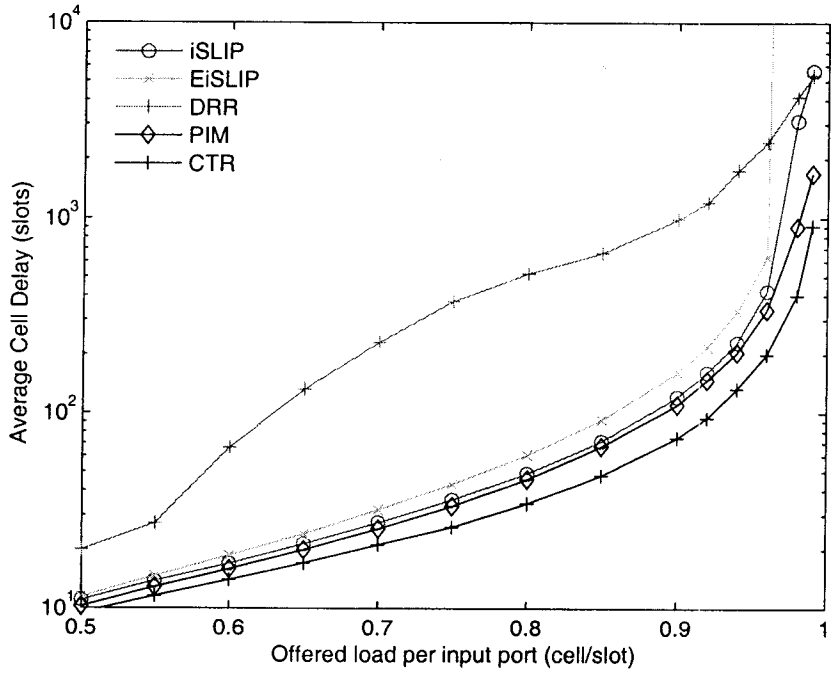


Figure 5.8: Average Delay under the uniform bursty traffic model.

As shown in Figure 5.8, CTR provides the lowest cell delay followed by iSLIP and finally by EiSLIP.

5.6.5 ON/OFF Markov-Modulated Arrivals

Each input port is connected to a burst source that generates traffic-cells using a 2-state Markov process that alternates between busy and idle states. The process

remains in the busy and idle states for a geometrically distributed number of cell times. When the server is in the busy state, cells arrive at the beginning of every time slot and all with the same set of destinations. This traffic model is described in detail in [For04]. An average burst size of 16 was used.

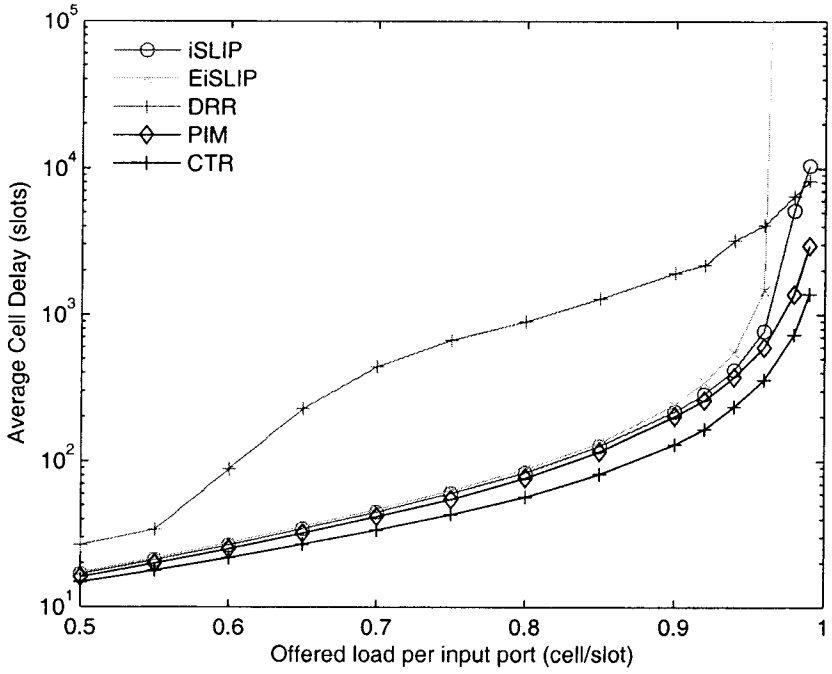


Figure 5.9: Average Delay under 2-state Markov-modulated arrivals with an average burst size of 16.

As shown in Figure 5.9, the same trend occurs and CTR provides the best performance.

5.6.6 Effects of increasing number of iterations

One of the main arguments for CTR is that its performance iteratively improves with increasing the number of iterations. The effect of increasing the number iterations is evaluated by simulation for CTR, iSLIP, EiSLIP, Dual Round-Robin (DRR), and

PIM. A 16×16 switch was used and the number of iterations executed were: 1, 2, 4, 8, and 16.

5.6.6.1 Uniform Bernoulli Traffic

As shown in Figure 5.10, most schemes perform well under uniform Bernoulli i.i.d. traffic and there is almost no improvement achieved by executing more than 4 iterations for most schemes except for CTR.

5.6.6.2 Log Diagonal Bernoulli Traffic

As shown in Figure 5.11, CTR outperforms all other schemes for any number of iterations under log diagonal traffic and is the only scheme able to sustain 100% traffic load even with a single iteration. Observe that regardless of executing more iterations, none of schemes other than CTR is able to sustain a traffic load larger than 90%.

5.6.6.3 Diagonal Bernoulli Traffic

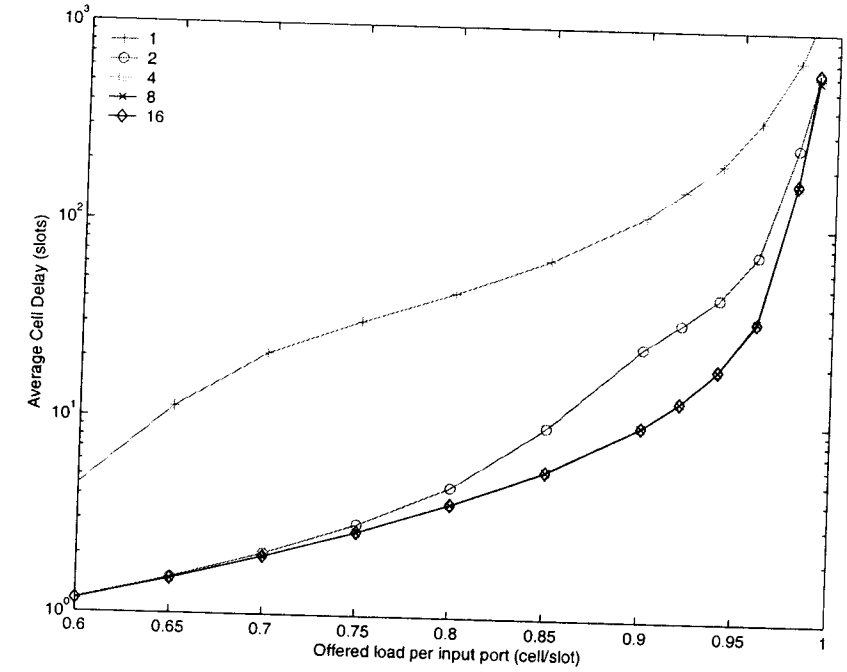
As the traffic becomes very skewed under diagonal traffic, none of the schemes other than CTR is able to sustain traffic load higher than 85%, whereas CTR provides 100% throughput with 4 iterations as shown in Figure 5.12. In addition, the performance of CTR keeps incrementally improving as more iterations are executed, as expected.

5.6.6.4 Uniform Bursty Traffic

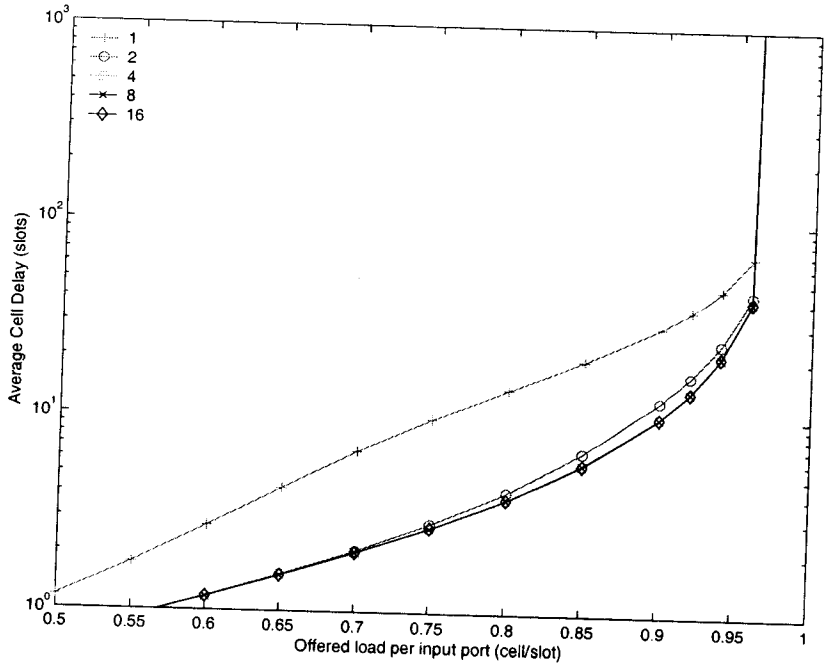
As shown in Figure 5.13, CTR outperforms all other schemes even with a single iteration.

5.6.7 ON/OFF Markov-Modulated Traffic

As shown in Figure 5.14, CTR outperforms all other schemes even with a single iteration under bursty traffic model with a geometrically distributed burst size of 16.

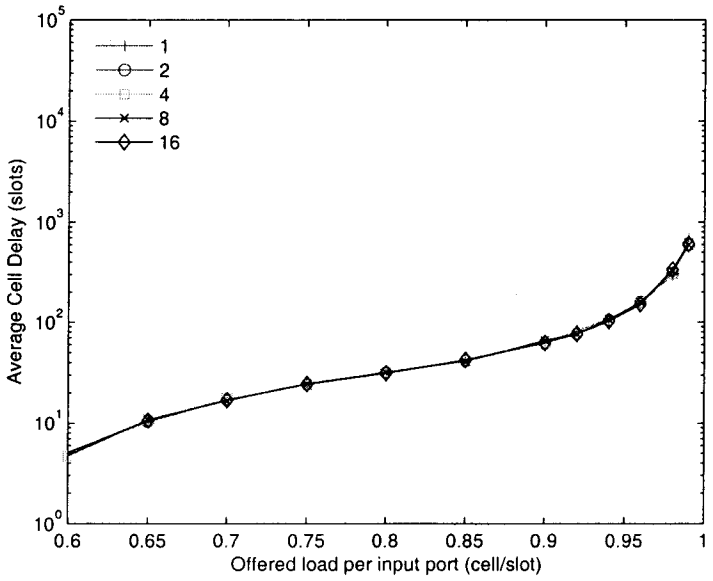


(a) iSLIP

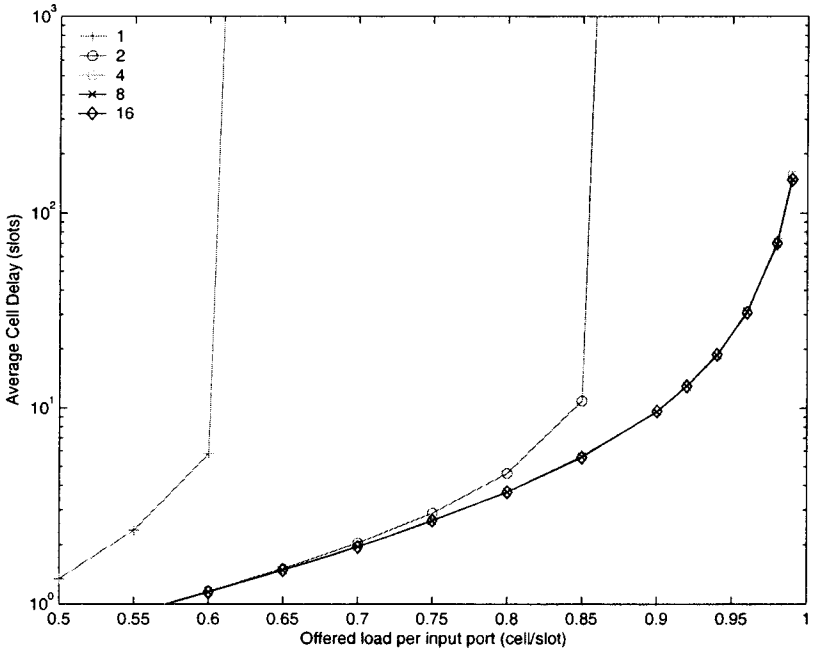


(b) EiSLIP

Figure 5.10: Effects of increasing the number of iterations under uniform Bernoulli i.i.d. traffic.

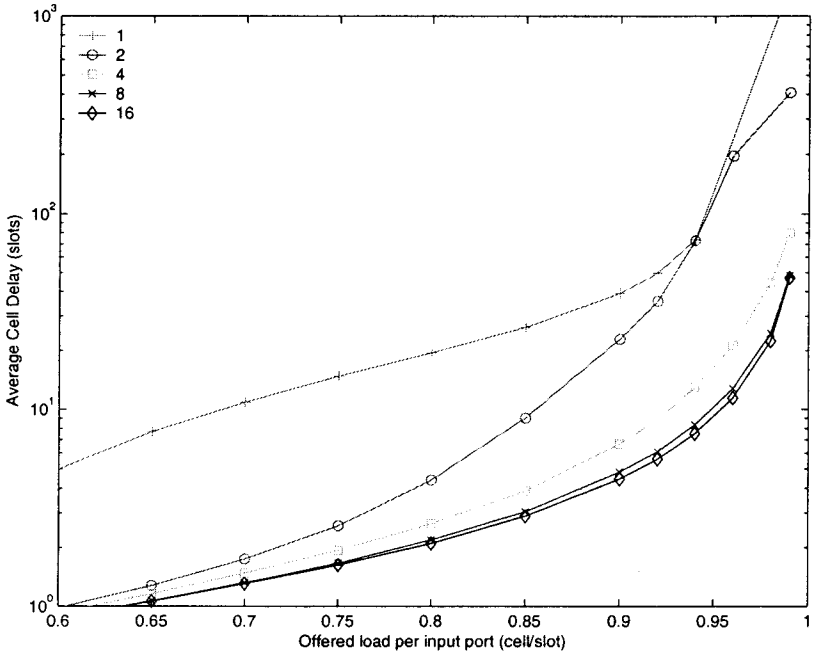


(c) DRR



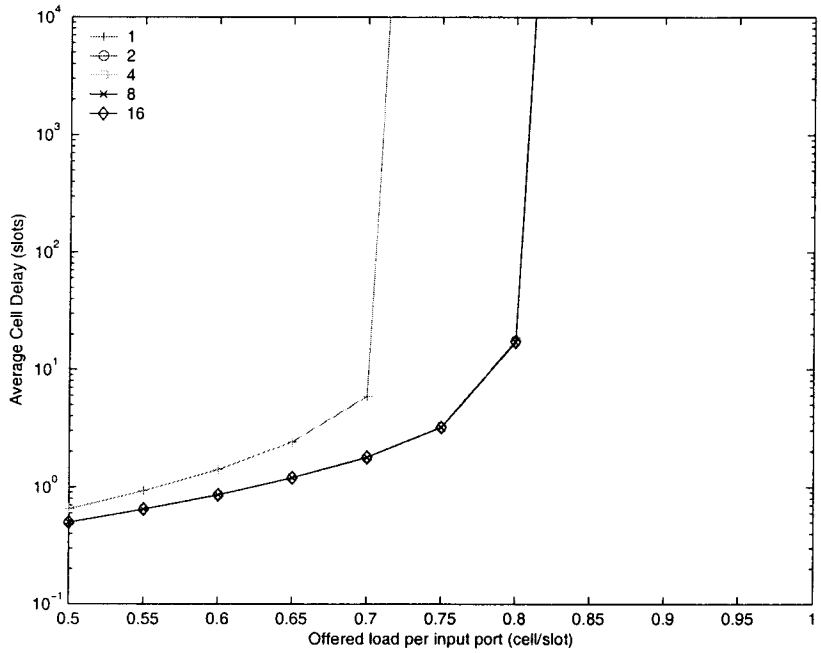
(d) PIM

Figure 5.10: Effects of increasing the number of iterations under uniform Bernoulli i.i.d. traffic (cont).

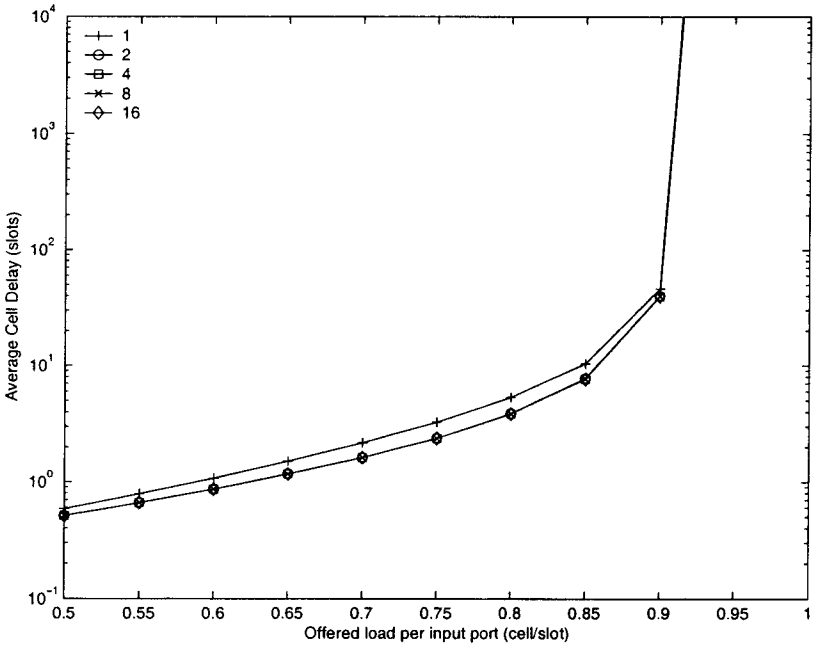


(e) CTR

Figure 5.10: Effects of increasing the number of iterations under uniform Bernoulli i.i.d. traffic (cont).

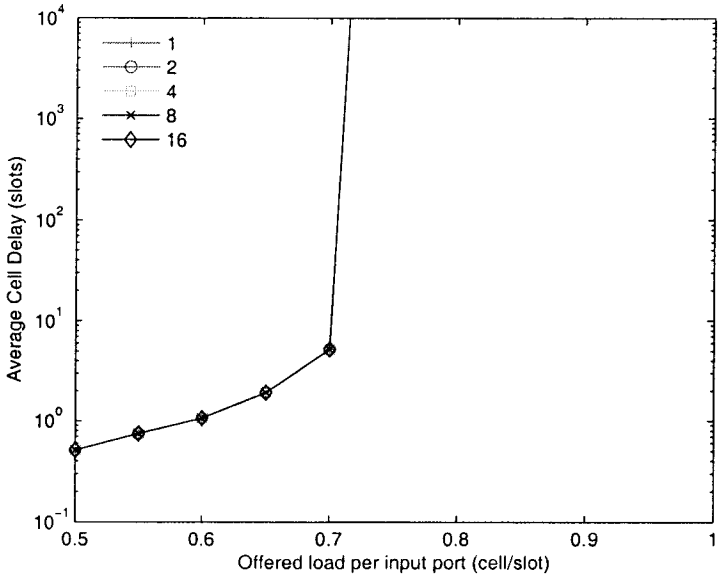


(a) iSLIP

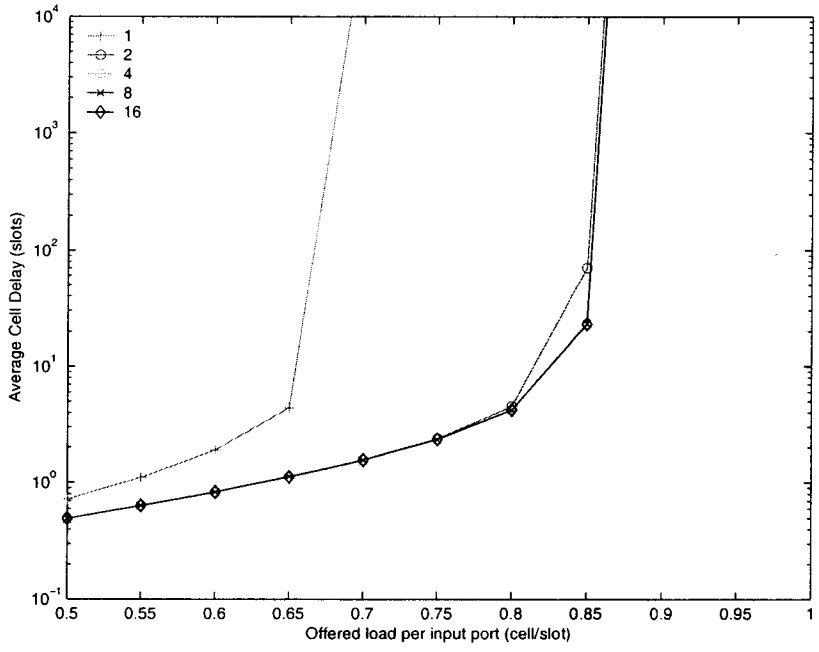


(b) EiSLIP

Figure 5.11: Effects of increasing the number of iterations under log diagonal Bernoulli i.i.d. traffic.

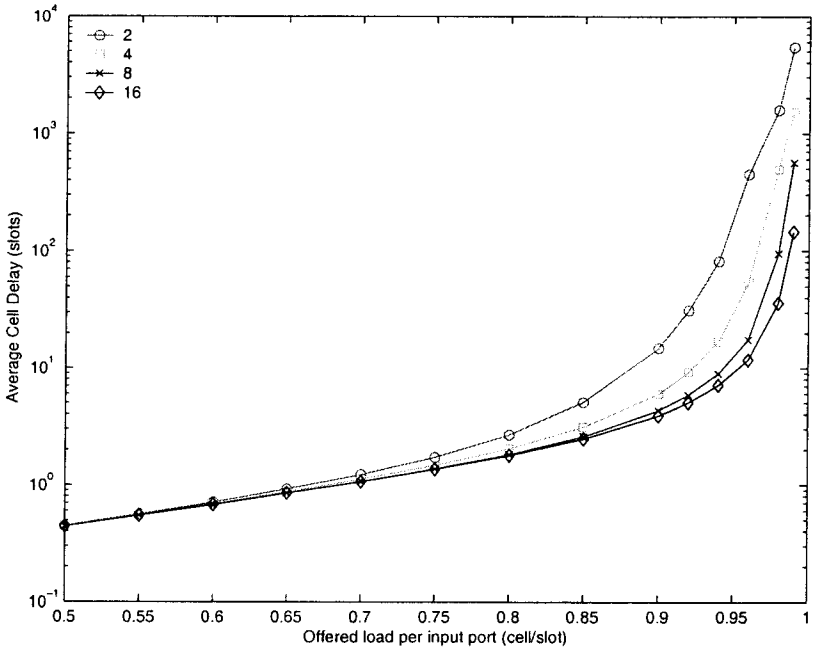


(c) DRR



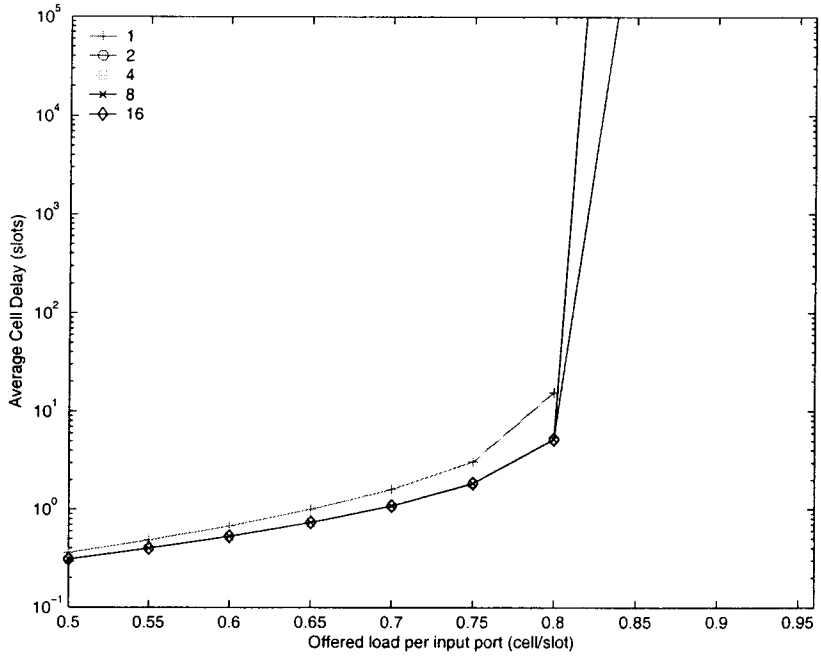
(d) PIM

Figure 5.11: Effects of increasing the number of iterations under log diagonal Bernoulli i.i.d. traffic (cont).

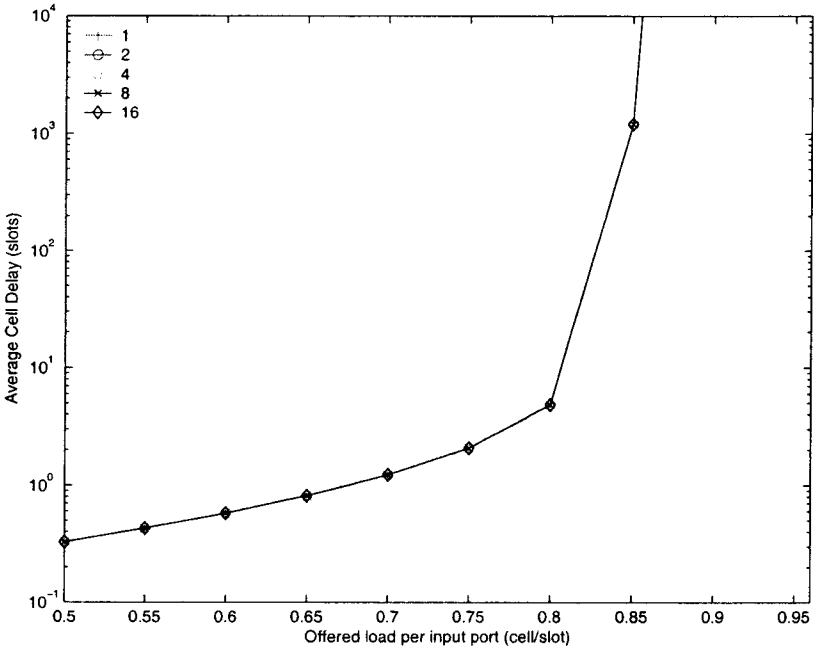


(e) CTR

Figure 5.11: Effects of increasing the number of iterations under log diagonal Bernoulli i.i.d. traffic (cont).

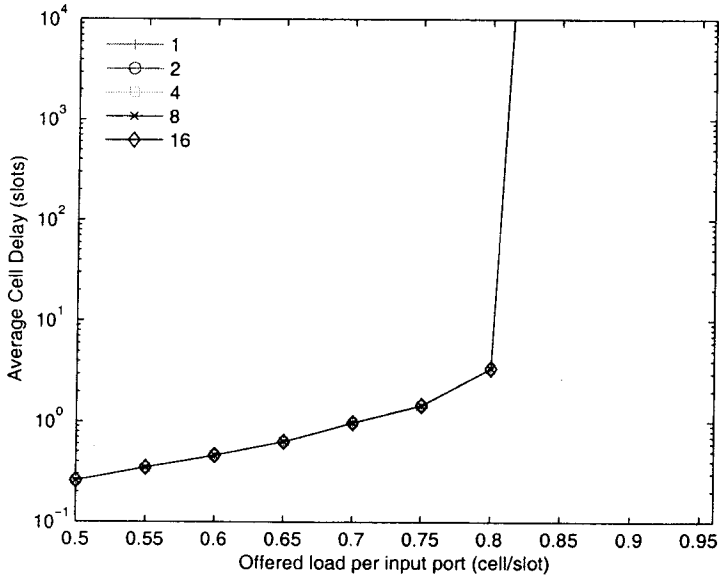


(a) iSLIP

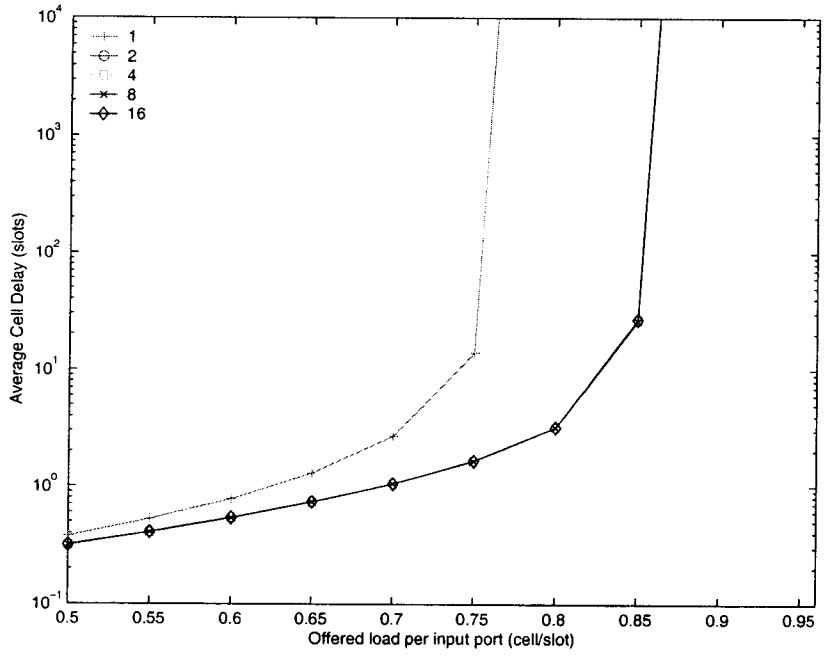


(b) EiSLIP

Figure 5.12: Effects of increasing the number of iterations under diagonal Bernoulli i.i.d. traffic.

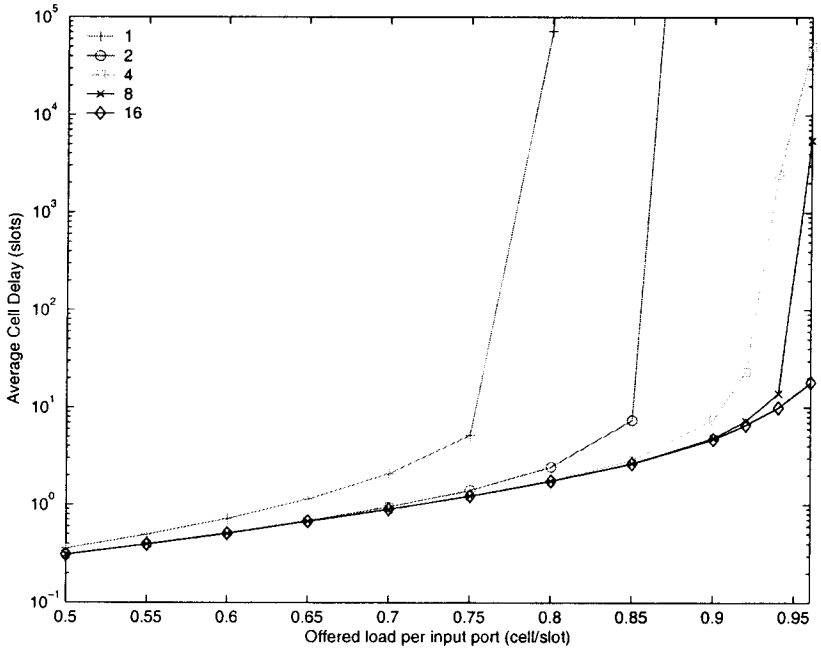


(c) DRR



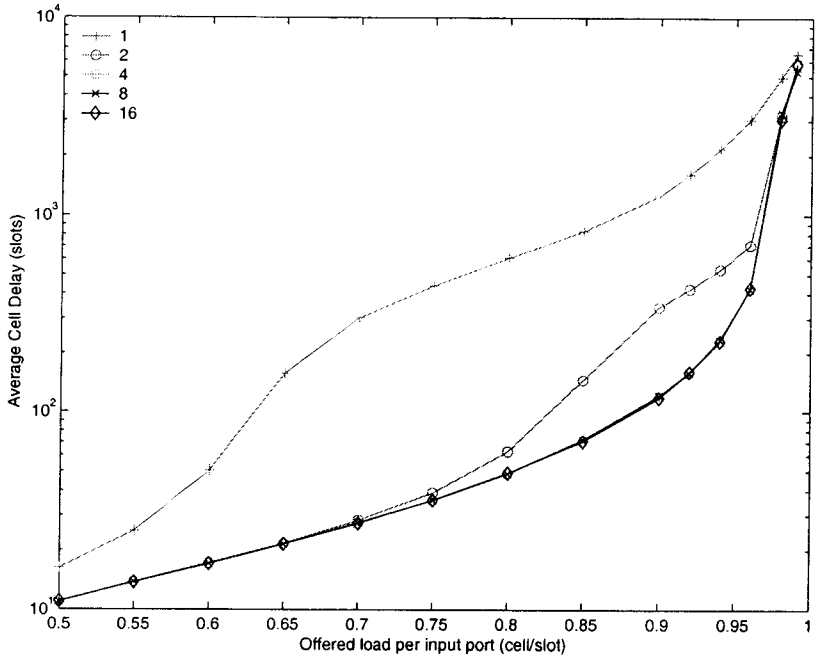
(d) PIM

Figure 5.12: Effects of increasing the number of iterations under diagonal Bernoulli i.i.d. traffic (cont).

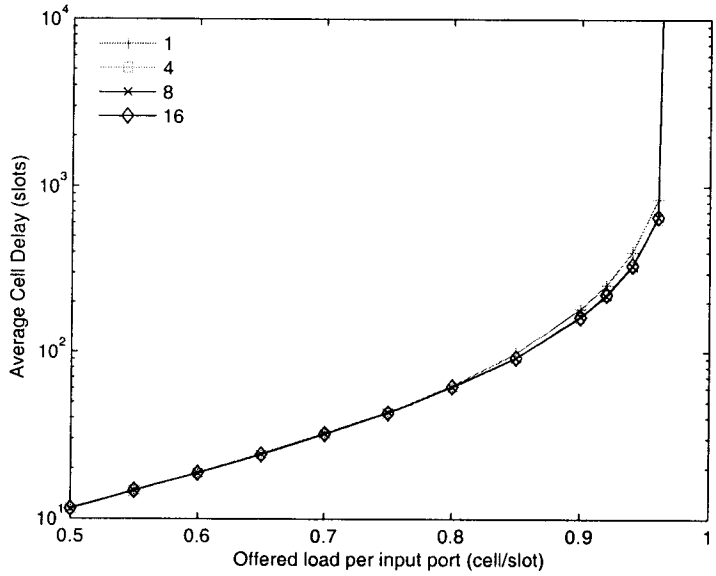


(e) CTR

Figure 5.12: Effects of increasing the number of iterations under diagonal Bernoulli i.i.d. traffic (cont).

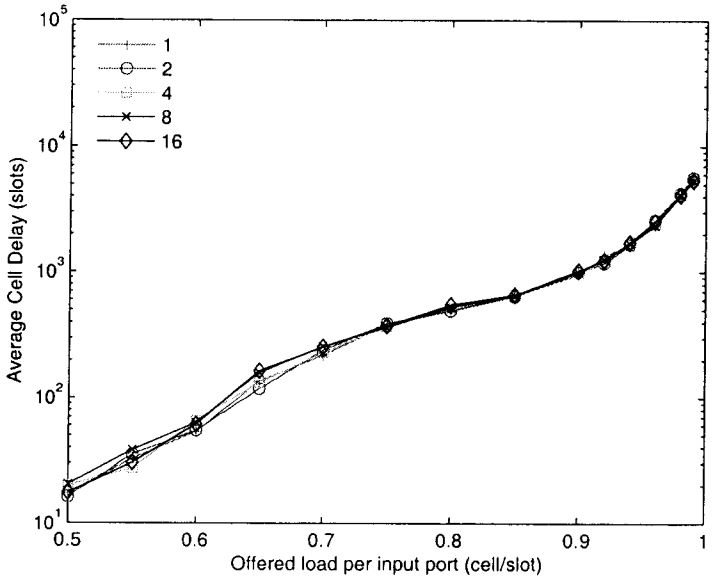


(a) iSLIP

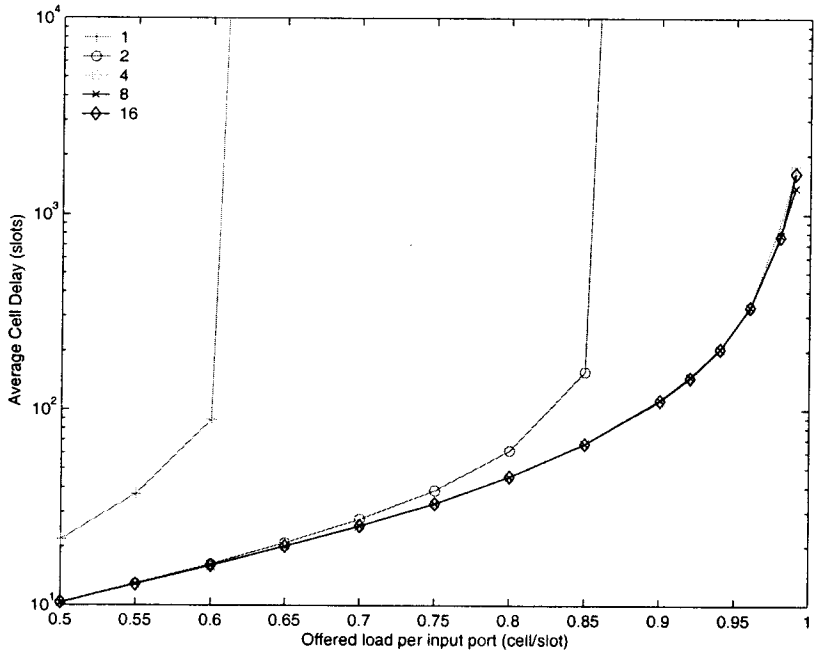


(b) EiSLIP

Figure 5.13: Effects of increasing the number of iterations under uniform bursty traffic with fixed burst size of 16.

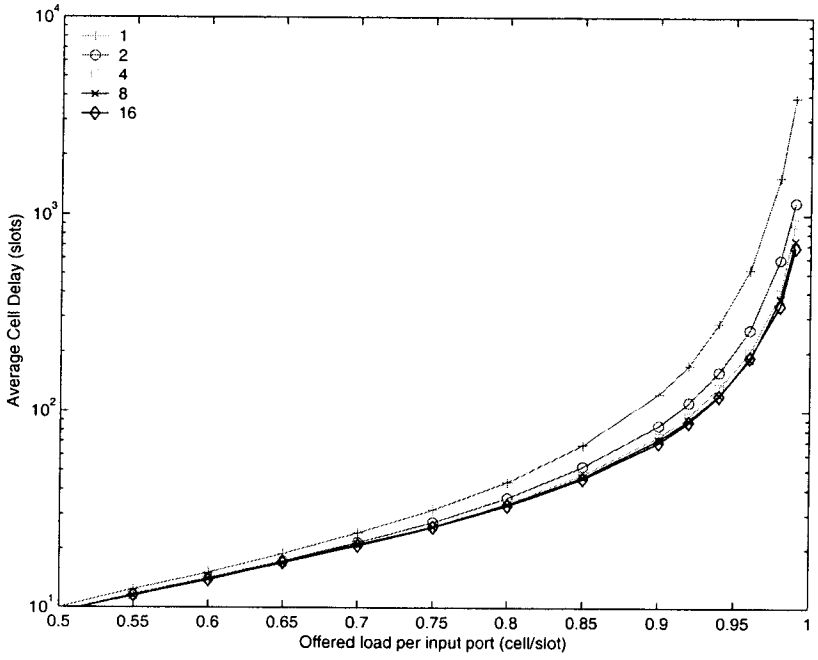


(c) DRR



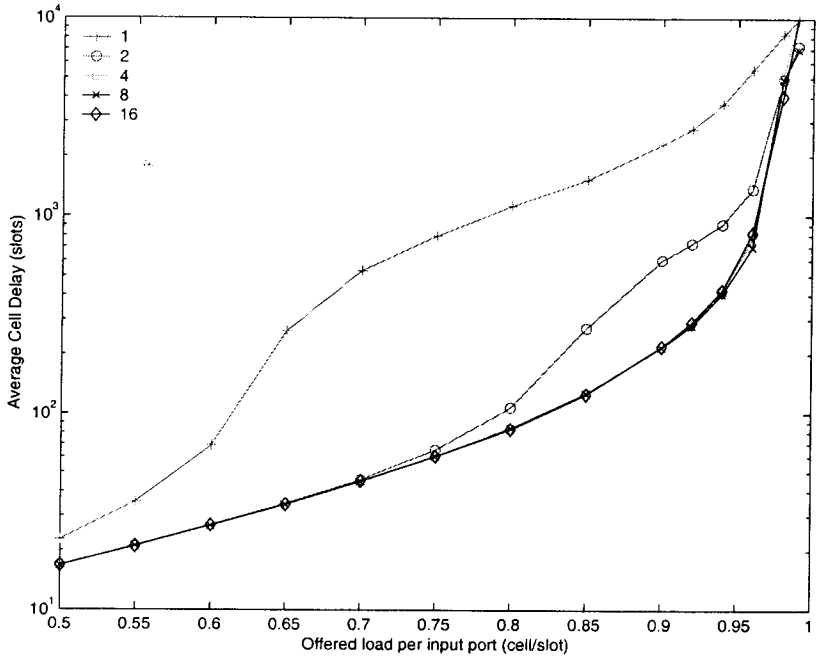
(d) PIM

Figure 5.13: Effects of increasing the number of iterations under uniform bursty traffic with fixed burst size of 16 (cont).

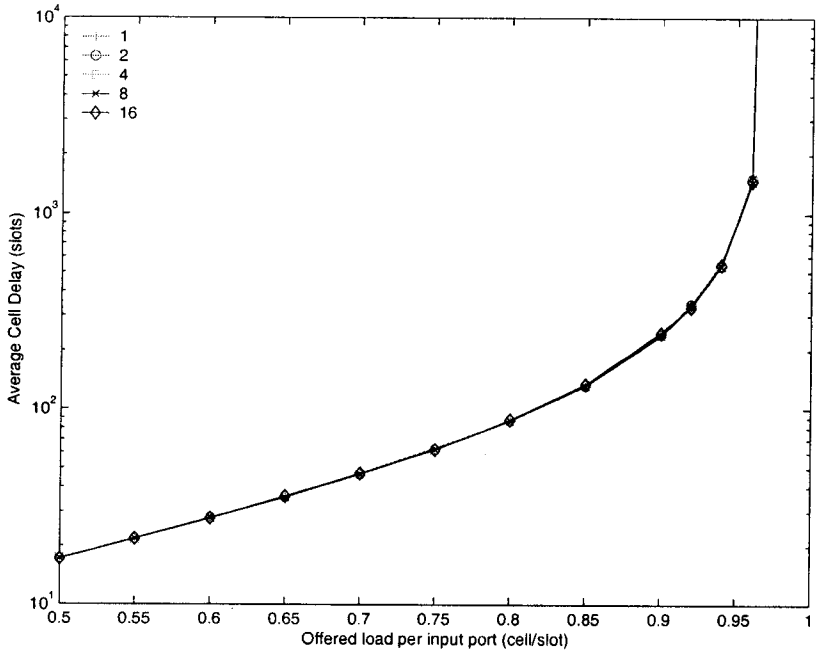


(e) CTR

Figure 5.13: Effects of increasing the number of iterations under uniform bursty traffic with fixed burst size of 16 (cont).

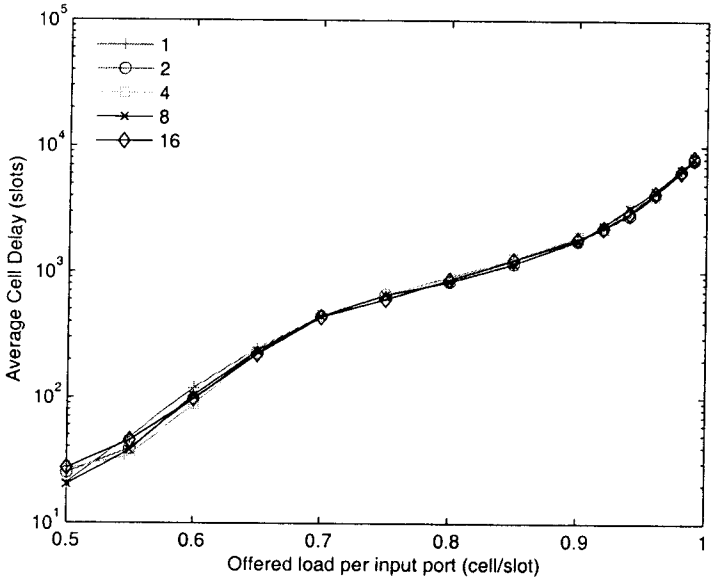


(a) iSLIP

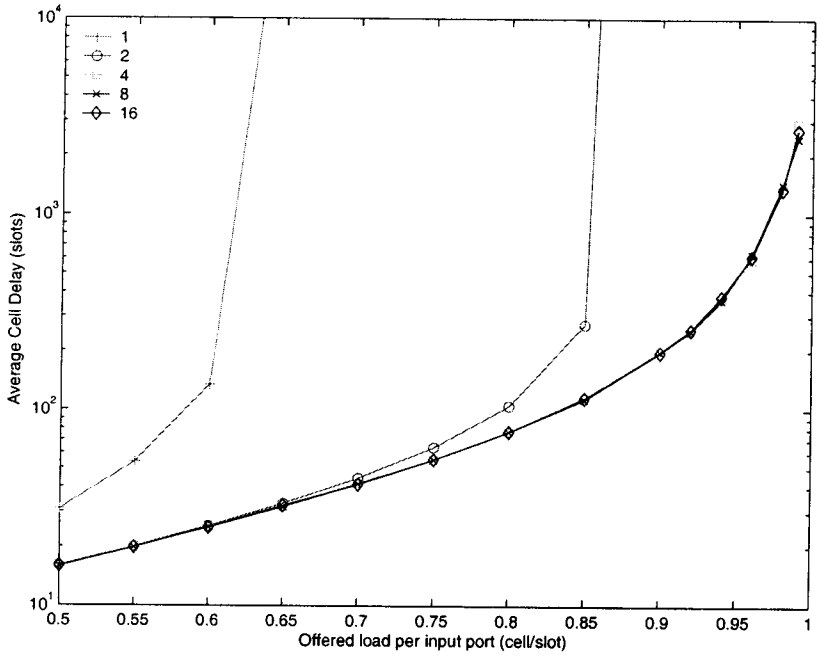


(b) EiSLIP

Figure 5.14: Effects of increasing the number of iterations under under 2-state Markov-modulated arrivals with an average burst size of 16.

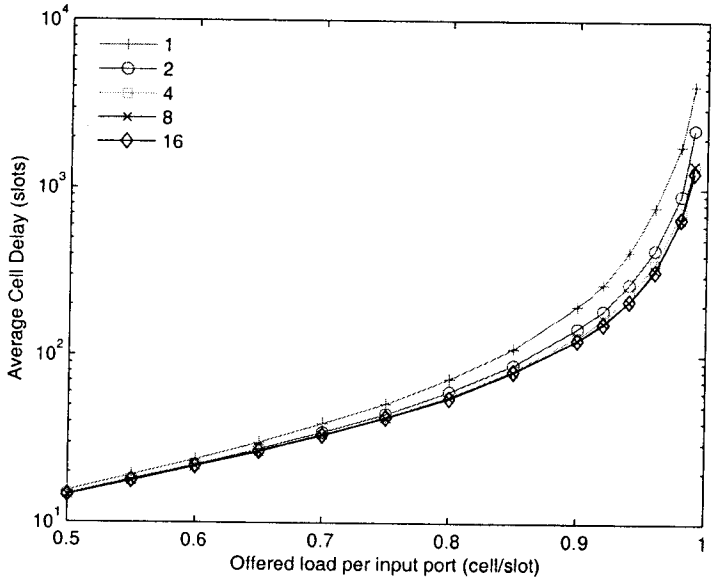


(c) DRR



(d) PIM

Figure 5.14: Effects of increasing the number of iterations under under 2-state Markov-modulated arrivals with an average burst size of 16 (cont).



(e) CTR

Figure 5.14: Effects of increasing the number of iterations under under 2-state Markov-modulated arrivals with an average burst size of 16 (cont).

5.7 Fairness of Cooperative Token-Ring Scheduling

Given that CTR scheduling potentially violates the strict ordering of round-robin arbitration to achieve high throughput, it is expected that it could suffer from a fairness problem for an adversarial traffic pattern. We describe the fairness problem in the current design and then describe several augmenting schemes to the proposed CTR scheduling policy that address fairness.

We have chosen to address the fairness issue separately for several reasons. First, there are various possible solutions with tradeoffs in their implementation complexities that depend on the desired granularity of fairness that we believe should be left to the designer. Second, the solutions to the fairness problem are complementary to the concept of cooperative token-ring scheduling and helps simplify our presentation. Third, and more importantly, we view the decoupling between achieving high throughput and providing fairness as one of our key contributions. It is our view that the tight coupling of a rigid fairness scheme in many scheduling policies, which almost dictates the next schedule, that forces the scheduler to not adapt to the traffic dynamics; thus, causing an overall performance degradation. On the one hand, in a strict round-robin scheduler the *uniform* selection of next matching element tends to dovetail with and *uniform* i.i.d. traffic and the scheduler can provide 100% throughput [LPC01]. However, for non-uniform traffic, strict round-robin scheduling causes performance degradation. On the other hand, exhaustive scheduling policies (e.g. EiSLIP [KC03]) potentially provide better performance than strict round-robin schedulers for bursty traffic, but the scheduler could still get locked into "bad modes" because each arbiter makes its decision obliviously of the state of the other arbiters in the switch; that is, the scheduler does not necessarily adapt to traffic dynamics. Our scheme alleviates this problem by using the cooperative mechanism described earlier.

Although our proposed CTR provides excellent performance for all admissible traffic as previously shown, under *inadmissible* traffic CTR could lead to *starvation* of some queues. An example of starvation behavior is shown in Figure 5.15 for a 2×2 switch. Because all three queues are permanently occupied, the algorithm will always

select the "cross" traffic: input 1 to output 2 and input 2 to output 1 and $VOQ_{1,1}$ will starve.

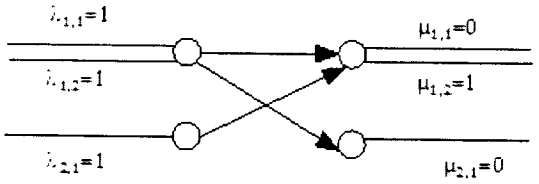


Figure 5.15: Under an inadmissible workload, the CTR scheduler will cause $VOQ_{1,1}$ to starve.

Although in a real-router, decongestion mechanisms (e.g., RED) are applied at the ingress ports to avoid buffer overflow associated with inadmissible traffic, it is still possible to construct an adversarial traffic pattern for CTR that leads to unfairness. There are several mechanisms for providing fairness in a CTR scheduler. One simple scheme is to set a threshold on the number of consecutive time slots for holding the acquired token by an input (e.g., k time slots). This would ensure that each input gets the chance to acquire any token every $k(N-1)$ time slots. Conversely, an input module with a VOQ that has not been served beyond a threshold period of time may send (broadcast) a "prioritized request", which must be honored by the input module that is currently matched to this token. If a higher granularity of fairness is desired, then a credit-based mechanism could be used such that a number of credits are allocated to each input-output pair and each CTR arbiter is allowed to acquire a token only if there are available credits for the corresponding output port. We explore this credit-based scheme in section 5.8 to provide rate guarantees in our proposed CTR scheduler.

5.8 Weighted Cooperative Token-Ring

To provide both rate-guarantees and proportional-bandwidth sharing, we propose a two-level scheduler, weighted cooperative token-ring (WCTR). In WCTR, QoS traffic is scheduled first and then best-effort traffic contend for the remaining bandwidth.

Scheduling QoS in WCTR is performed using frame-based scheduling such that time is divided into frames, and a counter is associated with each input-output pair. The counters are set according to their negotiated bandwidth shares at the beginning of each frame. Queues with positive counters compete with higher priority according to CTR. Then, the remaining queues contend according to CTR for the available bandwidth. During any time slot, an input module can acquire a token for an output-port to send either guaranteed-rate traffic or best effort traffic and a flag is used to indicate the traffic type for which the token is acquired. Scheduling in each level (QoS or best-effort) is performed similar to the original CTR description in Section 5.4: computing the tokens request paths phase followed by a token-selection phase. The main difference is that QoS traffic is prioritized over best-effort traffic. First the token request paths is computed for QoS traffic with the semantics that a module sends a token request only if it has QoS traffic and available credits. Subsequently, if a module that had previously acquired a token for best-effort traffic, and the acquired token is now requested by another unmatched input for QoS traffic, then it must release it. Conversely, during the best-effort scheduling level, a WCTR arbiter would not release a token that was acquired for a QoS traffic if this token is requested by another input.

There are many variations of the presented weighted cooperative token-ring scheduler; for example, it is straightforward to generalize the scheme to multiple priority levels scheduling. In addition, a centralized module could be used allocate credits for best-effort traffic to ensure fairness in the distribution of unreserved bandwidth among the inputs.

5.8.1 Simulation Results for Weighted Cooperative Token-Ring Scheduler

To illustrate the fairness of WCTR in bandwidth allocation, a 4×4 switch was simulated such that each input has four flows, each going to a different output with a different bandwidth reservation. Let $f_k(i,j)$ represent flow k from input port i to output port j. In the simulated switch, $f_1(0,0)$, $f_2(1,0)$, $f_3(2,0)$, and $f_4(3,0)$ have reserved 10, 20, 30, and 40 percent of the bandwidth, respectively, but they always

maintain the same actual arrival rate. Other flows have a load of 5 percent each. This traffic model has been used in [SV95] and [ZB03]. We used a frame size of 1000 slots and measured the throughput per flow at the end of one frame. As shown in Figure 5.16, WCTR is able to provide to each input its allocated rate.

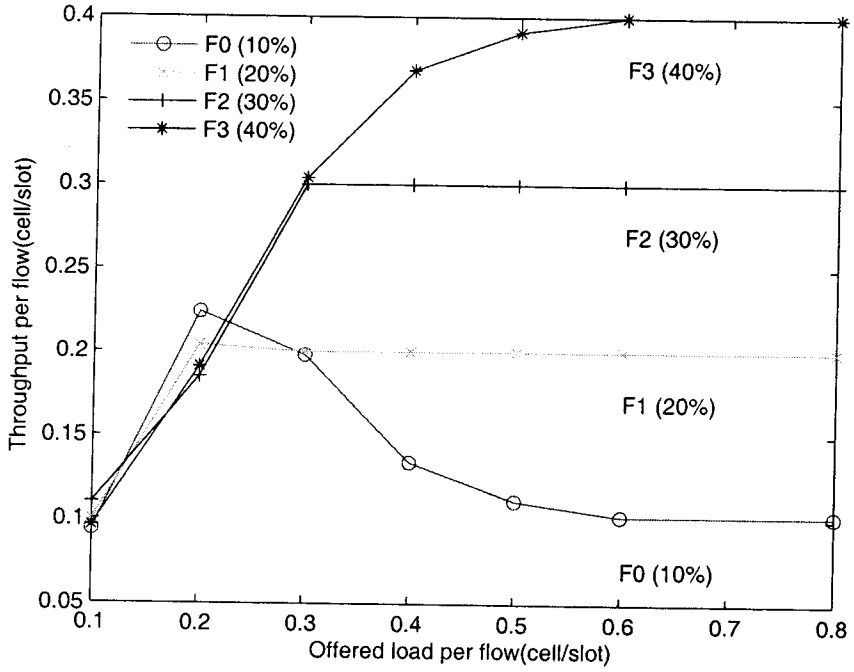


Figure 5.16: Plot of throughput per flow for WCTR at the end of one frame.

5.9 Hardware Implementation of Cooperative Token-Ring

In this section we analyze the hardware complexity of computing the token request paths and prove that its latency is $\Theta(\log N)$ and circuit size per node is $\Theta(N \log N)$.

Recall from Section 5.4.1 that each element in the TRV is given by:

$$TRP_i = R_{|i+1|} + \sum_{j=i+2}^{j=i+N-1} R_{|j|} \prod_{k=i+1}^{k=j-1} \overline{TP}_{|k|}$$

We make the following two observations:

1. Equation (5.1) can not be directly implemented using a parallel prefix circuit [LF80] because each element is computed using all the other elements in the ring in a circular (Modulo arithmetic) fashion.

2. Although it is simple to achieve an optimal latency of $\Theta(\log N)$ using a binary tree for computing each element in Equation (5.1), the circuit size per element (node) would be $\Theta(N \log N)$ per output bit value and consequently the circuit size per node would be $\Theta(N^2 \log N)$, whereas we describe a technique with optimal latency and circuit size per node of $\Theta(N \log N)$.

Rather than providing a specific solution for computing Equation (5.1), we generalize the problem and present a generic circuit for computing all elements in a list such that each element depends on other elements in the list, in a circular fashion, in $\Theta(N \log N)$ circuit size with time complexity $\Theta(\log N)$ as described in Section 5.9.1. A similar hardware circuit for computing circular prefix computations with the same $\Theta(N \log N)$ circuit size and $\Theta(\log N)$ delay bounds was presented in [Szy02], upon which this circuit was based.

5.9.1 Complete Symmetrical Prefix Problem

Let $\otimes$ be a binary associative operation. The complete symmetrical prefix problem is to compute, given $x_1, x_2, \ldots, x_n$, the results $y_1, y_2, \ldots, y_n$, where $y_k = x_{|k+1|} \otimes x_{|k+2|} \otimes \ldots \otimes x_{|k+n-1|}$, for $1 \leq k \leq n$

The problem of computing the token request bit can be solved on a binary tree network in $2D$ steps where D is the depth of the tree. The algorithm consists of essentially two interleaving phases: upward phase, and downward phase. In the upward phase, each internal node computes the product of the entries in the leaves

spanned by the node. In the downward phase, these products are passed downward the tree so the leaf can form the result. We describe the algorithm in more detail in the next sections.

5.9.1.1 Upward phase

During the first step, x_i is input to the ith leaf for $1 \leq i \leq N$. This value is both stored and passed upward to the node's parent. In subsequent steps, internal nodes receiving inputs from children concatenate the inputs and pass the product upward in the tree. After D steps, every node, other than the root, will have computed the product of the inputs to the leaves covered by the node. Figure 5.17 shows the computation performed by each node, and Figure 5.18 shows the actual products computed by each node for an 8-string example.

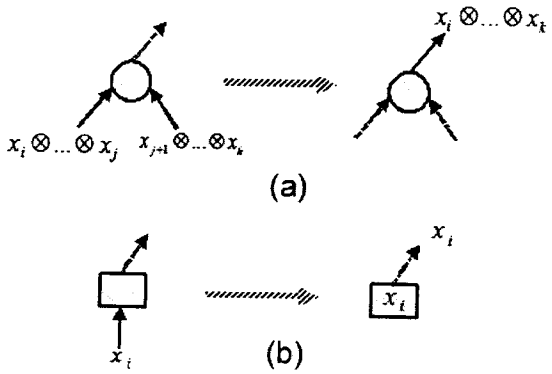


Figure 5.17: Action of an internal node (a) and leaf (b) during the upward phase. Inputs to the internal nodes are concatenated, and then passed upwards. Inputs to leaves are stored and passed upwards.

As each nonleaf node receives its input from below, it swaps the values computed by the left and right child. That is, it passes the value computed by the left child to the right child and conversely. Each node then stores the new value it receives from its parent as shown in Figure 5.19.

For example, Figure 5.20 shows the node values after swapping for the previous 8-string example.

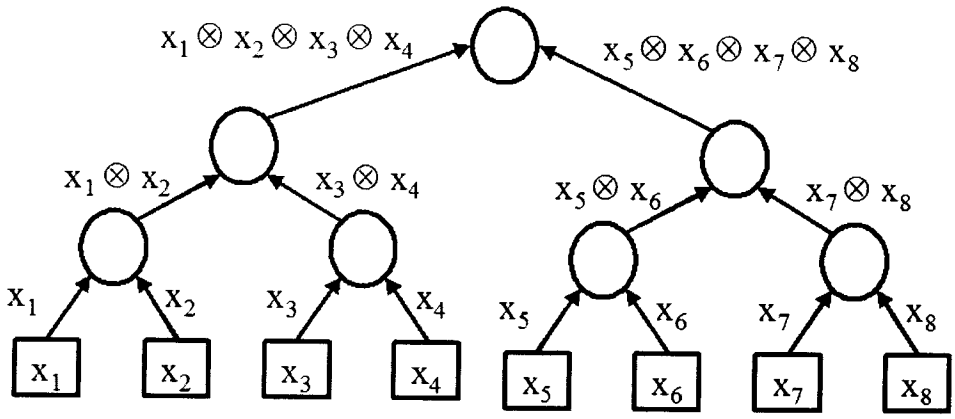


Figure 5.18: Concatenation performed by each node in the parallel prefix algorithm for an 8-element string. Each node computes the product of the inputs that it spans.

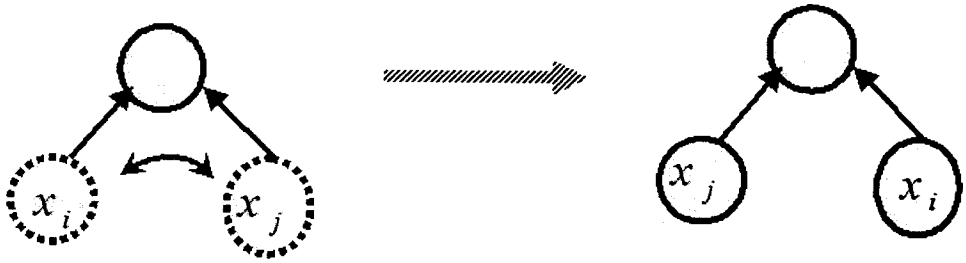


Figure 5.19: Swapping the node values during the upward propagation phase.

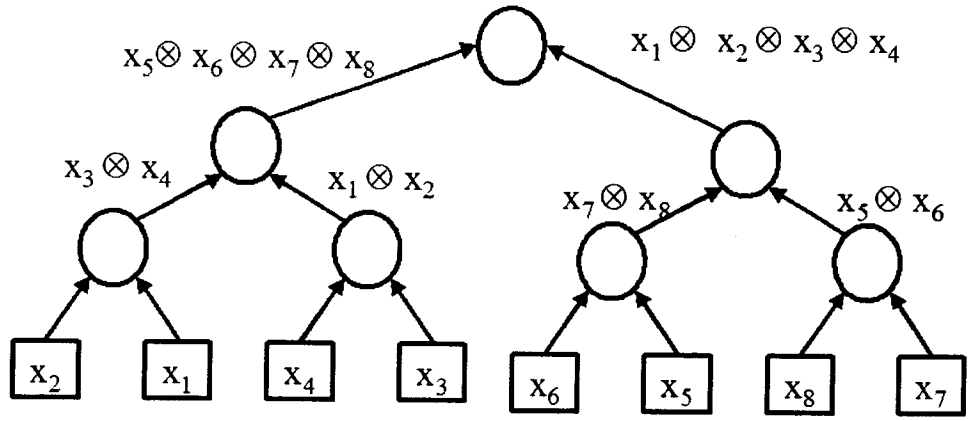


Figure 5.20: The node values after swapping the left and right child for the 8-string used in Figure 5.18

5.9.1.2 Downward Phase

During the downward phase each node receives the node value of its parent and computes the result as depicted in Figure 5.21. The operation is performed both for leaf and nonleaf nodes.

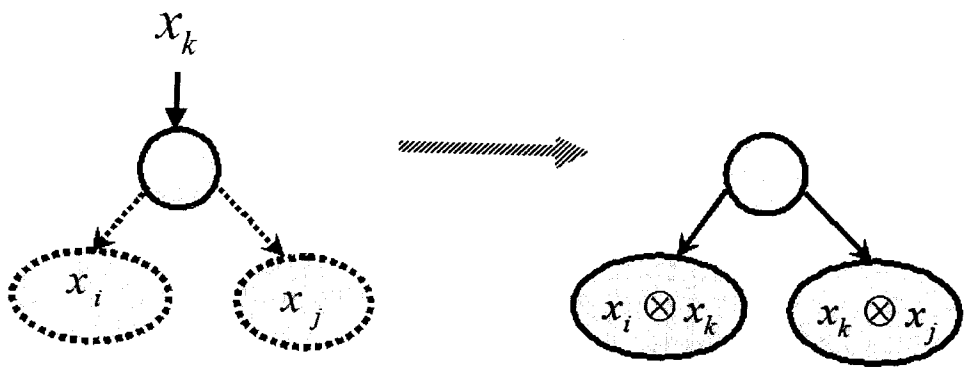


Figure 5.21: Operation of nodes (leaf and nonleaf nodes) during the downward phase.

In total, the algorithm takes 2D steps, where D is the depth of the tree and the circuit size is determined by the number of nodes in the tree, which is $\Theta(N \log N)$ assuming each node has a fixed size.

5.9.2 Computing the Token Request Vector as a Complete Symmetrical Prefix Problem

Computing the token request vector can be modeled as the complete symmetrical prefix problem by keeping track of whether each module stage *stops* a request, *propagates* a request or *generates* a request. Specifically, let the state at node i be:

stop(s) $\equiv$ if the token is currently at node i.

propagate(p) $\equiv$ if the token is not currently at node i and the node does not have a request for the token.

generate(g) $\equiv$ if module i has a request for the token.

Let x_i denote the s, p, or g value of ith node, and let $|k| = (k \ mod \ N)$.
Next, let $TRP_i = x_{|i+1|} \otimes x_{|i+2|} \otimes x_{|i+3|} \cdots \otimes x_{|i+N-1|}$ for $1 \leq i \leq N$ where $\otimes$ is the binary associative operator defined by the table in Table 5.1.

$\otimes$	S	P	G
S	S	S	S
P	S	P	G
G	G	G	G

Table 5.1: Multiplication table for the operator defined for computing the token request path. For example, $S \otimes G = S$

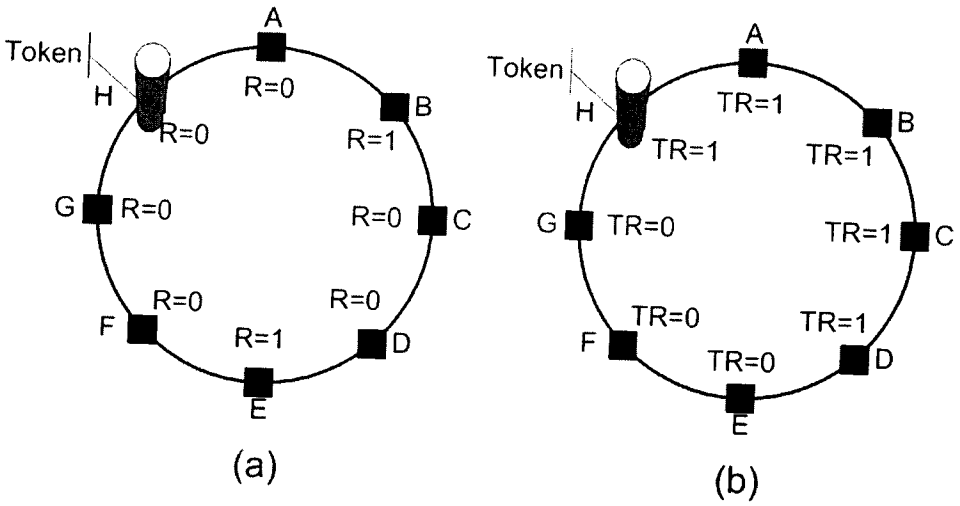


Figure 5.22: (a) The request bit at each node. (b) The corresponding value of TRV at each node.

For example, the signal for the input modules used in Figure 5.22 (a) are shown in Table 5.2. The token request bit at module x_3 is given by:

$$x_3 = x_4 \otimes x_5 \otimes x_6 \otimes x_7 \otimes x_8 \otimes x_1 \otimes x_2$$
$$x_3 = p \otimes g \otimes p \otimes p \otimes s \otimes p \otimes g$$
$$x_3 = g$$

Module	Signal
$x_1 \equiv x_A$	P
$x_2 \equiv x_B$	G
$x_3 \equiv x_C$	P
$x_4 \equiv x_D$	P
$x_5 \equiv x_E$	G
$x_6 \equiv x_F$	P
$x_7 \equiv x_G$	P
$x_8 \equiv x_H$	S

Table 5.2: The signal states for the modules in Figure 5.22

Figure 5.23 shows the circuit for computing the token request path for a for a ring with four inputs; this circuit was generated using Synopsys [2] design compiler, with optimization setting for high speed. The circuit computes one column of the **TP** matrix.

5.10 Examples of the Cooperative Token-Ring Scheduling Policy

In this section we provide detailed step-by-step examples of CTR implemented using the ΠRGA paradigm described in Section 5.5.

5.10.1 Notation

A matrix notation is used such that rows correspond to inputs and columns correspond to outputs. All matrix elements are binary values.

1. **Request** The request matrix in the request-grant-accept arbitration phase. This matrix is different from **R** matrix (Definition 25.)– a matched input can send a request to an unmatched output for which the corresponding TRV element is zero (i.e., a token that is not currently requested by other unmatched inputs), whereas in the **R** matrix only unmatched input can send requests.

[2]Synopsys is a trademark of Synopsys, Inc.

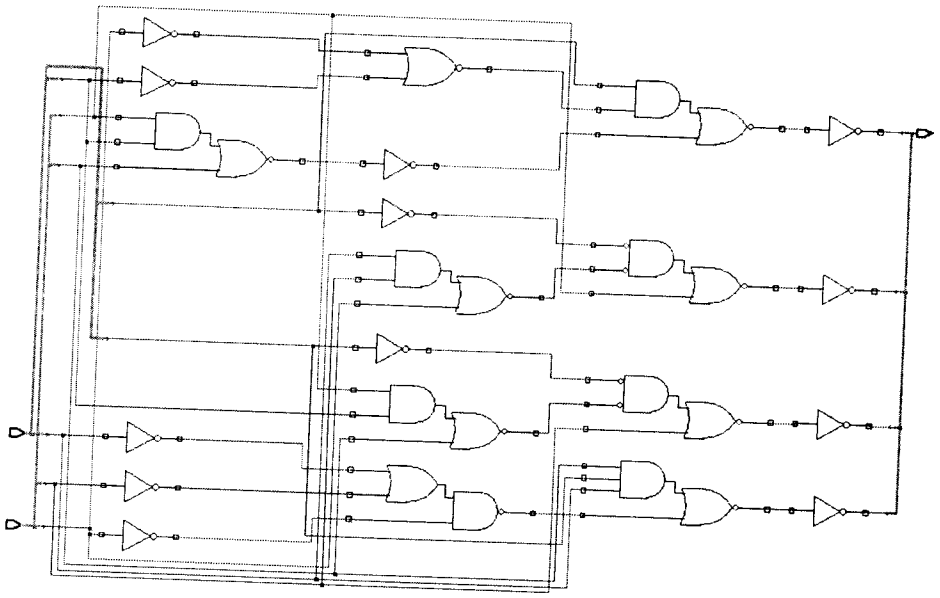


Figure 5.23: Circuit for computing the token request path in a ring with four nodes. The circuit uses approximately 30 standard cells to realize the binary tree structure shown in Figs. 5.18 and 5.21, with 4 input ports and 3 binary nodes. Therefore, each node in the binary tree consumes approximately 10 standard cells.

$Request_{i,j}$ is set to 1 if there is a request from input i for output j, and is set to zero otherwise.

2. **Grant** The grant matrix. $Grant_{i,j}$ is set to 1 if there is a grant from output j to input i.

3. **Accept** $Accept_{i,j}$ is set to 1 if there is an accept from input i to output j, and is set to zero otherwise.

5.10.2 First Example

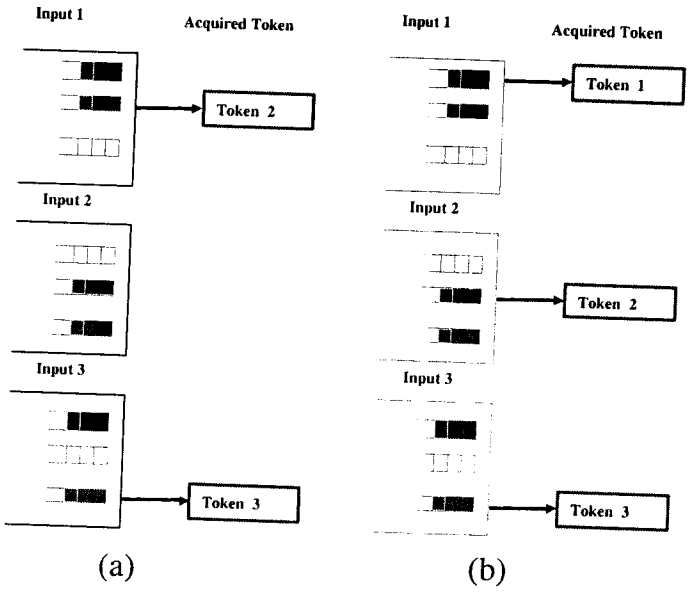


Figure 5.24: First Example of CTR scheduling policy (a) Initial State (b) Final State

Assume the initial switch configuration shown in Figure 5.24 (a).

5.10.2.1 Initial State

$$\mathbf{VOQ} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5.10.2.2 First Iteration: Computing the Tokens Request Paths Phase

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} \mathbf{TRP} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5.10.2.3 First Iteration: Request-Grant-Accept Phase

$$\mathbf{Request} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \mathbf{Grant} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{Accept} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

At the end of the first iteration, input 1 acquires the token for output 1 and releases the token for output 2 (token 2). Consequently, the state of the round-robin arbiters and the acquired tokens state is given by the following matrices.

$$\mathbf{TP} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5.10.2.4 Second Iteration:Computing the Tokens Request Paths Phase

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{TRP} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5.10.2.5 Second Iteration: Request-Grant-Accept Phase

$$\textbf{Request} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \textbf{Grant} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \textbf{Accept} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

At the end of the second iteration, input 2 acquires the token for output 2 and all the inputs are matched.

5.10.2.6 Final State

$$\textbf{TP} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \textbf{M} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The final state is shown in Figure 5.24 (b).

5.10.3 Second Example

This example shows how the CTR arbiters use the token request vector to select among multiple grant signals to improve the throughput.

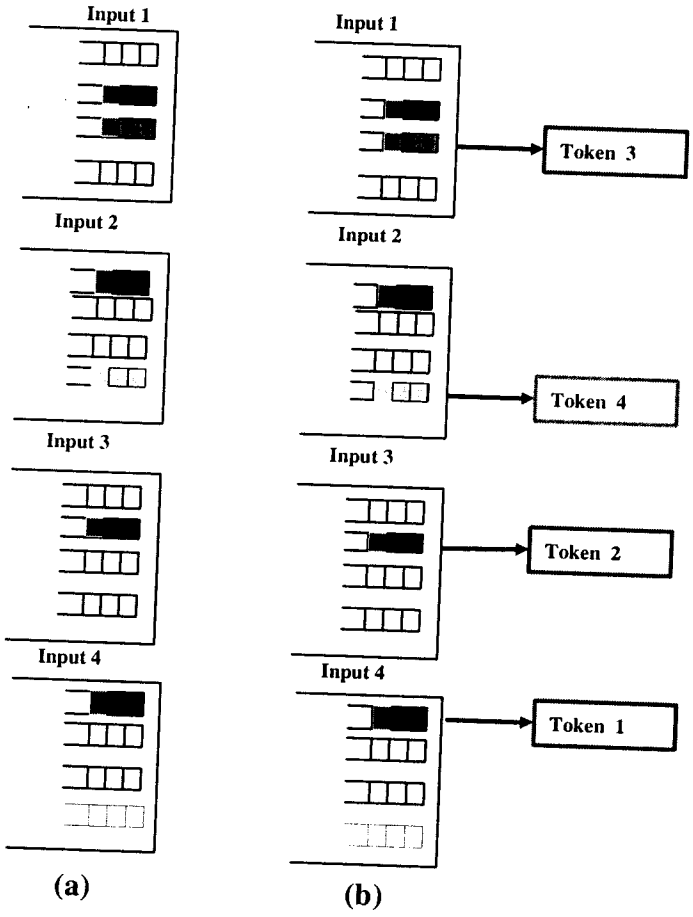


Figure 5.25: Second Example of CTR scheduling policy (a) Initial State (b) Final State

5.10.3.1 Initial State

$$\mathbf{VOQ} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The initial state is shown in Figure 5.25(a).

5.10.3.2 First Iteration: Computing the Tokens Request Paths Phase

$$\mathbf{R} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \mathbf{TRP} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

5.10.3.3 First Iteration: Request-Grant-Accept Phase

$$\mathbf{Request} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{Grant} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{Accept} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Input 1 receives two grants for outputs 2 and 3. Because the TRV bit for output 2 is set to 1 it chooses to send an accept to output 3, which is not requested by other inputs. Similarly, input 2 receives two grant signals for outputs 1 and 4, and chooses to accept the grant from output 4 because it is not requested by other inputs. Consequently, outputs 1 and 2 are left unmatched and will be matched in the second

iteration. The state of the output round-robin arbiter and the matched state is:

$$\mathbf{TP} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

5.10.3.4 Second Iteration:Computing the Tokens Request Paths Phase

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{TRP} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

5.10.3.5 Second Iteration: Request-Grant-Accept Phase

$$\mathbf{Request} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{Grant} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{Accept} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

At the end of the second iteration, input 1 is matched to output 3, input 2 is matched to output 4, input 3 is matched to output 2, and input 4 is matched to output 1.

$$\mathbf{TP} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

The final state is shown in Figure 5.25(b).

5.10.4 Third Example

This example shows how an input arbiter swaps an acquired token that is requested by another unmatched input arbiter for an unrequested token.

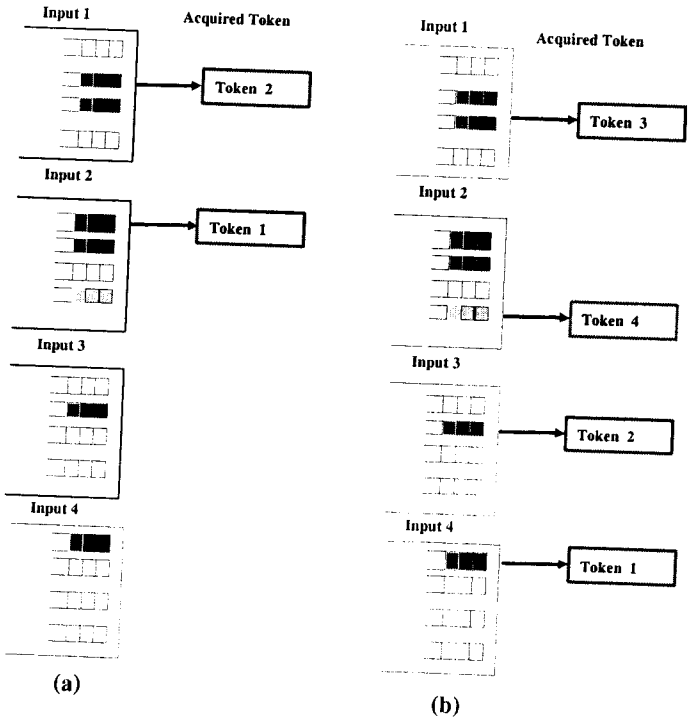


Figure 5.26: Example 3 of CTR scheduling policy (a) Initial State (b) Final State

5.10.4.1 Initial State

$$\mathbf{VOQ} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The initial state is shown in Figure 5.26(a).

5.10.4.2 First Iteration: Computing the Tokens Request Paths Phase

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{TRP} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

5.10.4.3 First Iteration: Request-Grant-Accept Phase

$$\mathbf{Request} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{Grant} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{Accept} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

At the end of the first iteration, input 1 acquires the token for output 3 and releases the token for output 2. Similarly, input 2 releases the token for output 1 and acquires the token for output 4 and releases the token for output 1. Consequently, the state of the output round-robin arbiter and the matched state is shown next.

$$\mathbf{TP} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

5.10.4.4 Second Iteration:Computing the Tokens Request Paths Phase

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{TRP} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

5.10.4.5 Second Iteration: Request-Grant-Accept Phase

$$\textbf{Request} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \textbf{Grant} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \textbf{Accept} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

At the end of the second iteration, input 3 acquires the token for output 2 and input 4 acquires the token for output 1.

$$\textbf{TP} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \textbf{M} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

The final state is shown in Figure 5.26(b).

5.10.5 Fourth Example

This example shows how swapping tokens could break a cyclic dependency (a module requests a token that is acquired by another module which in turn would release if it acquires another token that is currently acquired by another module) between the input modules and leads to improving the throughput.

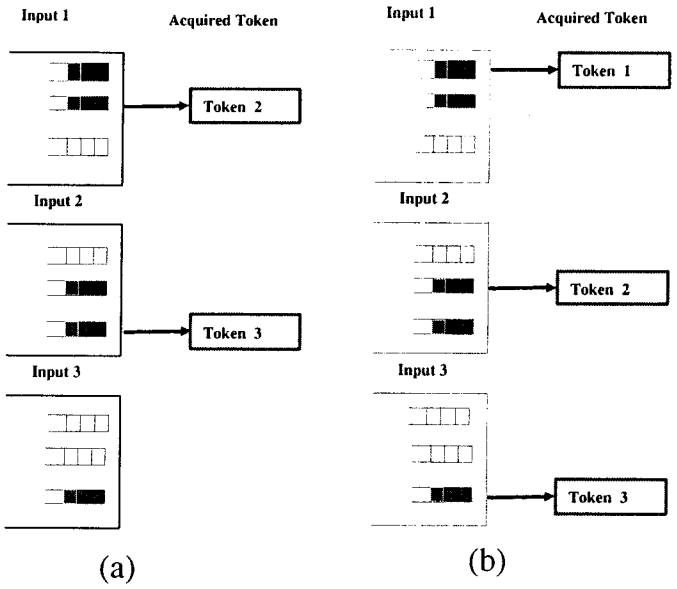


Figure 5.27: Example 3 of CTR scheduling policy (a) Initial State (b) Final State

5.10.5.1 Initial State

$$\mathbf{VOQ} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

The initial state is shown in Figure 5.27(a).

5.10.5.2 First Iteration: Computing the Tokens Request Paths Phase

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \mathbf{TRP} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

5.10.5.3 First Iteration: Request-Grant-Accept Phase

$$\mathbf{Request} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{Grant} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{Accept} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

At the end of the first iteration, input 1 releases token 2 and acquires token 1.

$$\mathbf{TP} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

5.10.5.4 Second Iteration:Computing the Tokens Request Paths Phase

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{TP} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{TRP} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

5.10.5.5 Second Iteration: Request-Grant-Accept Phase

$$\mathbf{Request} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{Grant} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{Accept} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

At the end of the second iteration, input 2 releases token 3 and acquires token 2.

$$\mathbf{TP} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

5.10.5.6 Third Iteration:Computing the Tokens Request Paths Phase

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{TRP} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

5.10.5.7 Third Iteration: Request-Grant-Accept Phase

$$\mathbf{Request} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{Grant} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{Accept} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

At the end of the third iteration, input 3 acquires the token for output 3 and the all the inputs are matched. The final state is shown in Figure 5.27(b).

$$\mathbf{TP} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{M} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5.11 CONCLUSION

We proposed cooperative token-ring scheduling, a novel scheduling paradigm that provides significant improvement over existing schedulers with comparable complexity. Our scheduling paradigm adapts to dynamically varying traffic, provides high throughput, and is easily implemented in hardware. We proposed WCTR to provide rate guarantees in IQ switches and proportional bandwidth sharing. Finally, we note

that although CTR was presented in the context of IQ switches, it is potentially applicable to several other systems including SONET all-optical circuit-switches, which schedules cells in circuit-based frames by using delay lines and star-based WDM broadcast-and-select optical system with tunable transmitters and fixed receivers. Generally, the proposed CTR scheme can be applied to solve any resource allocation problem with a set of nodes competing for exclusive access to a set of shared resources.

Chapter 6

Conclusions

In this chapter we summarize the main contributions of this dissertation and discuss future research directions.

6.1 Summary of Contributions

In Chapter 4 we introduced a theoretical framework for evaluating the performance of IQ switches, and proposed the "lag" concept as a performance metric that measures the difference between a packet's departure time in an IQ switch over that provided by an OQ switch. By tracking the behaviour of an OQ switch, an IQ switch resolves input and output contention fairly, eliminates any starvation of inputs, and approximates the behaviour of an OQ switch as close as possible. We presented MWL, a scheduling policy based on maximum weighted matching, that uses lag values for its weights. We proved that MWL provides 100% throughput under Bernoulli i.i.d. traffic and that the per packet lag is bounded. A bound on the mean lag value per packet was derived using a Lyapunov function technique. The MWL scheduling policy has a run time complexity of $\Theta(N^3 \log N)$ on a sequential model, which is prohibitively expensive to implement in practice. Consequently, we proposed a simple heuristic tracking scheduling policy, iLag, based on maximal matching. The performance of MWL and iLag was evaluated by simulation and compared to other scheduling policies of comparable complexity.

In Chapter 5 we presented the cooperative token-ring (CTR) scheduling paradigm,

for Internet routers with IQ switches that provides significant performance improvement over existing scheduling schemes with comparable complexity. We showed that by using a simple cooperative mechanism between the otherwise non-cooperative token-rings (arbiters) the performance can be significantly improved and the scheduler is able to dynamically adapt to any non-uniform traffic pattern. In addition, the cooperative mechanism is simple to to implement in hardware. To provide adequate support for rate guarantees in IQ switches, we proposed Weighted Cooperative Token-Ring (WCTR) scheduling policy, a simple hierarchical scheduling mechanism that provides both rate guarantees and proportional bandwidth sharing. The performance of WCTR was evaluated by simulation.

We analyzed the hardware complexity introduced by the proposed cooperative mechanism and provided an optimal hardware implementation for an $N \times N$ switch implementing a CTR scheduler. We proved that the hardware complexity of the cooperative mechanism is $\Theta(\log N)$ time and $\Theta(N \log N)$ circuit size, per port.

In Chapter 3 we addressed the problem of short-term fairness in QoS scheduling for IQ switches. We presented a scheduling algorithm for Internet routers with IQ switches based on credit-based fair queueing that provides better short-term fairness in QoS scheduling than existing solutions with comparable complexity. A flow-based iterative credit-based fair scheduler (iCBFS) was proposed that provides fair bandwidth distribution among flows at a fine granularity than existing schemes with comparable complexity. To reduce the implementation complexity of iCBFS, we presented a port-based version of iCBFS that is tailored towards high-speed hardware implementation.

6.2 Future Work

Future work will focus on multi-stage switch fabrics, variable- length packet scheduling, and multicast scheduling as described next.

As the demand for high capacity switch increases, future switches will use more ports and higher data rate per port than existing switches and consequently require scalable schedulers that support hundreds or thousands of ports. This scalability will

likely require using a multi-stage switch fabric. Future work will address scheduling in multi-stage switch fabrics to achieve fairness and high throughput.

This work assumed fixed-length cells such that incoming variable-length IP packets are segmented into fixed-length cells at the input and are reassembled at the output of the switch. Using fixed-length cells simplifies the switch's scheduler at the overhead cost of the segmentation and reassembly process. Future work will investigate scheduling policies for variable-length packets.

A growing portion of IP traffic is multicast as point-to-multipoint and multipoint-to-multipoint applications such as audio and video conferencing are being used. A trivial way to schedule multicast traffic is to duplicate each multicast packet such that the scheduler still operates on unicast packets. It is interesting to consider more memory efficient multicast scheduling policies that do not use this approach. The issue of scheduling QoS multicast traffic is also an interesting area of future work.

Bibliography

[ADH98] Armen S. Asratian, Tristan M. J. Denley, and Roland Haggkvist. *Bipartite graphs and their applications.* Cambridge University Press, New York, NY, USA, 1998.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley, 1974. AHO a 74:1 1.Ex.

[AHU82] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms.* Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley, 1982. AHO a 82:1 P-Ex.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[AOST93] T.E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker. High speed switch scheduling for local area networks. *ACM Trans. Computer Systerms*, 11(4):319–352, November 1993.

[Avi04] Avici systems, Inc., Billercia, MA., 2004.

[AZ03] M. Andrews and L. Zhang. Achieving stability in networks of input-queued switches. *IEEE/ACM Trans. Networking*, 11(5):848 – 857, October 2003.

[BDEA04] H. Balakrishnan, S. Devadas, D. Ehlert, and Arvind. Rate guarantees and overload protection in input-queued switches. In *IEEE INFOCOM*, pages 2185–2195, March 2004.

[BTC01] B. Bensaou, D. H. K. Tsang, and K. T. Chan. Credit-based fair queueing (cbfq): a simple service-scheduling algorithm for packet-switched networks. *IEEE/ACM Trans. Networking*, 9(5):591–604, October 2001.

[CB97] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Networking*, 5(6):835–846, December 1997.

[CCH01] C. S. Chang, W. J. Chen, and H. Y. Huang. Birkhoff-von neumann input buffered crossbar switches for guaranteed-rate services. *IEEE Trans. Commun.*, 49(7):1145–1147, July 2001.

[CGMP99] S. T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input/output-queued switch. *IEEE J. Select. Areas Commun.*, 17(6):1030–39, June 1999.

[Cha00] H. J. Chao. Saturn: a terabit packet switch using dual dound-robin. *IEEE Commun. Mag.*, 38(12):78–84, December 2000.

[Cis04] Cisco 12000 series-internet routers, 2004.

[CJGL99] H. J. Chao, Y. R. Jenq, X. Guo, and C. H. Lam. Design of packet fair queueing schedulers using a ram-based searching engine. *IEEE J. Select. Areas Commun.*, 17(6):1105–1126, June 1999.

[CLJ02] C. S. Chang, D. S. Lee, and Y. S. Jou. Load balanced birkhoff-von neumann switches, part i: one-stage buffering. *Computer Communications*, 25:611–622, 2002.

[CLR90] Thomas H. Cormen, E. Leiserson, Charles, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990. COR th 01:1 1.Ex.

[CP98] J. H. Chao and J. S. Park. Centralized contention resolution schemes for a large-capacity optical atm switch. In *Proc. IEEE ATM Workshop*, May 1998.

[Cru91] R. L. Cruz. A calculus for network delay, part II: network analysis. *IEEE Trans. Inform. Theory*, 37:132–141, 1991.

[Din70] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[DP00] J. G. Dai and B. Prabhakar. The throughput of data switches with and without speedup. In *IEEE INFOCOM*, pages 556–564, Tel Aviv, Israel, March 2000.

[FMP94] T. Feder, N. Megiddo, and S. Plotkin. A sublinear parallel algorithm for stable matching. *Fifth ACM-SIAM Symposium on Discrete Algorithms*, pages 632–637, 1994.

[For04] Network Processing Forum. Switch fabric benchmarking group documents: Switch fabric benchmark test suites(NPF 2002.276.08), performance testing methodology for fabric benchmarking(NPF 2003.213.06), fabric benchmarking traffic models, fabric benchmarking performance metrics, switch fabric benchmarking framework, 2004.

[GKP97] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics; Second Edition.* Addison-Wesley, 1997. GRA r 94:1 1.Ex.

[GKR00] M. W. Goudreau, S. G. Kolliopoulos, and S. B. Rao. Scheduling algorithms for input-queued switches: Randomized techniques and experimental evaluation. In *IEEE INFOCOM*, pages 1634–1643, March 2000.

[GLPS04] P. Giaccone, E. Leonardi, B. Prabhakar, and D. Shah. Delay bounds for combined input-output switches with low speedup. *Performance Evaluation*, 55(1-2):113–128, January 2004.

[GM99] P. Gupta and N. McKeown. Design and implementation of a fast crossbar scheduler. *IEEE Micro*, 19(1):20–28, 1999.

[Gol94] S. Golestani. A self-clocked fair queueing scheme for broadband applicatons. In *IEEE INFOCOM*, pages 636–646, April 1994.

[Gou06] A. Gourgy. *On Packet Switch Scheduling in High-Speed Data Networks.* PhD thesis, McMaster University, January 2006.

[GPS03] P. Giaccone, B. Prabhakar, and D. Shah. Randomized scheduling algorithms for high-aggregate bandwidth switches. *IEEE J. Select. Areas Commun.*, 21(4):546–559, May 2003.

[GS62] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *Amer. Math. Monthly*, 69:9–15, 1962.

[GSD05] A. Gourgy, T. H. Szymanski, and D. G. Down. On tracking the behaviour of an output-queued switch using an input-queued switch with unity speedup. Technical report, McMaster University, 2005.

[GVC96] P. Goyal, H. M. Vin, and H. Chen. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In *SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, pages 157–168, New York, NY, USA, 1996. ACM Press.

[GVC97] P. Goyal, H. M. Vin, and H. Cheng. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. *IEEE/ACM Trans. Networking*, 5(5):690–704, October 1997.

[JaJ92] Joseph JaJa. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992. JAJ j 92:1 1.Ex.

[KC03] Y. Kim and H. J. Chao. Performance of exhaustive matching algorithms for input-queued switches. In *Proc. IEEE ICC*, volume 3, pages 1817 – 1822, May 2003.

[KHM87] M. Karol, M. Hluchyj, and S. Morgan. Input versus output queueing on a space-division switch. *IEEE Trans. Commun.*, 35:1347–1356, December 1987.

[KK03] S. Kumar and A Kumar. On implementation of scheduling algorithms in high speed input queuing cell switches. In *Proc. IEEE ICC*, 2003.

[KKLS03] I. Keslassy, M. Kodialam, T. V. Lakshman, and D. Stiliadis. On guaranteed smooth scheduling for input-queued switches. In *IEEE INFOCOM*, 2003.

[KM95] P. R. Kumar and S. P. Meyn. Stability of queueing networks and scheduling policies. *IEEE Trans. Automat. Contr.*, 40, February 1995.

[KM01] I. Keslassy and N. McKeown. Analysis of scheduling algorithms that provide 100% throughput in input-queued switches. In *Proc. of the 39th Annual Allerton Conference on Communication, Control, and Computing*, October 2001.

[KM03] I. Keslassy and N. McKeown. Maximum size matching is unstable for any packet switch. *IEEE Commun. Lett.*, 7(10):496–498, October 2003.

[KP05] M. Katevenis and G. Passas. Variable-size multipacket segments in buffered crossbar (CICQ) architectures. In *Proc. IEEE ICC*, May 2005.

[KPCS99] P. Krishna, N. Patel, A. Charny, and R. Simcoe. On the speedup required for work-conserving crossbar switches. *IEEE J. Select. Areas Commun.*, 17:1057–1066, June 1999.

[KS99] A. Kam and K. Siu. Linear-complexity algorithms for qos support in input-queued switches with no speedup. *IEEE J. Select. Areas Commun.*, 17(6):1040–1056, June 1999.

[KSBS98] A. Kam, K. Y. Siu, R. A. Barry, and E. Swanson. A cell switching WDM broadcast LAN with bandwidth guarantee and fair access. *J. Lightwave Technol.*, 16:2265–2280, December 1998.

[Lei92] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes.* Morgan Kaufmann, 1992. LEI f 92:1 1.Ex.

[LF80] R.E. Ladner and M.J. Fischer. Parallel prefix computation. *JACM*, 27(4):831–838, October 1980.

[Li92] S. Q. Li. Performance of a nonblocking space-division packet switch with correlated input traffic. *IEEE Trans. Commun.*, 40(1):97–108, January 1992.

[Li04] Yihan Li. *Design and Analysis of Scheduling for High Speed Input Queued Switches.* PhD thesis, Polytechnic University, January 2004.

[LMNM01a] E. Leonardi, M. Mellia, F. Neri, and M. A. Marsan. Bounds on average delays and queue size averages and variances in input-queued cell-based switches. In *IEEE INFOCOM*, pages 1095–1103, Alaska, April 2001.

[LMNM01b] E. Leonardi, M. Mellia, F. Neri, and M. A. Marsan. On the stability of input-queued switches with speed-up. *IEEE/ACM Trans. Networking*, 9:104–118, February 2001.

[LMNM03] E. Leonardi, M. Mellia, F. Neri, and M. A. Marsan. Bounds on delays and queue lengths in input-queued cell switches. *JACM*, 50:520–550, July 2003.

[LPC01] Y. Li, S. Panwar, and H. J. Chao. On the performance of a dual round-robin switch. In *IEEE INFOCOM*, pages 1688–1697, April 2001.

[LPC02] Y. Li, S. Panwar, and J. H. Chao. The dual round-robin matching switch with exhaustive service. In *Workshop on High Performance Switching and Routing*, pages 58–63, May 2002.

[LTWW94] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Trans. Networking*, 2(1):1–15, February 1994.

[Luc04] Lucent technologies, Inc., Holmdel, NJ., 2004.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996. LYN n 96:1 P-Ex.

[Man89] Udi Manber. *Introduction to Algorithms. A Creative Approach.* Addison-Wesley, 1989. MAN u 89:1 P-Ex.

[McK95] N. McKeown. *Scheduling algorithms for input-queued cell switches.* PhD thesis, Univ. of California, Berkeley, 1995.

[McK99] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Networking*, 7(2):188 –201, April 1999.

[Meh84] Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching.* EATCS Monographs on Theoretical Computer Science. Springer, 1984. MEH k 84:1 2.P-Ex.

[Mek98] A. Mekkittikul. *Scheduling Non-uniform Traffic in High Speed Packet Switches and Routers.* PhD thesis, Stanford University, November 1998.

[MIM+] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz. The tiny tera: A packet switch core. In *IEEE Micro.*

[Min02] C. Minkenberg. Work-conservingness of CIOQ packet switches with limited output buffers. *IEEE Commun. Lett.*, 6:452 –454, October 2002.

[MM98] A. Mekkittikul and N. McKeown. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *IEEE INFOCOM*, pages 792–799, 1998.

[MMAW99] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Trans. Commun.*, 47(8):1260 –1267, August 1999.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge Univ. Press, 1995. MOT r 95:1 1.Ex.

[MRS03] B. Magill, C. Rohrs, and R. Stevenson. Output-queued switch emulation by fabrics with limited memory. *IEEE J. Select. Areas Commun.*, pages 606–615, May 2003.

[MS03] S. Mneimneh and K. Y. Siu. On achieving throughput in an input queued switch. *IEEE/ACM Trans. Networking*, 11(5):858–867, October 2003.

[MVW93] N. McKeown, P. Varaiya, and J. Warland. Secheduling cells in an input-queued switch. *IEE Electron. Lett.*, 29(25):2174–2175, December 1993.

[NB02] N. Nan and L. N. Bhuyan. Fair scheduling in internet routers. *IEEE Trans. Comput.*, 51(6):686–701, June 2002.

[PCB+98] C. Partridge, P. P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, T. Ma, J. Mcallen, T. Mendez, W. C. Milliken, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober, G. D. Troxel, D. Waitzman, and S. Winterble. A 50-Gb/s IP router. *IEEE/ACM Trans. Networking*, 6(3):237–248, June 1998.

[PG93] A. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Networking*, 1(3):344–357, June 1993.

[PG94] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM Trans. Networking*, 2:137–150, April 1994.

[Raw92] Gregory J. E. Rawlins. *Compared to What? - An Introduction to the Analysis of Algorithms.* Principles of Computer Science Series. Freeman, 1992. RAW g 92:1 1.Ex.

[RGT04] M. Rosenblum, M. X. Goemans, and V. Tarokh. Universal bounds on buffer size for packetizing fluid policies in input queued, crossbar switches. In *IEEE INFOCOM*, pages 1126 – 1134, March 2004.

[Sed90] Robert Sedgewick. *Algorithms in C.* Addison-Wesley Series in Computer Science. Addison-Wesley, 1990. SED r 90:1.

[SF96] Robert Sedgewick and Philippe Flajolet. *An Introduction to the Analysis of Algorithms.* Addison-Wesley, 1996. SED r 96:1 1.Ex.

[SGP02] D. Shah, P. Giaccone, and B. Prabhakar. Efficient randomized algorithms for input-queued switch scheduling. *IEEE Micro*, 22(1):10–18, 2002.

[SV95] D. Stiliadis and A. Varma. Providing bandwidth guarantees in an input-buffered crossbar switch. In *IEEE INFOCOM*, pages 960–968, April 1995.

[SV96] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Trans. Networking*, 4(3):375–385, June 1996.

[Szy97] T. Szymanski. Design principles for self-routing nonblocking connection networks with $n \log(n)$ bit-complexity. *IEEE Trans. Comput.*, pages 1057–1069, October 1997.

[Szy02] T. H. Szymanski. Circular prefix computations. Technical report, McMaster University, 2002.

[Tar83] Robert Endre Tarjan. *Data Structures and Network Algorithms.* Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1983. TAR r 83:1 P-Ex.

[Tas98] L. Tassiulas. Linear complexity algorithms for maximum througput in radio networks and input queued switches. In *IEEE INFOCOM*, pages 533–539, 1998.

[TF88] Y. Tamir and G. L. Frazier. High-performance multi-queue buffers for vlsi communications switches. In *ISCA '88: Proceedings of the 15th Annual International Symposium on Computer architecture*, pages 343–354, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

[TGT01] V. Tabatabaee, L. Georgiadis, and L. Tassiulas. QoS provisioning and tracking fluid policies in input queueing switches. *IEEE/ACM Trans. Networking*, 9(5), October 2001.

[ZB03] X. Zhang and L. N. Bhuyan. Deficit round robin scheduling for input-queued switches. *IEEE J. Select. Areas Commun.*, 21(4):584–594, May 2003.