

## ABSTRACT

Title of Dissertation: SCHEDULING AND RATE PROVISIONING FOR  
INPUT-BUFFERED CELL BASED SWITCH FABRICS

Vahid Tabatabaee, Doctor of Philosophy, 2003

Dissertation directed by: Professor Leandros Tassiulas  
Department of Electrical and Computer Engineering

In this dissertation, we develop and analyze algorithms for scheduling in input-buffered switch fabrics. We have introduced new deterministic and randomized scheduling algorithms that are capable of rate provisioning, achieves 100% throughput and have lower complexity than other proposed solutions. We consider QoS provisioning in general and rate provisioning in particular as the basic requirements for the next generation switch fabrics.

To do rate provisioning, we extend the concept of packetized tracking policies for fluid policies to the input-buffered switches. It is considered that the speed up of the switch is one and the fluid policy is feasible, i.e., utilization of all ports is less than one. For the  $2 \times 2$  switches, we show that ideal non-anticipative tracking policies always exist. By ideal, we mean a tracking policy that is at most one cell

behind the corresponding fluid policy. Using a  $3 \times 3$  counter example, we show that non-anticipative policies do not generally exist. For the  $N \times N$  switches, a heuristic tracking policy is provided.

The encouraging results for the heuristic policy motivated us to explore for analytical result for its performance. This effort leads us to the introduction of maximum node contained matching (MNCM) a new class of deterministic maximal size matching algorithms. We use fluid model techniques to prove that these algorithms achieve 100% throughput with no speedup. The only assumption on the arrival pattern is that it satisfies strong law of large numbers. We also introduce a new weighted matching algorithm in MNCM, maximum first matching (MFM) with complexity  $O(N^{2.5})$ . MFM, to the best of our knowledge, is the lowest complexity deterministic algorithm that delivers 100% throughput. We extend the concept of MNCM schedulers and introduce the Maximum Size Unit Interval Matching (MSUIM) algorithm for rate provisioning

Finally, we propose a general parallel architecture for self-randomized algorithms that is appropriate for practical applications. We introduce the concept of max-min fair self-randomized scheduling algorithms for rate provisioning. Using fluid model technique, we provide analytical results for the performance of the self-randomized schedulers.

SCHEDULING AND RATE PROVISIONING FOR  
INPUT-BUFFERED CELL BASED SWITCH FABRICS

by

Vahid Tabatabaee

Dissertation submitted to the Faculty of the Graduate School of the  
University of Maryland, College Park in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
2003

Advisory Committee:

Professor Leandros Tassiulas, Chairman and Advisor  
Professor Anthony Ephremides  
Professor Samir Khuller, Dean's Representative  
Professor Richard La  
Professor Uzi Vishkin

©Copyright by  
Vahid Tabatabaee  
2003

## DEDICATION

*To my mother who taught me the love for life,  
to my father who taught me the love for knowledge,  
to my wife Mojgan who is my source of inspiration and,  
to my son Omid who reminds me of these all everyday.*

## ACKNOWLEDGEMENTS

I would like to thank my advisor, professor Leandros Tassiulas for his support and guidance during my studies. He introduced me to the fascinating field of communication networks and gave me the opportunity to work on several interesting problems in this field. His support and understanding has quite often gone beyond the academic matters during this period.

I am specially grateful to professor Leonidas Georgiadis from Aristotle University. During my research, Leo has been a second advisor to me, and working with him was a great experience. I have learned a lot from his research and problem solving methodology. In fact, results of chapter 2 are outcome of our joint work.

I am thankful to the members of the dissertation committee, Professors Anthony Ephremides, Samir Khuller, Richard La and Uzi Vishkin for kindly reviewing this dissertation and serving on my thesis committee.

In particular, I would like to thank professor Samir Khuller for several discussions that we had on the complexity of alternative matching algorithms for scheduling. These early discussions motivated me to pursue a path that resulted in the introduction of the maximum node contained matching algorithms in chapter 3.

During the course of my academic studies, I was fortunate to work and become friend with some of the brightest and kindest people that I have ever known. I cherish and honor their friendship and thank them for what I have learned from them during this long journey.

Last, but not least, I express sincere gratitude to my parents for all the sacrifices, to my wife Mojgan for all the support and patience and to my dear son Omid for his cheerfulness.

# TABLE OF CONTENTS

<b>List of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Overview . . . . .	1
1.1.1 Data Path in a packet switching system . . . . .	1
1.1.2 Switch Fabric Architectures . . . . .	9
1.2 Problem Statement . . . . .	16
1.3 Related Work . . . . .	19
1.3.1 Maximum Size Matching . . . . .	20
1.3.2 Maximum Weighted Matching . . . . .	21
1.3.3 Parallel Matching Algorithms . . . . .	22
1.3.4 Randomized Scheduling Algorithms . . . . .	23
1.3.5 Fluid Model Techniques . . . . .	26
1.4 Contributions of Dissertation . . . . .	27
1.4.1 Packetized Tracking Policies . . . . .	27
1.4.2 The Maximum Node Contained Matchings . . . . .	29
1.4.3 Self-randomized Scheduling Algorithms and Rate Provisioning	32

<b>2</b>	<b>Rate Provisioning and Tracking Fluid Policies in Input-buffered Switches</b>	<b>35</b>
2.1	Fluid and Packetized Tracking Policies . . . . .	37
2.2	Multi-periodic TDMA Satellite Switches . . . . .	39
2.3	The $2 \times 2$ switch . . . . .	41
2.4	Heuristic Algorithms . . . . .	56
2.4.1	Port Based Tracking . . . . .	59
2.4.2	Critical Ports and Links . . . . .	67
2.4.3	Fix Rate Scheduler Simulation . . . . .	70
2.5	Summary and Conclusion . . . . .	74
<b>3</b>	<b>MNCM: A class of efficient maximal size matching scheduling algorithms with no speedup</b>	<b>76</b>
3.1	Model and Definition . . . . .	79
3.2	Link and Port Fluid Models . . . . .	82
3.3	Stability Results . . . . .	87
3.3.1	Comparison of the MNCM and LPF . . . . .	90
3.4	Maximum First Matching (MFM) . . . . .	91
3.5	Rate Provisioning and the MSUIM Scheduling Algorithm . . . . .	98
3.6	Simulations . . . . .	102
3.7	Summary . . . . .	104
<b>4</b>	<b>Analysis and design of self-randomized max-min fair schedulers</b>	<b>105</b>
4.1	Models and Definitions . . . . .	109
4.1.1	Randomized Scheduling Algorithms and Fluid model . . . . .	109
4.2	Stability of Randomized Schedulers . . . . .	112

4.3	General Architecture . . . . .	115
4.4	Backlogged weight function . . . . .	117
4.4.1	B1 : Self-Randomized Based on Instantaneous Arrivals . . .	118
4.4.2	B2 : Self-Randomized Based on Total Arrivals . . . . .	119
4.4.3	Simulation Results . . . . .	121
4.5	Max-min fair scheduling weight function . . . . .	123
4.5.1	A1 : Self-Randomized Based on Instantaneous Token and Cell Arrivals . . . . .	129
4.5.2	A2 : Self-Randomized Based on Total Token and Cell Arrivals	132
4.5.3	A4 : Self-Randomized Based on Total Link Weight and Cell Arrivals . . . . .	133
4.5.4	Simulation Results . . . . .	136
<b>5</b>	<b>Summary and concluding remarks</b>	<b>146</b>
	<b>Bibliography</b>	<b>150</b>

## LIST OF TABLES

2.1	Example of Sub-permutation Matrices . . . . .	45
2.2	Performance for different inspection horizons ( $U=(0.85, 0.8)$ , $N=32$ )	73
2.3	Performance for different switch sizes ( $U=(0.85, 0.8)$ ) . . . . .	74
2.4	Performance for different utility pairs ( $N=32$ ) . . . . .	74

## LIST OF FIGURES

1.1	Block Diagram of a Data Switching System. . . . .	2
1.2	Schematic Diagram of a Chassis Based Switch. . . . .	3
1.3	A Three Layer Hierarchical Scheduler . . . . .	5
1.4	Sources of Cell Overhead in a Switching System. . . . .	8
1.5	Schematic Architecture for a $4 \times 4$ Shared Memory Switching and Scheduling Elements. . . . .	11
1.6	Schematic Architecture for a $4 \times 4$ Buffered Crossbar Switching and Scheduling Elements. . . . .	14
1.7	Schematic Architecture for a $4 \times 4$ Input Buffered Switch Fabric. . . . .	15
1.8	Schematic Architecture of an Input-Buffered Switch Fabric. . . . .	17
2.1	Effect of Cell $p_3$ on completion times of $p_1$ and $p_2$ . . . . .	43
2.2	Cell transfer from input port $i$ to output port $j$ . . . . .	46
2.3	Arrangement of cells for Case 1 in the proof of Theorem 1. . . . .	49
2.4	Arrangement of cells for Case 2 in the proof of Theorem 1. . . . .	52
2.5	The backlogged cells for the fluid and packetized tracking. . . . .	60
2.6	Example of node and link weight matching in bipartite graphs. . . . .	64

3.1	Four possible forms of sub-graphs in a combined matching. Links of $M_1$ are shown with solid lines and $M_2$ with dashed lines. Marked nodes are shown as black nodes. . . . .	94
3.2	Average Delay v.s. Throughput for different matching algorithms. The output queueing system is modelled as an M/D/1 system. . . . .	104
4.1	The generic block diagram adopted for self-randomized schedulers. . . . .	116
4.2	Mean delay for B1 under diagonal traffic. . . . .	123
4.3	Mean delay for B1 under log-diagonal traffic. . . . .	124
4.4	Mean delay for B1 under uniform traffic. . . . .	124
4.5	Mean delay for B2 under diagonal traffic. . . . .	125
4.6	Mean delay for B2 under log-diagonal traffic. . . . .	125
4.7	Mean delay for B2 under uniform traffic. . . . .	126
4.8	Mean delay for A1 under diagonal traffic. . . . .	137
4.9	Mean delay for A1 under log-diagonal traffic. . . . .	137
4.10	Mean delay for A1 under uniform traffic. . . . .	138
4.11	Mean delay for A2 under diagonal traffic. . . . .	139
4.12	Mean delay for A2 under log-diagonal traffic. . . . .	139
4.13	Mean delay for A2 under uniform traffic. . . . .	140
4.14	Mean delay for A4 under diagonal traffic. . . . .	141
4.15	Mean Delay for A4 under log-diagonal traffic. . . . .	142
4.16	Mean Delay for A4 under uniform traffic. . . . .	142
4.17	Mean delay for different schemes under diagonal traffic. . . . .	143
4.18	Mean delay for different schemes under log-diagonal traffic. . . . .	144
4.19	Mean delay for different schemes under uniform traffic. . . . .	144
4.20	Serving rate adaptation to arrival pattern changes. . . . .	145

# Chapter 1

## Introduction

### 1.1 Overview

Due to the progress in optical transmission technology and increase in Internet traffic, very fast switching technology are necessary for the internet core and edge switching and routing systems. In this section, we introduce the basic architecture and components of a switching system in communication networks. We then focus on the switch fabric component and review a number of common architectures that are used for switch fabrics.

#### 1.1.1 Data Path in a packet switching system

Consider the generic packet switching system in figure 1.1. We focus on the data plane components. There are two cards shown in the picture: the line card, and the switch card. In a real chassis based switching system, there are multiple line cards, and depending on the capacity, and the redundancy model of the system there can be one or multiple switch cards (figure 1.2). The Line cards are interface

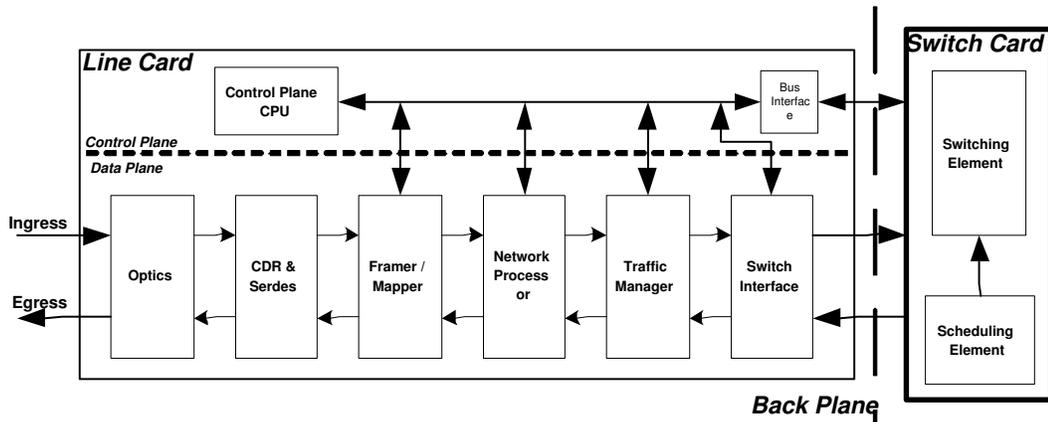


Figure 1.1: Block Diagram of a Data Switching System.

of the switching system with the outside world. Packets or cells enter the system through ingress interface of the line cards, and exit the system through the egress interface of them. The Switch card task is to allow data units that enter the system through an ingress line card get to their targeted egress line card.

### Line Card Functions and Components

We start from the line interface side of the line card. The first block is the optics that does the optical to electrical and electrical to optical conversions. Next block performs clock and data recovery (CDR) and deserialization (serialization) of bit stream in the ingress (egress) direction. The framer/mapper extracts payload from the layer 1 framing protocol (SONET) in the ingress direction and maps data back into the frames in the egress direction.

The Packet processing starts from the network processor block, which is mainly a packet classifier. In the ingress direction it identifies the flows based on the multi-field lookup. Multi-field lookup, for instance, can be based on the IP N-tuple lookup, or MPLS header. After flow identification it determines the egress

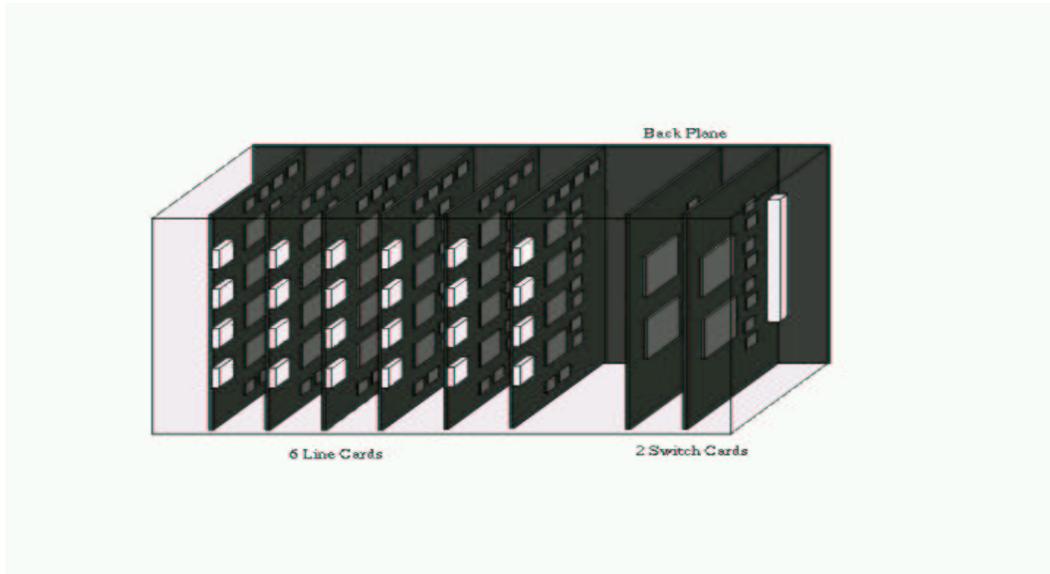


Figure 1.2: Schematic Diagram of a Chassis Based Switch.

forwarding port address, mark the QoS parameters, and append the system internal header information for traffic manager.

Next block is the traffic manager that performs the following functions:

- Police ingress traffic, for instance based on IETF DiffServ or ITU GCRA.
- Provide congestion management based on WRED profiles.
- Enqueue packets into class based queues per destination port.
- Schedule ingress traffic into the fabric interface using appropriate scheduling techniques.
- Segment packets into fixed sized switch-cells for switch fabric.
- Reassemble switch-cells back into packets in the egress direction.
- Shape the egress traffic into outgoing lines.

- Schedule the egress traffic using the appropriate dequeuing method.

The switch fabric interface unit is a queuing management unit that resides on the line cards. Even though it is located on the line card it is considered as part of the switch fabric. It buffers switch cells and sends request for connection to the other ports. Once a request is granted by the switch fabric scheduling element, it schedules a cell from one of the class based queues for that egress port.

The fabric interface unit also manages the flow control messaging between the line card and the switch card(s). Note that the switch fabrics are usually designed based on a lossless model, where as the traffic manager is entitled to discard packets under congestion using WRED profile for example.

Traffic managers are usually capable of supporting a large number flows using a hierarchical scheduling model [3]. Consider the three layer hierarchical scheduler shown in figure 1.3. In practice there can be more than three layers of scheduling. The third layer scheduler invokes one of the second layer schedulers. The selected second layer scheduler invokes one of the first layer schedulers, that selects a flow in turn.

In a desirable design with  $k$  layers of scheduling in the traffic manager, number of the queues that are supported in the switch fabric interface unit should match the number of schedulers in the  $k - 1$  layer of the traffic manager. In this way, we would not lose queue granularity in the switch fabric interface, and the fabric interface can potentially functions as an extension of the traffic manager scheduler.

Performance of the traffic manager scheduler, is highly dependent on the switch fabric interface behavior. Usually switch fabrics have a limited space for buffering the cells, and are relying on a flow control mechanism between the fabric interface and the traffic manager to control the data flow, and avoid data loss. In essence, the

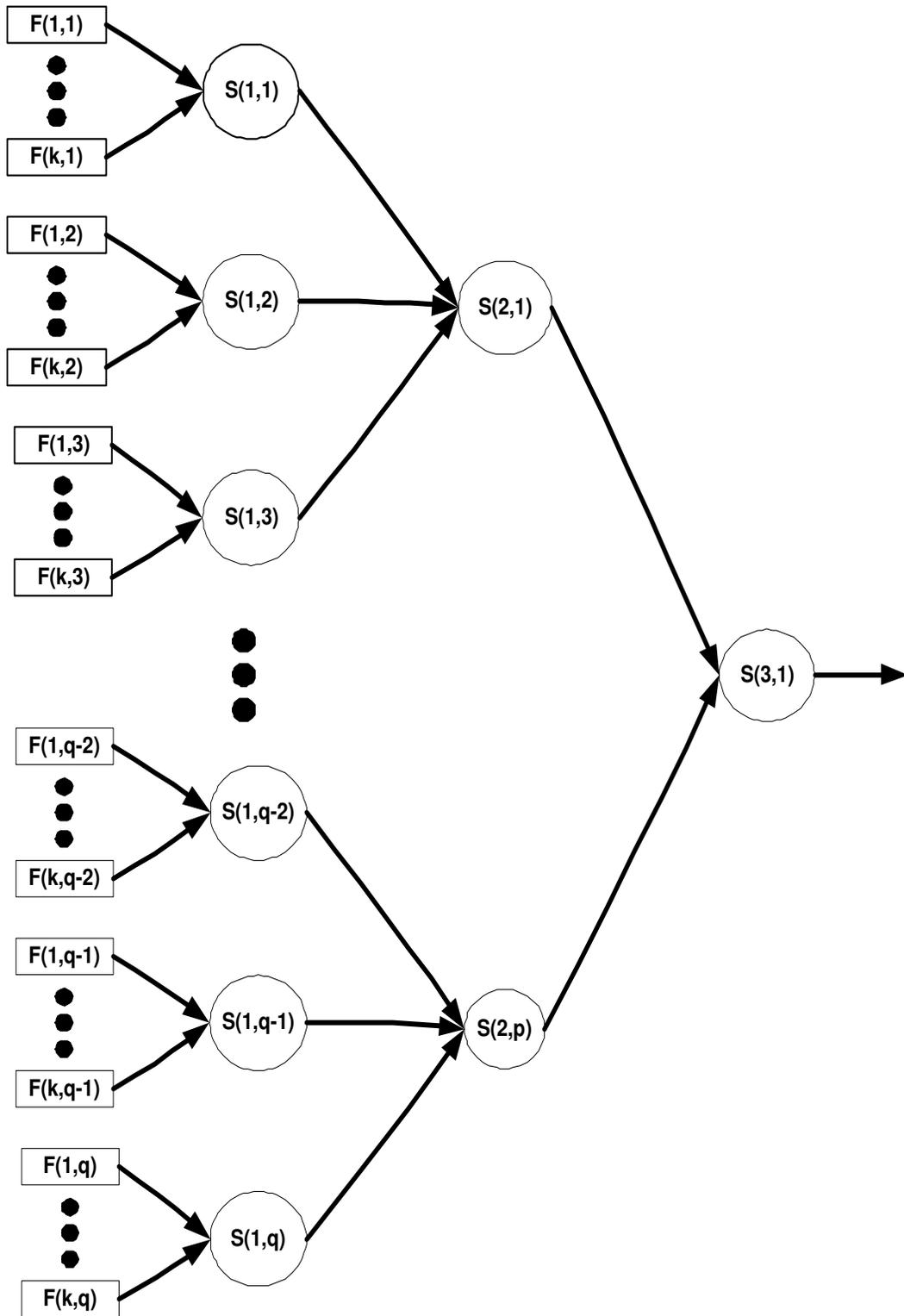


Figure 1.3: A Three Layer Hierarchical Scheduler .

switch fabric can block the traffic manager scheduler and impacts its performance.

The scheduling techniques that are generally used and employed in the traffic managers, such as weighted round robin (WRR), and weighted fair queuing (WFQ), can provide service guarantees to the flows [10], [11], [32], [33], [18]. In order to maintain a manageable and non-disruptive transfer of data through the switch fabric, it is necessary to be able to control and manage the serving rate of the queues in the switch fabric too.

### **Switch Card**

The interface between the line and switch cards are established through high speed serial links (HSSL) that are passed through the backplane. Reliable data rate over the backplane is limited to 3.125 Gbps with today's technology, even though a number of companies and research labs are working on higher speed prototypes. Therefore, to achieve the required rates each line card uses multiple high speed serial links in parallel. The problem becomes even more complicated, when we realize that the serial links overhead reduces the effective data rate even further. As an example consider an HSSL with raw data rate of 3.125 Gbps. Using 8b/10b line coding the effective data rate reduces to 2.5 Gbps. Therefore, for a line rate of 10 Gbps, and speedup factor of 2, we need to run 8 parallel active and 8 redundant HSSL from each line card. In a system with 16 line cards this adds up to 256 bidirectional high speed serial links. Each pair of HSSLs of a line card is connected to a separate switching element in the switch card. For instance, in the previous example, there will be 16 separate switching elements working in parallel.

Number of HSSLs and switching elements is directly related to the complexity of the backplane design, number of switch cards, power dissipation and ultimately

cost of the system. Therefore, it should be clear that having efficient scheduling algorithms that do not rely on extensive speedup factors is highly desirable.

The scheduling unit works as the arbiter between the line cards. While the traffic manager schedulers perform a many to one decision between flows residing on a single line card, the scheduling unit of the switch fabric performs a many to many arbitration between the line cards. Generally, the switch interface unit sends a series of connection requests to the switch scheduling unit, based on the occupancy of its local queues. The scheduling unit arbitrates between the requests, and at every time accepts a set of non-conflicting requests. Accepted messages are sent back to the switch interface unit, and they send the accepted cells to the switching unit.

### **Effective Speedup**

In the previous section, we emphasized that speedup is costly and it can not be increased arbitrarily. However, there is a misnomer in the industry that scheduling inefficiencies can be compensated with moderate speedup factors around 2. The problem is in definition of speedup and the way it is calculated.

We defined speedup based on the switch interface unit interfaces in figure 1.1. Let  $r_{SF}$  be the cell rate over the line between the switch interface unit and the switch card,  $r_{TM}$  be the cell rate over the line between the switch interface unit and the traffic manager. Speedup factor  $s$  is,

$$s = \frac{r_{SF}}{r_{TM}} \tag{1.1}$$

The discrepancy in definition occurs, when for some switch fabrics the incoming rate of data to the line card is used instead of  $r_{TM}$  in speedup calculations. For

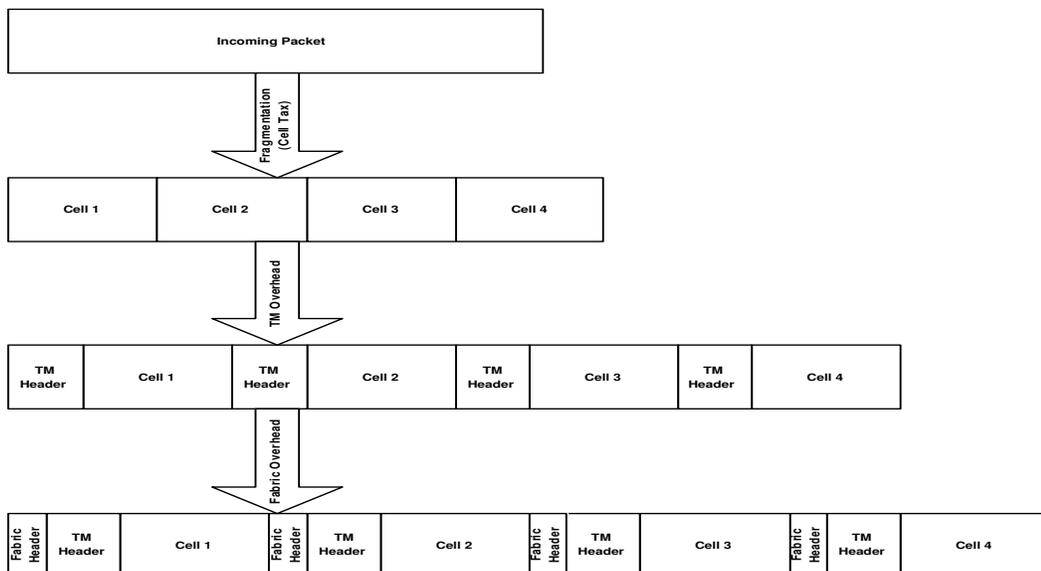


Figure 1.4: Sources of Cell Overhead in a Switching System.

instance, for a 10Gbps line rate, due to the system overhead,  $r_{TM}$  can be higher than 15Gbps. There are several factors contributing to the system overhead that are illustrated in 1.4. Cell tax is due to segmentation of variable size packets into fixed size cells. This process usually ends with a cell that carries last part of the packet and is augmented with zero's to fill the cell size. The traffic manager and the switch fabric headers are two other main sources of the overhead.

For instance, consider a 40 byte packet that has to be carried over a switch with 72 byte cell size. For the header size, consider 8 byte header for the fabric and 12 byte header for the traffic manager. In this system 10Gbps line rate, results in 18Gbps of  $r_{TM}$ , and with  $r_{SF} = 20$ Gbps its effective speedup factor is 1.1 rather than 2.

Our discussion on the system overview and role of the switch fabric in the system provides enough background to highlight major requirements of a switch fabric:

- **Number of Queues:** Number of the queues in the switch fabric interface unit should match the hierarchical scheduler structure of the traffic manager unit.
- **Queue Management:** Similar to traffic managers, switch fabrics should provide service guarantees to each one of the queues that they recognize.
- **Efficient Scheduling:** Scheduling algorithm should be effective enough and can not rely on the speedup to achieve high throughput.

### 1.1.2 Switch Fabric Architectures

In the previous section we gave an overview of a switching system architecture and components. The discussion was ended with a short list of major performance issues and requirements for the next generation switch fabrics. In this section, we build up on that discussion by reviewing the main switch fabric architectures, their challenges and advantages.

#### Output-Buffered Switch Fabric

In this architecture all cells are buffered in the egress side of the switch fabric, and there is a fixed delay for all cells through the fabric. This architecture requires that the interconnection through the fabric works  $N$  times faster the cell rate on the line card, i.e.  $r_{SF} = Nr_{TM}$ , so that all ingress ports can forward one cell to their targeted egress ports in every cell time. Moreover, the buffers in the egress ports should support  $N$  writes and 1 read in every cell time.

There is no contention in this architecture, since every ingress port has its dedicated reserved time for forwarding the cells. The buffering is only at the

output side of the switch fabric, and the conventional many to one scheduling algorithms are sufficient to provide service guarantees.

However, this architecture has serious scalability problem. As the number of ports and the line rate increases, it is simply not possible to meet the interconnection bandwidth and memory speed requirements of this architecture.

### **Shared Memory Switch Fabric**

In this architecture the switching element is a shared memory. Consider the 4x4 switching element in figure 1.5. The ingress interface units forward the cells to the switching unit. The switching unit is in essence a shared memory that constitutes from multiple queues per egress port. In figure 1.5 there are 4 queues, supporting 4 classes of service, for each egress port. The scheduling unit constitutes of 4 many to one schedulers. Each scheduler is designated to one egress port and performs scheduling between multiple queues of that egress port.

Scalability is still an issue with this structure. This architecture has resolved  $N$  times over speed requirement of the output-buffered switch fabric by using a shared memory in the middle. However, it requires even higher bandwidth memories, since at every cell time we need to perform  $N$  writes and  $N$  reads from the shared memory.

The centralized buffering structure have had other problematic consequences for switch fabrics that are based on this architecture. In general in order to control and provide rate provisioning there should be dedicated queues for every input, output and class of service. However, it is not simple to implement and schedule the required number of queues in a centralized architecture such as shared memory. For instance in a 32x32 switch fabric with 8 classes of service this requires 8192

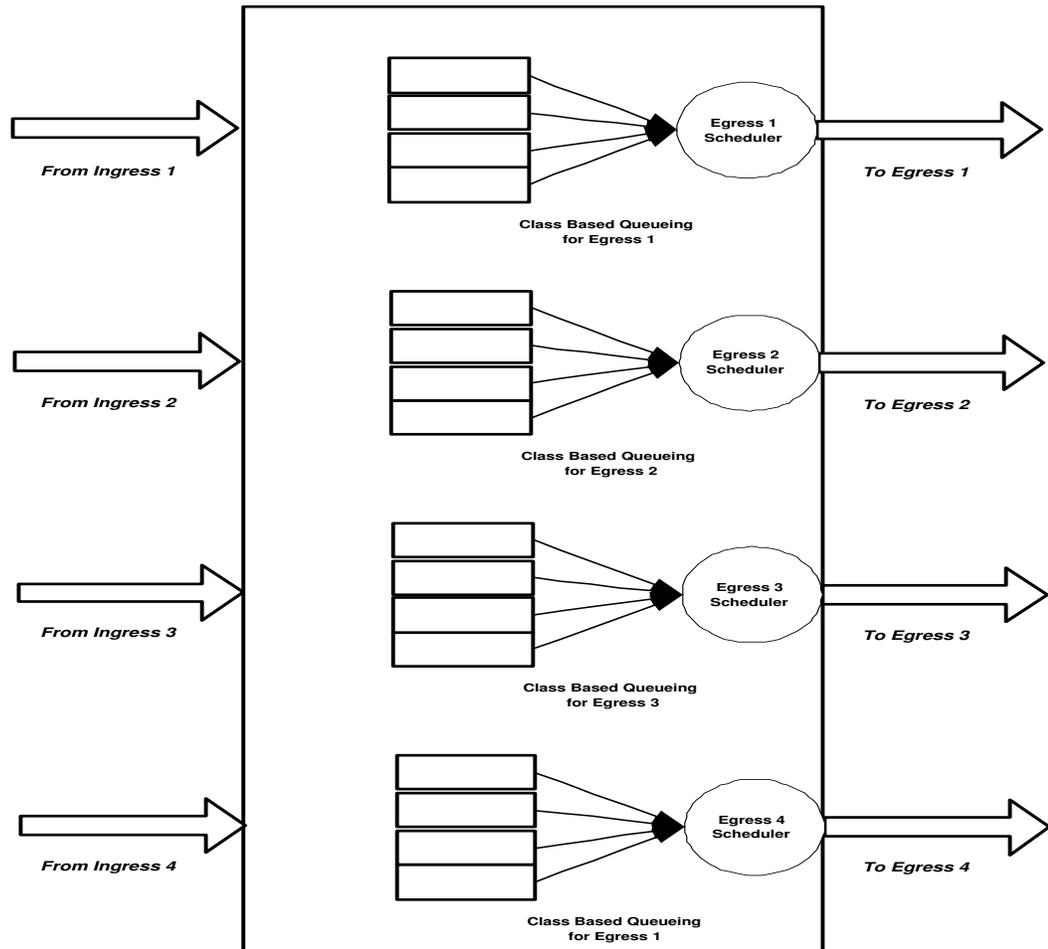


Figure 1.5: Schematic Architecture for a  $4 \times 4$  Shared Memory Switching and Scheduling Elements.

distinct queues. A common solution to this problem is aggregation of queues. For instance, cells that are coming from different ingress ports and have the same priority are buffered in the same queue and it would become impossible to provide service guarantees to them. As an example, let say that we want to provide 0.1 of egress port 1 capacity to input 1 and 0.2 to input 2. However, since cells that are coming from input 1 and input 2 share the same queue in the switch it is impossible to guarantee the rates individually. This is basically aggregation of traffic manager queues in the switch fabric that makes it impossible for the switch fabric to provide service guarantees to each of the traffic manager queues.

Another practical problem with this architecture is the flow control mechanism that should be used between the shared memory element in the switch card and the switch interface units in the ingress line cards (Fig. 1.1). Due to the finite size of the shared memory element that is used for buffering the cells, and to avoid cell drop on the shared memory, a sophisticated flow control signalling should be used between the shared memory and the switch interface units. Basically, at every cell time, the shared memory element should inform the line card, whenever each of its queues are full and it can not accept any more cells. This incurs considerable flow control overhead as well as performance degradation. The problem gets more serious when we realize that the flow control mechanism should be conservative and take into account the flow control latency since it will take several cell times for a flow control message to reach the ingress line card from the switching elements.

### **Buffered Crossbar Switch Fabric**

The buffered crossbar architecture has recently got a lot of attention, and is considered to resolve some of the inherent problems with the shared memory architecture

[7]. In this architecture, the switching element consists of separate buffer units per input, output and class. Consider the  $4 \times 4$  buffered crossbar switch fabric in figure 1.6. There are 2 classes of service supported in this switch fabric. A many to one scheduler is used for each egress port to perform the scheduling between queues that are targeting that egress port. At every cell time there is at most one read and one write to each memory section, therefore memory speed requirements are resolved by using dedicated memory instead of shared memory for queues. However, number of queues, as well as size of the queues are limiting factors in this architecture.

Implementation constraints in the number of queues restricts number of classes of service in the buffered crossbars. This results in aggregation of classes of services in the traffic manager, and because of that switch fabric can not provide service guarantees with the required granularity.

### **Input-Buffered Switch Fabric**

Consider the  $4 \times 4$  input-buffered switch fabric in figure 1.7. Cells are buffered in the switch interface units, and the switching unit is a simple buffer-less crossbar. Cells are queued in separate queues that are called virtual output queue (VOQ). Each VOQ is associated to one output and cells are buffered based on their destination and class of service.

Introduction of VOQs resolves HOL blocking problem in the input buffered switches [16]. HOL blocking occurs if a single FIFO queue is used for all cells in the ingress side. If the cell at the head of the queue can not be forwarded, due to contention with cells from other ingress units, other cells in the queue are also blocked. Note that other cells may have reachable destinations that are idle. In fact

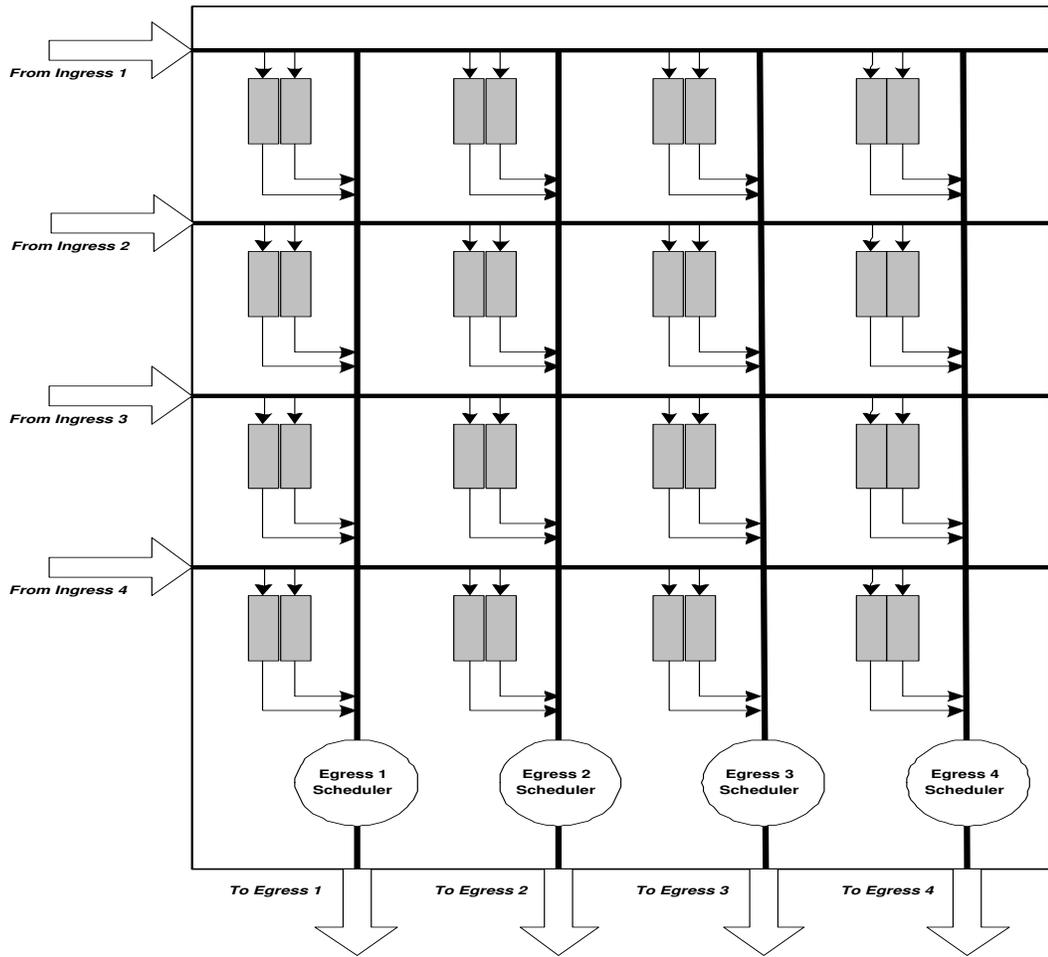


Figure 1.6: Schematic Architecture for a  $4 \times 4$  Buffered Crossbar Switching and Scheduling Elements.

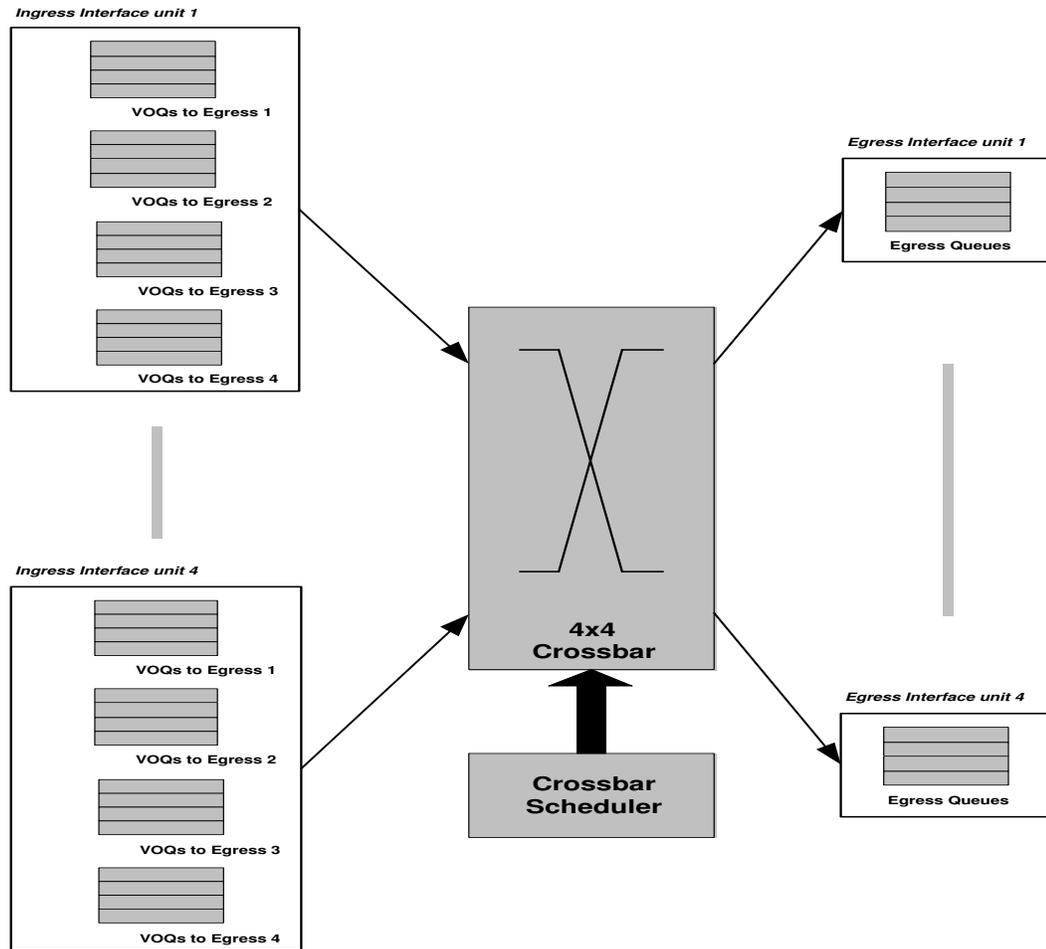


Figure 1.7: Schematic Architecture for a  $4 \times 4$  Input Buffered Switch Fabric.

without VOQs, for Bernoulli i.i.d. arrivals with uniform destination distribution the maximum achievable throughput is 58.6% [26].

In this architecture, buffering is distributed between ingress interface units, and there is no concentrated buffering in the switching element. Therefore, the scalability problems that are related to the memory speed, queue size, and number of queues are not an issue in this architecture.

When contention happens, the crossbar scheduler determines which cells will be forwarded and which ones will stay in their ingress buffers. It is clear that

the performance of this switch fabric is determined by the crossbar scheduler. In essence, crossbar scheduler design is the main challenge of this architecture.

In summary, due to its distributed queue structure, input buffered switch fabric is the appropriate, scalable architecture for next generation switch fabrics. However, the challenge is to come up with a low complexity scheduling algorithm that can provide service guarantees to the queues and provides high throughput at the same time. Design and analysis of such schedulers is the problem that we have tackled in this thesis.

## 1.2 Problem Statement

We consider input queued switches that serve fixed size packets. Each input and output has the capacity of serving 1 cell per unit time. At most  $s$  packet per unit time can be transferred from the input ports to a given output port, where  $s$  is the speedup factor of the switch. To avoid head of line (HOL) blocking, we consider that the buffer at input  $i$  is partitioned into  $N$  queues,  $\text{Queue}(i, 1), \dots, \text{Queue}(i, N)$ . The scheduling policy is basically a matching algorithm  $m$  that based on the state of the switch selects a matching between the inputs and outputs in every time slot. If input  $i$  is matched to output  $j$ , and the  $\text{Queue}(i, j)$  is not empty, a cell is sent from input  $i$  to output  $j$ . A matching can be represented by a permutation matrix  $\pi$ . Input ports are represented by the rows and output ports by the columns of this matrix, and input  $i$  is matched to output  $j$  if and only if  $\pi_{ij} = 1$ .

We assume that the cells arrive at the switch at the beginning of a time slot. A cell that has arrived at the beginning of time slot  $n$  can be scheduled at the same time slot and depart the switch during the same time slot  $n$ . Let  $A_{ij}(n)$  be the number of packets that arrived at input  $i$  and are destined for output  $j$  up to

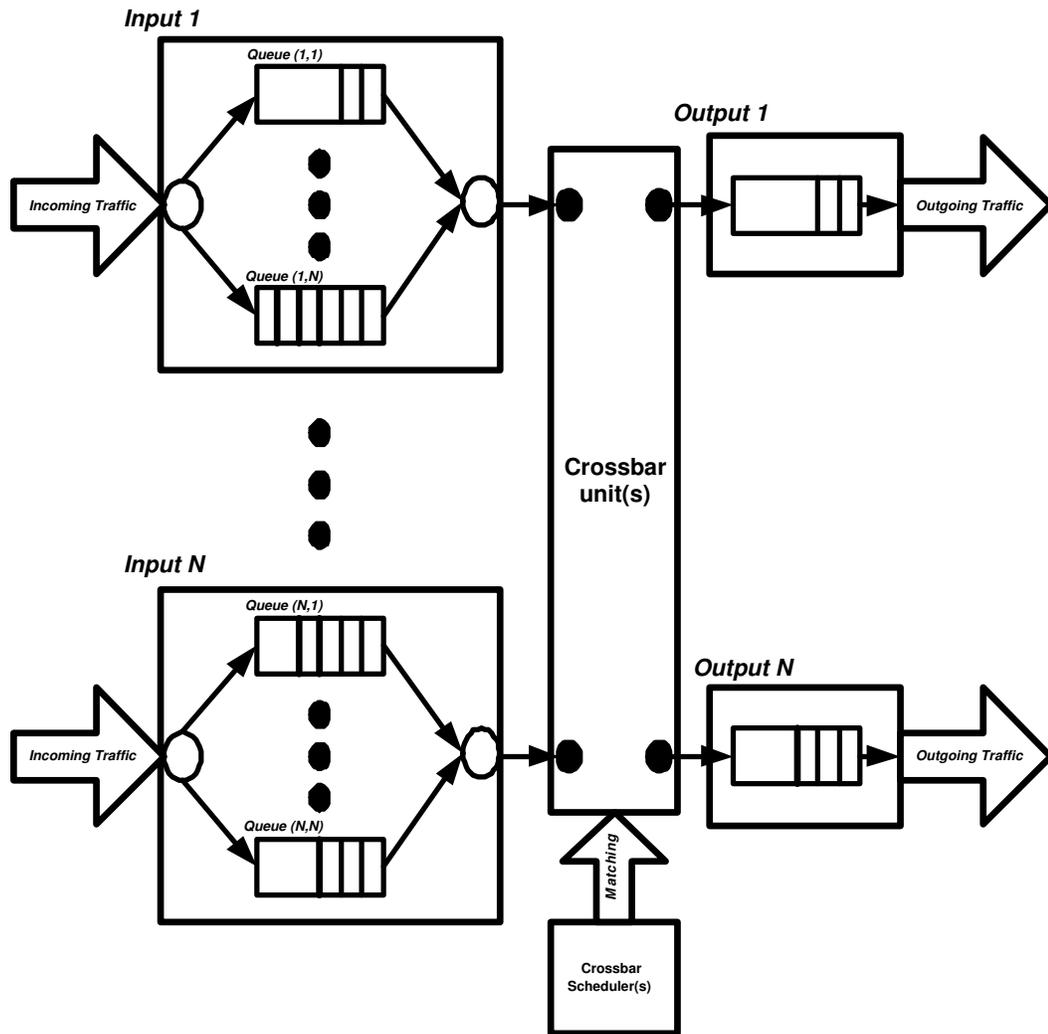


Figure 1.8: Schematic Architecture of an Input-Buffered Switch Fabric.

time  $n$ . We assume that there are no arrivals before time 0, i.e.,  $A_{ij}(0) = 0$ . The arrival processes  $\{A_{ij}(\cdot), i, j = 1, \dots, N\}$  satisfy strong law of large numbers, that is with probability one,

$$\lim_{n \rightarrow \infty} \frac{A_{ij}(n)}{n} = \lambda_{ij} \quad i, j = 1, \dots, N. \quad (1.2)$$

Arrival process is considered to be admissible if no input or output is oversubscribed, i.e.,

$$\begin{aligned} \sum_{i=1}^N \lambda_{ij} &\leq 1 \quad j = 1, \dots, N, \\ \sum_{j=1}^N \lambda_{ij} &\leq 1 \quad i = 1, \dots, N. \end{aligned} \quad (1.3)$$

In this thesis, we are looking for different matching algorithms that can be used for scheduling (arbitration) in input-buffered switch fabrics. There are a number of desirable features for the scheduling algorithms that we attempt to achieve:

- **Stability:** For a given admissible traffic pattern, we define a matching algorithm to be stable (rate stable) if with probability one,

$$\lim_{n \rightarrow \infty} \frac{D_{ij}(n)}{n} = \lambda_{ij} \quad i, j = 1, \dots, N, \quad (1.4)$$

where  $D_{ij}(n)$  is the number of departed cells up to time  $n$ , from input  $i$  to output  $j$ .

- **Delay Efficient:** The average latency of cells in a switch fabric is a key performance metric. A scheduling algorithm is delay efficient, if it has a good delay performance compared to benchmark matching algorithms. Note that we consider the delay efficiency as a comparative metric rather than an absolute metric. The benchmark algorithms will be introduced later.
- **No Starvation:** A queue is starved if for a given traffic pattern, it remains unserved, and non-empty indefinitely. Note that starvation focuses on one

queue and perhaps latency of one specific cell in that queue, while the delay efficient criterion focuses on the average latency in the switch fabric.

- **Complexity:** The matching algorithms that are considered should be implementable in special purpose hardware for application in high speed switches. Therefore, the algorithm should be simple enough to be realized in hardware with finite logic.
- **Guaranteed Service Delivery:** Providing performance (delay, jitter, throughput) guarantees will be a key requirement for next generation switching systems. This can not be achieved at the system level, unless we have granular control over the switch fabric input queues services. The scheduling algorithm should be able to effectively control and manage the serving rate of each of the queues, without sacrificing throughput of the system.

### 1.3 Related Work

In section 1.1, we briefly reviewed and studied alternative switch fabric architectures. Based on the requirements of next generation switch fabrics, we reached the conclusion that the crossbar based input-buffered switch fabric is the appropriate switch fabric architecture. We emphasized that the crossbar scheduler determines performance of the input-buffered switch fabrics, and it is one of the main design challenges of the input-buffered switch fabrics. In this section, we review some of the major techniques, and analytical results for scheduling in input-buffered switches.

### 1.3.1 Maximum Size Matching

In section 1.2, we explained that the crossbar scheduling problem is equivalent to the matching problem in a bipartite graph, where the ingress ports are the first group of vertices (nodes) and the egress ports are the second group of them. There is an edge between two vertices if and only if the corresponding ingress and egress ports are backlogged, i.e. there is a cell in the ingress port destined to the egress port. The scheduling problem is to select a set of connections between backlogged ingress and egress ports, such that no port involves in more than one connection. Equivalently, the problem is to select a matching, i.e. to find a set of the bipartite graph edges, such that no vertex is connected to more than one edge in the set.

Many of the scheduling algorithms that are proposed for the input-buffered switches are originated in graph theory, where they are studied and analyzed as matching algorithm for bipartite graphs.

The maximum size (cardinality) matching for bipartite graphs are one of the most famous problems in the graph theory. Given a bipartite graph  $G$ , the maximum size matching of  $G$  is a matching set of  $G$  edges with maximum cardinality. In other words, the maximum size matching has the maximum possible number of edges among all possible matchings. Several algorithms are proposed for the maximum size matching; the most efficient one has  $O(n^{2.5})$  complexity [23].

Even though the maximum size matching, schedules maximum possible number of cells at every scheduling time, it is not rate stable [29]. Note that for a given bipartite graph the maximum size matching is not necessarily unique. We will later prove that a specifically characterized maximum size matching is rate stable.

### 1.3.2 Maximum Weighted Matching

Consider a bipartite graph  $G$ , where every edge of the graph has a weight value. The maximum weighted matching is a matching of  $G$  with maximum weight, where weight of a matching is the summation of its edges weights. In the scheduling problem, weight of a link can be number of backlogged cells for that connection, or delay of first cell in the associated VOQ. For both mentioned weight functions it is proved that the maximum weighted matching algorithm is rate stable. The stability proofs was originally for Bernoulli i.i.d. arrivals [37], [29], but was later extended to all feasible arrival patterns [13].

In the previous section, we explained that even though the maximum size matching scheduler, transfers maximum possible number of cells in every time slot, it is not rate stable. To achieve 100% throughput the scheduling algorithm needs to take into account other factors. In a rate stable maximum weighted matching algorithm the required information is carried in the weights of the graph.

The maximum weighted matching algorithm is also known as the assignment problem, and can be described by a concise linear program. The corresponding linear program, due to its specific architecture, can be solved using the Hungarian method [27]. Complexity of the Hungarian method that solve the weighted matching problem for a complete bipartite graph is  $O(N^3)$ .

In chapter 3, we will introduce Maximum Node Containing Matchings (MNCM), a class of weighted scheduling algorithms that are not necessarily maximum weighted matching, but are rate stable. We also introduce a specific matching algorithm, Maximum First Matching (MFM) that is rate stable and its complexity is  $O(N^{2.5})$ . To the best of our knowledge, MFM is the lowest complexity rate stable deterministic scheduling algorithm introduced for input-buffered switches.

### 1.3.3 Parallel Matching Algorithms

Most of the solutions that are proposed for the maximum size and the maximum weighted matching algorithms are sequential. Sequential algorithms inherently runs in multiple consequent steps. Number of the steps are usually proportional to the number of ports ( $N$ ) or edges of the graph ( $N^2$ ). Therefore, these algorithms have serious scalability problems, as the number of ports increases. The parallel matching algorithms, on the other hand, have a distributed structure and can be divided into multiple simultaneous tasks.

The most common parallel matching algorithms for scheduling can be implemented on  $2N$  blocks (arbiters) working in parallel. Each arbiter is associated to one of the ingress or egress ports of the system, and makes one selection in every iteration of the algorithm. Each iteration of the algorithm is consisted of three steps:

1. **Request:** Each unmatched input sends a request to every output for which it has a backlogged cell.
2. **Grant:** Each unmatched output arbiter selects one of the requests that are sent to it, and sends a grant to the corresponding input arbiter.
3. **Accept:** Each input arbiter selects one of the grants that are sent to it.

At the end of each iteration, a set of new connections are added to the matching.

One of the main issues with the parallel matching algorithms is arbiter synchronization. Synchronization happens, when multiple output (grant) arbiters select the same input. In that case, only one of them will get the accept signal and the rest will remain unmatched.

Parallel Iterative Matching (PIM) [1] uses a random selection mechanism to avoid synchronization and improve the performance. The PIM performance is not predictable because of its random nature. Furthermore, it can be unfair between the contending connections [28].

The most common parallel matching algorithm is perhaps iSLIP [28], where the arbiters are modified round robin schedulers. The modifications has been done to avoid synchronization, and hence to improve performance of the scheduler. Many of current implementations of input-buffered switch fabrics are using iSLIP, or its modifications.

The iSLIP scheduling algorithm has major unresolved problems. Performance and throughput of iSLIP is very sensitive to the traffic destination distribution. For non-uniform traffic distributions, iSLIP arbiters suffer from synchronization problem, which negatively affect the switch fabric throughput. Moreover, since iSLIP uses simple round robin arbiters, it lacks a proper mechanism for QoS provisioning, and control of the performance.

### 1.3.4 Randomized Scheduling Algorithms

Scalability is one of the main concerns about the input-buffered switch fabrics. In the scheduling algorithms that we have discussed so far, the objective is to find an appropriate matching for a bipartite graph. Some of these algorithms have a good performance, but are too complex for real time implementation in high speed switches. For instance, the MWM algorithm is proved to achieve 100% throughput, but its complexity is  $O(N^3)$ . On the other hand, those matching algorithms that are less complex and are appropriate for high speed switches lack in performance. Parallel matching algorithms fall in the second category.

The basic idea behind randomized scheduling algorithms is, instead of finding a good matching, to select the best matching among an appropriate set of predetermined matchings. The best matching is the matching with the maximum weight. Note that complexity of calculating weight of a matching is  $O(N)$ , and complexity of finding the maximum weight among  $K$  candidates is  $O(K)$ . Therefore for a constant  $K$ , this algorithm has linear complexity in switch size  $N$ .

The key issue here is to have an appropriate set of candidate matchings. The first algorithm that comes into mind is:

**Algorithm 1:**

- At every scheduling time, pick matching  $R$  uniformly and randomly from all possible  $N!$  matchings.
- Use  $R$  as the schedule.

It can be proved that this algorithm does not deliver 100% throughput [19].

Tassiulas introduced the first stable (100 % throughput) randomized scheme with linear complexity [36] as follow,

**Algorithm 2 (Tassiulas):**

- Let  $S(t)$  be the schedule (matching) used at time  $t$ .
- At time  $t + 1$ , pick matching  $R$  uniformly and randomly from all possible  $N!$  matchings.
- Let the schedule at time  $t + 1$ ,  $S(t + 1)$ , be the heavier weighted of  $S(t)$  and  $R(t + 1)$ .

Tassiulas proved that for Bernoulli i.i.d arrivals this algorithm delivers 100% throughput [36]. In fact, he showed that if at step 2 of the algorithm probability of selecting the MWM is  $\epsilon > 0$ , then the matching is stable.

The candidate matching set in this scheme contains two matchings, one randomly selected matching and the matching that was used in the previous time slot. Intuitively, since state of the switch (number of backlogged cells) changes slightly in every time slot, a good matching in previous slot will remain a good matching for the next time slot.

Although this randomized algorithm is stable, it does not have a good delay performance. Paper [19] proposed several modifications to the basic randomized algorithm to improve its delay performance. One of the contributions of [19] is introduction of the self-randomized algorithms. The idea behind self-randomized scheduling algorithms is instead of selecting a pure random matching, using the arrival cell pattern as the source of randomization. SERENA is the self-randomized algorithm introduced in [19]:

**Algorithm 3 (SERENA):**

- Let  $S(t - 1)$  be the schedule (matching) used at time  $t - 1$ .
- Let  $A(t) = [a_{ij}(t)]_{N \times N}$  denote the arrival graph, where  $a_{ij}(t) = 1$  indicates arrival, and  $a_{ij}(t) = 0$  indicates no arrival.
- Let the schedule at time  $t$ ,  $S(t)$ , be the matching that results from merging  $A(t)$  and  $S(t - 1)$ .

The merging procedure is discussed in chapter 4, where we propose several modifications to the basic self-randomized algorithm. In [19], it is proved that SERENA delivers 100% throughput for Bernoulli arrivals.

### 1.3.5 Fluid Model Techniques

Most of the classical analytical results on stability of the input-buffered scheduling algorithms are based on the stability theorems for Markov processes. Using these theorems, in [37], [29], [31], it is shown that input buffered switches with a maximum weighted matching scheduler can achieve 100% throughput. However these results hold when the arrival process is independent identically distributed (i.i.d.), so that the system evolution can be modelled as a Markov process.

It has been believed that the stability results can be extended for arbitrary arrival process if no ingress or egress port is overloaded. Dai and Prabhakar in [13] used the fluid model techniques to prove this for the first time. In fact, they proved that if number of backlogged cells in each VOQ is used as the weight of the corresponding link, a maximum weighted matching scheduler can achieve 100% throughput for any arbitrary arrival process that satisfies following conditions:

1. It obeys the strong law of large numbers.
2. It does not overload any ingress or egress port.

Results of [13] are derived by considering the fluid model analogs of an input-buffered switch. The framework of fluid models has proved to be powerful in obtaining the maximum throughput region of input-buffered switches, under very mild condition on the input traffic. A general review of the stability analysis of stochastic networks using fluid models is given in [12]. Basically in this framework, in order to prove that a switch delivers 100% throughput it is sufficient to prove that the corresponding fluid model is weakly stable. We will elaborate more on this in chapters 3 and 4.

In chapter 3, we use the fluid model technique to prove that every matching

algorithm in MNCM class achieves 100% throughput. In chapter 4, we use the fluid model technique to prove self-randomized scheduling algorithms are rate stable.

## 1.4 Contributions of Dissertation

### 1.4.1 Packetized Tracking Policies

Fluid and packetized policies are two broad class of scheduling policies that are studied and considered in switching systems. In the fluid policy it is assumed that the link capacity can be divided between the queues and each one of them can use a fraction of the capacity at any time. Contrary, in a packetized policy, at any time, only one of the queues can be served. A fluid policy determines how much of the link capacity should be given to each one of the flows at any time, while a packetized policy determines which one of the queues should be sole user of the link at any time.

One of the main issues in the design of integrated services networks is to provide performance requirements to a broad range of applications. Application requirements are translated into network quantitative parameters. The most common performance measures are packet loss probability, throughput, delay, and jitter. Scheduling algorithm that are used in a switch has a direct impact on its throughput, delay and jitter characteristics. On the other hand, the network should also be capable to analyze the amount of resources that each particular flow requires. It is therefore, very important for the network designer to understand effects a scheduling policy has on the connection performance and on the usage of network resources.

In many cases, it is easier to perform the analysis and design of scheduling

policies under the modelling assumption that the traffic arrives and is treated as a fluid, i.e., the realistic case where information is organized into packets is not taken into account, [10],[11],[32],[33],[17]. Under the fluid policy, we assume that at every time instant arbitrary fractions of the link capacity can be shared among different applications. Although in most of the practical situations this is an idealistic assumption, it enables us to analyze the effect of a scheduling policy on the network resources as well as the major performance parameters, and therefore to design the scheduling policies more conveniently. One approach to the design of packetized policies is to first find an appropriate fluid policy, and then to derive a packetized policy that resembles or “tracks” the fluid policy in a certain sense.

Existence of packetized tracking policies is a well established fact in the single link case. In fact, several tracking policies are suggested and their performance and efficiency are analyzed [32],[33],[17],[14],[21]. Existence of such tracking policies in input queueing switches is studied chapter 2.

In our model, we consider an input queueing switch, where every input and output port can serve 1 cell per time slot. In a fluid policy model, at every time slot every input (output) port can be connected to several output (input) ports, however the total service rate of any port should not exceed its capacity. Under a packetized policy, every input (output) port can at most be connected to one output (input) port at every time slot, i.e., there is no speed up in the switch fabric. Under these circumstances, our objective is to find a packetized policy that tracks a given fluid policy.

A packetized policy tracks a given fluid policy perfectly if under the packetized policy every cell departs the system, at most at the end of the time slot that it departs under the fluid policy. For the special case of  $2 \times 2$  switches, we prove

that the tracking policies always exist and we provide a non-anticipative tracking policy in chapter 2. We should clarify that a scheduling policy is non-anticipative if its decision at any time does not depend on the future arrivals. For the general case, we use a counter-example for a  $3 \times 3$  switch fabric to prove that a perfect tracking policy does not exist. However, a heuristic algorithm with good, but not perfect tracking properties is proposed.

The heuristic algorithm is based on a weighted matching algorithm. The weighted matching algorithms are usually too complex to implement in hardware. Here, we employ two notions to reduce the complexity. The first concept is to do port based weighted matching rather than link weighted matching. Mekkitikul and McKeown [31] used a similar concept. They used the queue length as the weights in their work, and illustrate that they can achieve 100% throughput. Our weights reflect the amount of work by which the fluid policy is ahead of the tracking policy at each port, and our main objective is to be able to provide rate guarantees and high throughput at the same time. The second notion that aids us in improving the performance and reducing the complexity is the concept of critical ports and links. Basically, criticality relates to the urgency by which a port/link needs to be scheduled in order to ensure proper tracking. We detect all critical links and remove all non-critical links that contend with the critical ones. In this way, number of contending links and consequently the complexity of the algorithm is reduced.

### **1.4.2 The Maximum Node Contained Matchings**

In chapter three, we use fluid model techniques to establish some new results for the throughput of input-buffered switches. In particular, we introduce a new class of

deterministic maximal size matching algorithms that achieves 100% throughput. Dai and Prabhakar [13] have shown that any maximal size matching algorithm with speedup of 2 achieves 100% throughput. We introduce a class of maximal size matching algorithms that we call them maximum node containing matching (MNCM) algorithms, and prove that they have 100% throughput with no speedup. We also introduce a new weighted matching algorithm, maximum first matching (MFM) with complexity  $O(N^{2.5})$  that belongs to MNCM. MFM, to the best of our knowledge, is the lowest complexity deterministic algorithm that delivers 100% throughput. The only assumption on the input traffics is that they satisfy the strong law of large numbers. Besides throughput, average delay is the other key performance metric for the input-buffered schedulers. We use simulation results to compare and study the delay performance of MFM.

Stability and throughput of input-buffered switches is a well studied problem. In papers [37], [29] it is proved that maximum weighted matching (MWM) algorithm can achieve 100% throughput. In [29] number of backlogged packets and maximum delay of waiting packets in each VOQ are considered as two potential weight functions. In another work [31], Mekkittikul and McKeown considered the case where weights are associated to the ports rather than links and showed that the proposed algorithm, longest port first (LPF), achieves 100% throughput. Complexity of LPF is also  $O(N^3)$ , even though for practical purpose it seems to be more favorable than MWM [30]. Stability of these algorithms are all proven under the assumption of i.i.d. arrivals.

We extend some of the results of [13], by introducing maximum node containing matching (MNCM) algorithms. MNCM is a new class of maximal size matching scheduling algorithms, that achieves 100% throughput with no speedup. The norm

function that is commonly used for stability analysis is norm 2 ( $\|\cdot\|_2$ ). Here, we use norm infinity ( $\|\cdot\|_\infty$ ) instead of it, and focus on the matching algorithms that function based on that. We prove that these matching schemes achieve 100% throughput too. We also introduce the maximum first matching (MFM) algorithm that is in the MNCM class, and therefore has 100% throughput. The MFM algorithm employs maximum size matching algorithms with  $O(N^{2.5})$  complexity rather than maximum weighted matching algorithms with  $O(N^3)$  complexity.

We also introduce another maximal deterministic matching algorithm with  $O(N^2)$  complexity, the maximal sorted matching (MSM). The MSM algorithm is a trivial generalization of the iLPF algorithm introduced in [31]. MSM is not in MNCM class, but for practical applications, we think that it performs similar to MFM. Even though we were not able to prove that MSM has 100% throughput, due to its similarity to MFM, we think that for all practical purposes it achieves the 100% throughput. This conjecture is in accordance with our simulation results, where delay performance of MSM matches performance of MFM. Since we have used the fluid model technique, the stability results are under very general conditions for arrival process. The only assumption on the input traffic is that it satisfies the strong law of large numbers. Recall that the results of [31] are for i.i.d. arrivals. Therefore, our results are the first stability results for port weighted matching algorithms under general arrival patterns.

For rate provisioning applications, we introduce Maximum Size Unit Interval Matching (MSUIM) algorithm that can provide rate guarantees and is not in MNCM. It is true that any scheduling algorithm in MNCM can also provide rate guarantees since they are rate stable. However, MSUIM is easier to implement and we were able to bound its performance. In fact, it is proved that MSUIM can be

used to track any feasible fluid policy and it lags the fluid policy by no more than  $N$  cells on every node.

### 1.4.3 Self-randomized Scheduling Algorithms and Rate Provisioning

In chapter four, we consider self-randomized scheduling policies for input buffered switches. Randomized scheduling policies are considered for scheduling in input-buffered switches. A scheduling policy is said to be self-randomized if the randomized component (process) of the randomized scheduling policy is replaced by a pseudo-random process that is a function of the cell arrival process. We use fluid model techniques to show that the self-randomized scheduling algorithms deliver 100% throughput. The only assumption on the arrival pattern is that it satisfies strong law of large numbers, and no input or output port is oversubscribed. We provide a general architecture for the design of self-randomized algorithms, and introduce two algorithms that consider number of backlogged cells as the weight function and three algorithms. We then introduce concept of the max-min fair self-randomized scheduling algorithms. The idea here is to introduce self-randomized algorithms that can provide rate guarantees by sharing the switch bandwidth proportional to the assigned weights. We introduce three self-randomized scheduling algorithms, and prove that they are max-min fair. In order to study and compare the performance of the proposed scheduling algorithms, several simulations are carried out and their results are provided and discussed.

In section 1.3.4 we mentioned that paper [19] introduced notion of self-randomized algorithms to enhance performance of the basic randomized algorithm [36]. The idea behind self-randomized scheduling algorithms is instead of selecting a pure

random matching, using the arrival cell process to generate and derive a pseudo-random process.

In chapter four, we focus on the self-randomized scheduling algorithms and introduce some new algorithms that have better delay performance compared to other proposed algorithms. To that end, we introduce a general architecture for the self-randomized schedulers and propose two specific scheduling algorithms that use number of backlogged cells as the weight of the links.

We use fluid model techniques to establish some new results for the throughput of the randomized scheduling algorithms. More specifically, we prove that the original randomized algorithm [36], and all of the self-randomized scheduling algorithms that are introduced in chapter 4 delivers 100% throughput. The only assumptions on the input traffic is that it satisfies SLLN, and is admissible.

Rate provisioning is the other performance measure for scheduling algorithms that is overlooked in the context of randomized scheduling algorithms. Rate provisioning scheduling algorithms are very common for scheduling in output-buffered switches, however for the input-buffered schedulers with no speedup problem is more challenging, and there are few theoretical and practical results [6], [35]. Paper [38] introduces and studies a new token based max-min fair scheduling algorithm. The basic idea behind a max-min fair scheduler is to allocate bandwidth among flows proportional to their weights, and if a flow can not utilize its bandwidth, because of constraint elsewhere in the network, then the residual bandwidth is distributed proportionally among others [22]. Note that this scheduling algorithm is not originally proposed for input-buffered switches, but it is directly applicable to them too. We can categorize this scheduling algorithm as a deterministic MWM scheduling algorithm that weight of links (flows) are number of tokens allocated

to them. Tokens are distributed among links with their max-min rate.

We extend the idea of max-min fair scheduling algorithms and introduce the concept of max-min fair self-randomized scheduling algorithms. Three different self-randomized max-min fair algorithms are introduced. For each algorithm, it is proved that it delivers the max-min rate to all of the links.

## Chapter 2

# Rate Provisioning and Tracking Fluid Policies in Input-buffered Switches

The concept of tracking policies is proven to be a very useful technique in design and analysis of scheduling algorithms for multiple flows sharing a single link [10], [11],[32],[33], [17]. In fact, Weighted Fair Queueing (WFQ) [32], and its extensions such as Self Clock Fair Queueing (SCFQ) [21], and Worst case Fair, Fair Queueing (WF2Q) [2] are tracking policies for Generalized Processor Sharing (GPS) fluid policy. The single link sharing scheduling model is applicable to an output queued switch fabric, where all cells reside in egress line cards and scheduling problem is many to one among flows that share an egress port. However, this is not extendible to input buffered switches, where the scheduling problem is many to many among flows residing in different ingress line cards and targeting arbitrary egress line cards.

We consider the problem of tracking fluid policies by packetized policies and

extend it to input queueing switches. It is considered that the speed up of the switch is one. One of the interesting applications of the tracking policy in TDMA satellite Switches is elaborated. For the special case of  $2 \times 2$  switches it is shown that a tracking non-anticipative policy always exists. It is found that in general non-anticipative policies do not exist for switches with more than 2 input and output ports. For the general case of  $N \times N$  switches a heuristic tracking policy is provided. The heuristic algorithm is based on two notions, port tracking and critical links. These notions can be employed in the derivation of other heuristic tracking policies as well. Simulation results show the usefulness of the heuristic algorithm and the two basic concepts it relies on.

One way for the design of tracking policies is to have Combined Input Output Queueing (CIOQ) switches with limited speed up that matches the output sequence of a purely output queueing switch. In fact, it is shown in [8] that speed up of 2 is sufficient to resemble the output pattern of any output queueing switch. However, the scheduling algorithm proposed to do that is fairly complicated, and the arbiter still requires to receive information from the input ports of the switch with speed up of  $N$ . Here, we consider only switches with no speedup, and look for low complexity algorithms.

In section 2.1, we review the concepts of fluid and tracking policies, and provide the feasibility condition for both cases. The problem of scheduling multi-periodic messages in TDMA-SS is explained and elaborated in section 2.2. It is indicated that this problem is essentially a special case of the input queueing scheduling problem considered in this paper. In section 2.3, we show that for the  $2 \times 2$  switches a tracking policy always exist, and we provide a non-anticipative algorithm to find the tracking policy. In section 2.4, some useful ideas regarding the design of

heuristic tracking policies are given. Based on these concepts a heuristic scheduling algorithm is proposed, and used to implement a fixed rate scheduler. The fix rate scheduler is simulated and used for scheduling in input-buffered switches. Results of the simulation are provided and effect are different parameters are discussed.

## 2.1 Fluid and Packetized Tracking Policies

We consider input queueing switches that serve fixed size cells. Each input and output port has the capacity of serving 1 cell per time unit. Since queues exist only at the input ports, the latter assumption implies that traffic of at most 1 cell per unit of time can be transferred from the input ports to a given output port.

We assume that the time is slotted and the length of each slot is equal to the length of a cell. Slots are numbered starting from 1, 2, ... . Slot  $k$  is taking the time interval  $(k - 1, k]$ . Time  $k - 1$  ( $k$ ) is the beginning (end) of time slot  $k$ . Cells arrive at the beginning of each time slot.

Two broad classes of policies are considered, fluid and packetized policies. During time slot  $k$  a fluid policy transmits,  $w_{ij}(k) \geq 0$ , units of information from input port  $i$  to output port  $j$ .  $w_{ij}(k)$  is a nonnegative real number and is measured in units of cells. Since at most one unit of work can be transferred from a given input port to the output ports, and since no queueing is permitted at the output ports, the  $w_{ij}(k)$ 's must satisfy the following inequalities.

$$\begin{aligned}
 w_{ij}(k) &\geq 0 \\
 \sum_{j=1}^N w_{ij}(k) &\leq 1, \quad \forall i \in \{1, \dots, N\} \\
 \sum_{i=1}^N w_{ij}(k) &\leq 1, \quad \forall j \in \{1, \dots, N\}
 \end{aligned} \tag{2.1}$$

A packetized policy is based on the assumption that during a time slot an input

port can transmit a single cell to any one of the output ports. Therefore, for a packetized policy we have that  $S_{ij}(k)$ , the number of cells transmitted from input port  $i$  to output port  $j$  during slot  $k$ , is either 0 (no cell transmission during slot  $k$ ) or 1 (a single cell transmission during slot  $k$ ). A packetized policy is feasible if at every time slot  $k$  we have,

$$\begin{aligned} \sum_{j=1}^N S_{ij}(k) &\leq 1, \quad \forall i \in \{1, \dots, N\} \\ \sum_{i=1}^N S_{ij}(k) &\leq 1, \quad \forall j \in \{1, \dots, N\} \\ S_{ij}(k) &\in \{0, 1\}. \end{aligned} \tag{2.2}$$

Note that the conditions in (2.2) imply that for any  $k$ , there can be at most a single 1 in each column and row of the matrix  $[S_{ij}(k)]$ . That is, the matrix  $S_{ij}(k)$  is a sub-permutation matrix.

Usually fluid policies cannot be applied directly in a network since mixing of traffic belonging to different cells is not allowed. However, as mentioned in the introduction, they are considered in this paper, because the performance analysis and the scheduling policy design is often more convenient for fluid policies. An approach to the design of packetized policies is to first design and analyze a fluid policy, and then implement a packetized policy that resembles in a certain sense the departure process of the fluid policy. Such a packetized policy is called a tracking policy. More precisely, for our purposes, we use the following definition:

**Definition I:** Given a fluid policy  $\pi_f$ , we say that a packetized policy is *tracking*  $\pi_f$  if every cell departs under the packetized policy at the latest by the end of the time slot at which the same cell departs under the fluid policy.

A basic question is if tracking policies exist for a given fluid policy. This ques-

tion is answered positively for the single link case, where different sessions share a single link [32], [17]. In that case, perhaps the best known fluid policy is the Generalized Processor Sharing (GPS) policy. Several tracking policies are suggested for the single link case [32], [17], [21]. The concept of GPS can be appropriately extended to the multi-input, multi-output input queueing switches. However, the existence of tracking policies for these switches is still an open question. In section 2.3, we study the special case of  $2 \times 2$  switches. We will prove that for the special case of  $2 \times 2$  switches, for every feasible fluid policy there exists a feasible packetized policy. In fact, our proof is constructive and provides an algorithm to derive a tracking policy. Before discussing the  $2 \times 2$  case, in the next section we present the problem of multi-periodic TDMA-SS scheduling and indicate how the problem could be trivially solved by the construction of tracking packetized policies.

## 2.2 Multi-periodic TDMA Satellite Switches

One of the potential applications of tracking policies is in the scheduling of TDMA Satellite Switches (TDMA-SS). The conventional method to do the scheduling is based on the Inukai method [25]. This method is based on the assumption that all messages have the same period. The scheduling is done for a frame length equal to the period of the messages and it is repeated periodically thereafter. Let  $L$  be equal to the maximum number of cells that can be serviced by an input/output port during one period. A set of messages is schedulable if for every port the total number of cells that should be serviced is no more than  $L$ . Inukai provided a scheduling algorithm for any set of schedulable messages.

The Inukai algorithm does not work appropriately when messages have different periods. Let message  $m$  from input port  $s_m$  to output port  $d_m$  have period  $p_m$ .

To apply the Inukai method the frame length should be set to the Least Common Multiplier (LCM) of all message periods, say  $L$ . For each message  $m$ ,  $L/p_m$  unit length cells are scheduled in the frame. Each of these cells is associated to one period of the original message. Then, we can use the Inukai method to allocate these cells inside the frame length  $L$ . The problem is that there is no control over the place of cells inside the frame in the Inukai method. Thus, it is possible that all cells attributed to a single periodic message are placed next to each other. Such an assignment suffers from high jitter. Moreover, the delay of a cell can be equal to  $L$ , which can be very large.

Suppose that the objective is to schedule every cell in the time frame of its period. Thus, every cell can tolerate a delay up to its period. The question then arises whether it is possible to provide a schedule under these constraints. A necessary condition for schedulability, is that the utilization of every input port  $i$  and output port  $j$  should not be greater than unity, i.e.,

$$u_i = \sum_{m:s_m=i} \frac{1}{p_m} \leq 1,$$

$$u_j = \sum_{m:d_m=j} \frac{1}{p_m} \leq 1,$$

If one considers fluid policies, then it is easy to provide a schedule provided that (2.3) is satisfied. Specifically, consider the fluid policy that assigns the fix service rate of  $1/p_m$  to every message  $m$ . Under this policy the switch starts servicing every cell immediately after its arrival and it takes  $p_m$  time units to complete its service. This means that the target deadlines are all accomplished. Therefore, if we can provide a packetized policy that tracks the fluid policy, then this packetized policy will satisfy the delay constraints as well. In [34] Philp and Liu conjectured that (2.3) is the necessary and sufficient condition for schedulability under the

specified delay constraints. Giles and Hajek [20] have proved this conjecture for a special case. In their model the messages are sorted based on their period, such that,

$$p_1 \geq p_2 \geq \dots \geq p_M.$$

Moreover, for every two subsequent messages we have,

$$p_m = kp_{m+1},$$

where  $k$  is an integer. Unfortunately, their algorithm does not work well in the general case. Bonuccelli and Clo [5] presented a counter-example for a  $4 \times 4$  switch, and illustrate that the conjecture is in general not correct. In the following section, we show the existence of tracking policies for the special case of  $2 \times 2$  switches. Thus, the conjecture is proved for the special case of  $2 \times 2$  switches. We will also provide a counter-example for a  $3 \times 3$  switch.

## 2.3 The $2 \times 2$ switch

In this section, we consider a  $2 \times 2$  input queueing switch and provide an algorithm for designing a packetized policy  $\pi$  that tracks a given fluid policy  $\pi_f$ .

We make the following assumption regarding the fluid policy.

**Assumption I:** The fluid policy is non-anticipative (its decisions do not depend on future arrivals) and such that the order in which cells with origin port  $i$  and destination port  $j$  complete service, is not affected by new cell arrivals.

This assumption is similar to the one used in the design of tracking fluid policies in the single server case [17]. Note that the assumption does not preclude

the possibility that new cell arrivals affect the order in which some cells complete service. Consider the following example of a scheduling policy in a  $2 \times 2$  switch (see Figure 2.1). Input port 1 employs a GPS scheduler to schedule cells destined to output ports 1, 2 with weights  $1/4$  and  $3/4$  respectively. The scheduler operates in a work-conserving fashion, serving all eligible cells (i.e., cells that have no transmission constraints at any of the output ports) according to their weights. Output port 2 uses a strict priority scheme to schedule cells from input ports 1 and 2; cells from input port 2 have higher priority. Assume now that at the beginning of time slot 1, two cells  $p_1, p_2$  arrive at input port 1, destined to output ports 1 and 2 respectively. If no new arrival occurs at the beginning of slot 2, then the finishing times of  $p_1$  and  $p_2$  are 2 and  $4/3$  respectively, as shown in Figure 2.1, a). Assume next that at the beginning of time slot 2, cell  $p_3$  arrives at input port 2, destined for output port 2 -see Figure 2.1, b). Then at time slot 2, cell  $p_3$  will be transferred from input port 2 to output port 2, since at output port 2 cell  $p_3$  has higher priority than cell  $p_2$ . Since  $p_2$  cannot be transmitted in slot 2, the GPS scheduler at input port 1 completes transmission of  $p_1$  by time  $7/4$ . The completion time of cell  $p_2$  is now  $9/4$ . We see that the order by which cells  $p_1$  and  $p_2$  complete service is reversed by the arrival of cell  $p_3$ . However, if for cells that are going from the same input port to the same output port we use a policy such as strict priority scheme, then the specified switch scheduling policy satisfies Assumption I.

Examples of fluid policies that satisfy Assumption I are:

- Any nonanticipative fluid policy that serves cells with origin port  $i$  and destination port  $j$  in a First Come First Served (FCFS) manner.
- Any fluid policy that assigns fixed priorities to cells with origin port  $i$  and destination port  $j$

$p_1$  : Destined to Output Port 1  
 $p_2$  : Destined to Output Port 2  
 $p_3$  : Destined to Output Port 2

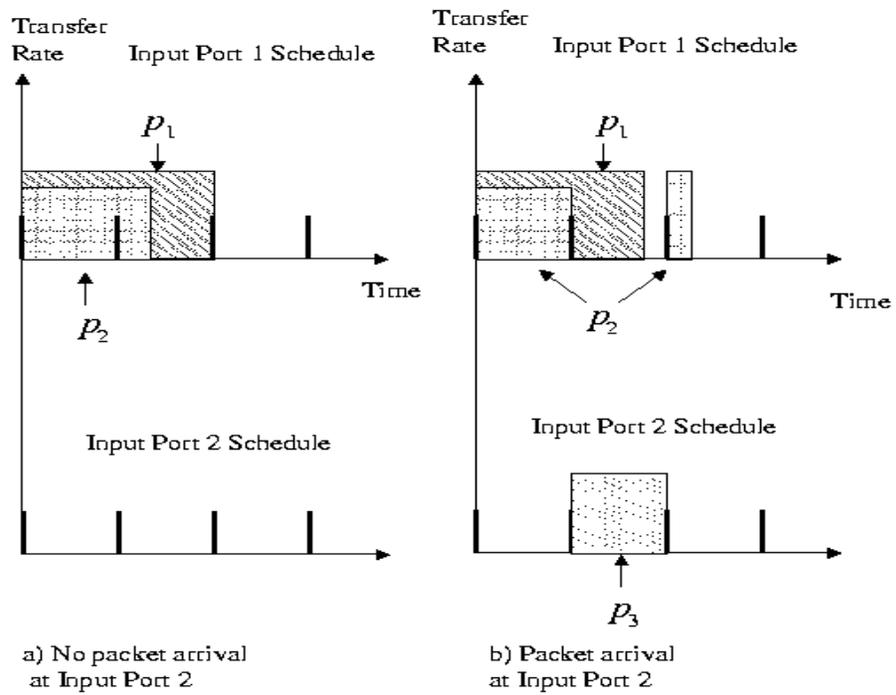


Figure 2.1: Effect of Cell  $p_3$  on completion times of  $p_1$  and  $p_2$ .

- A policy that employs a general non-anticipative fluid scheduler at input port  $i$  to provide transmission intervals to cells destined to different output ports, and another GPS scheduler to schedule, within the provided transmission intervals, cells at port  $i$  destined to a particular output port  $j$ .

The main difficulty in the design of tracking policies for input buffered switches arises from the fact that if one sees an input-output port  $(i, j)$  as a server, this server will not be work-conserving, since it may be forced to idle at some slot  $k$ . This can happen if, for example, during slot  $k$  a cell is transmitted from input port  $i$  to another output port  $j_1$ , instead of the cell destined to output port  $j$ . Below we approach the design of the tracking policy in two steps: First we compute a sequence of sub-permutation matrices  $I_{ij}(k)$  that “tracks” the work performed by the fluid policy for any origin-destination pair. The meaning of tracking in this case is provided below in (2.3). The interpretation of the matrix  $I_{ij}(k)$  is the following. Whenever  $I_{ij}(k) = 1$ , a cell with origin  $i$  and destination  $j$  may be transferred through the switch at slot  $k$  (if there is such a cell in the queue). Next, we provide a packetized policy,  $\pi_p$ , that decides which cells among those using a particular origin-destination pair are to be transmitted at the slots specified by  $I_{ij}(k)$ .

Before we proceed we need the following definition.

**Definition II:**  $\ell_{ij}(k)$  is the largest time slot less than or equal to  $k$ , at the beginning of which there is no work at port  $i$  destined for port  $j$  under  $\pi_f$ .

Let  $I_{ij}(k)$  be a sequence of integer sub-permutation matrices that have the following property.

$$\max_{\ell_{ij}(k) \leq l \leq k} \left\{ \left\lfloor \sum_{s=l}^k w_{ij}(s) \right\rfloor - \sum_{s=l}^k I_{ij}(s) \right\} \leq 0, \quad k = 1, 2, \dots \quad (2.3)$$

Slot	$\ell_{ij}(k)$	$\ell_{ij}(k) + 1$	$\ell_{ij}(k) + 2$	$\ell_{ij}(k) + 3$	$\ell_{ij}(k) + 4$
$w_{ij}(s)$	0	1/2	1/4	3/4	1/2
$\sum_{s=l}^k w_{ij}(s)$	2	2	3/2	5/4	1/2
$\lfloor \sum_{s=l}^k w_{ij}(s) \rfloor$	2	2	1	1	0
$\sum_{s=l}^k I_{ij}^{(1)}(s)$	2	2	2	1	1
$\sum_{s=l}^k I_{ij}^{(2)}(s)$	2	2	1	0	0

Table 2.1: Example of Sub-permutation Matrices

where  $\lfloor x \rfloor$  denotes the integer part of  $x$ . The design of  $I_{ij}(k)$  will be provided later in this section. Consider the example in Figure 2.2 where two sequences,  $I_{ij}^{(1)}(k)$  and  $I_{ij}^{(2)}(k)$  are provided. From Table 2.1 we see that the sequence  $I_{ij}^{(1)}(k)$  satisfies (2.3), while  $I_{ij}^{(2)}(k)$  does not, since

$$\left\lfloor \sum_{s=\ell_{ij}(k)+3}^k w_{ij}(s) \right\rfloor - \sum_{s=\ell_{ij}(k)+3}^k I_{ij}^{(2)}(s) = 1.$$

Note that  $I_{ij}(k)$  specifies whether in a slot there may be a transfer of a cell between ports  $i$  and  $j$  (if  $I_{ij}(k) = 1$  and there is work at input port  $i$  at time  $k - 1$  destined for output port  $j$ ), but it does not specify which cell from the corresponding queue will be chosen for transfer. We now define a packetized policy  $\pi_p$ , that specifies the cell to be chosen for transfer in such a way that the fluid policy  $\pi_f$  is tracked.

**Definition III:**  $\pi_p$  is the packetized policy that whenever  $I_{ij}(k) = 1$  and there is work at input port  $i$  at time  $k - 1$  destined for output port  $j$ , it transfers the cell that completes earliest under  $\pi_f$ .

Provided that  $I_{ij}(k)$  can be designed in a non-anticipative fashion,  $\pi_p$  is also non-anticipative, since by Assumption I, one can decide the order of completion

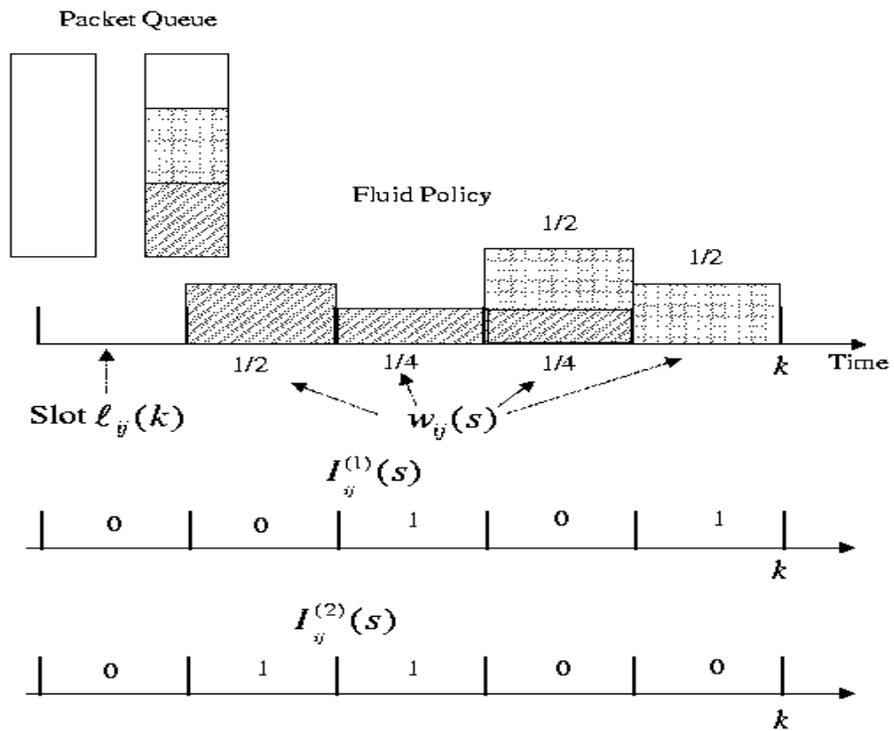


Figure 2.2: Cell transfer from input port  $i$  to output port  $j$ .

times of cells with the same origin-destination pair without knowledge of future arrivals.

Note that  $\pi_p$  acts in the same manner that the tracking policy in the single server case acts [17], except that it first checks, based on  $I_{ij}(k)$ , whether a slot is eligible for transmission of cells with origin-destination pair  $(i, j)$ . On the other hand, the specified policy can be considered as a generalization of the policy for the single-server case. Indeed, specializing to the single-server case, we define  $I_{ij}(k) = 1$  whenever there is backlog in the system at time  $k - 1$  under  $\pi_f$  and  $I_{ij}(k) = 0$  otherwise. Then (2.3) is obviously satisfied, and the resulting policy is identical to the tracking policy for the single server case, proposed in [17].

The next theorem shows that  $\pi_p$  is tracking  $\pi_f$ . In the following it is assumed that the system is empty at time 0.

**Theorem 2.3.1** *Every cell leaves the switch under  $\pi_p$  at the latest by the end of the time slot at which the same cell leaves the switch under  $\pi_f$ .*

*Proof:* Let  $b_m$  be the  $m$ th time slot at the beginning of which there is no work at input port  $i$  destined for output port  $j$  under  $\pi_f$ , while at the end of  $b_m$  there is such work. In other words,  $b_m$  is the beginning of the  $m$ th busy period under  $\pi_f$ , for the server related to the pair  $(i, j)$ . Let also  $e_m$  be the end of the  $m$ th busy period under  $\pi_f$ , that is, the first time slot after  $b_m$  at the beginning of which there is work at input port  $i$  destined for output port  $j$  under  $\pi_f$ , while at the end of  $e_m$  there is no such work. The theorem is true until time slot  $b_1$ . Assume that the theorem is true for all cells that are transmitted up to time slot  $b_m$  under  $\pi_f$ . Notice that then, according to the statement of the theorem it follows that the same cells have been transmitted up to time slot  $b_m$  under  $\pi_p$ .

Next we will show that the theorem holds for all cells that arrive and are

transmitted from time  $b_m$  to  $e_m$ , and therefore, the theorem holds up to time slot  $b_{m+1}$ . Let  $p_n$  be the  $n$ th cell with origin port  $i$  and destination port  $j$  to complete transmission in the interval  $(b_m, e_m]$  under  $\pi_p$ . Let  $f_n$  ( $\widehat{f}_n$ ) be the finishing time of  $p_n$  under  $\pi_p$  (under  $\pi_f$ ). We will show that

$$f_n \leq \lceil \widehat{f}_n \rceil$$

( $\lceil x \rceil$  denotes the smallest integer larger or equal than  $x$ ). For the proof, assuming the contrary, i.e.,

$$f_n > \lceil \widehat{f}_n \rceil, \text{ or } f_n - 1 \geq \lceil \widehat{f}_n \rceil, \quad (2.4)$$

we will arrive at a contradiction.

Consider two cases.

*Case 1.* Suppose that for all cells  $p_l$ ,  $l < n$ , it holds (see Figure 2.3)

$$\widehat{f}_l \leq \widehat{f}_n.$$

Then, cells  $p_1, p_2, \dots, p_{n-1}$  leave before cell  $p_n$  under both  $\pi_p$  and  $\pi_f$ . Let  $\bar{\ell} \geq b_m$  be the last slot before  $f_n$  such that  $I_{ij}(\bar{\ell}) = 1$  and there is no cell to be transferred from port  $i$  to port  $j$  under  $\pi_p$  at time  $\bar{\ell} - 1$ . If there is no such slot, set  $\bar{\ell} = b_m$ . Let also  $p_l, p_{l+1}, \dots, p_n$  be the cells that are transmitted in the interval  $[\bar{\ell}, f_n]$  under  $\pi_p$ . Note that all these cells must have arrived at or after time  $\bar{\ell}$ . This is so since either  $\bar{\ell} = b_m$  and the statement is true by definition, or  $I_{ij}(\bar{\ell}) = 1$ , and therefore, if one of these cells was in the system at the beginning of slot  $\bar{\ell}$ , it would have been transmitted in that slot. Therefore, we have the following important properties

- Cells  $p_l, p_{l+1}, \dots, p_n$  are transmitted in slots  $\bar{\ell} + 1$  to  $f_n$  under  $\pi_p$  and in slot  $\bar{\ell} + 1$  to  $\widehat{f}_n$  under  $\pi_f$ .
- Whenever  $I_{ij}(k) = 1$ ,  $\bar{\ell} + 1 \leq k \leq f_n$ , one of the cells  $p_l, p_{l+1}, \dots, p_n$  is transferred from port  $i$  to port  $j$  under  $\pi_p$ .

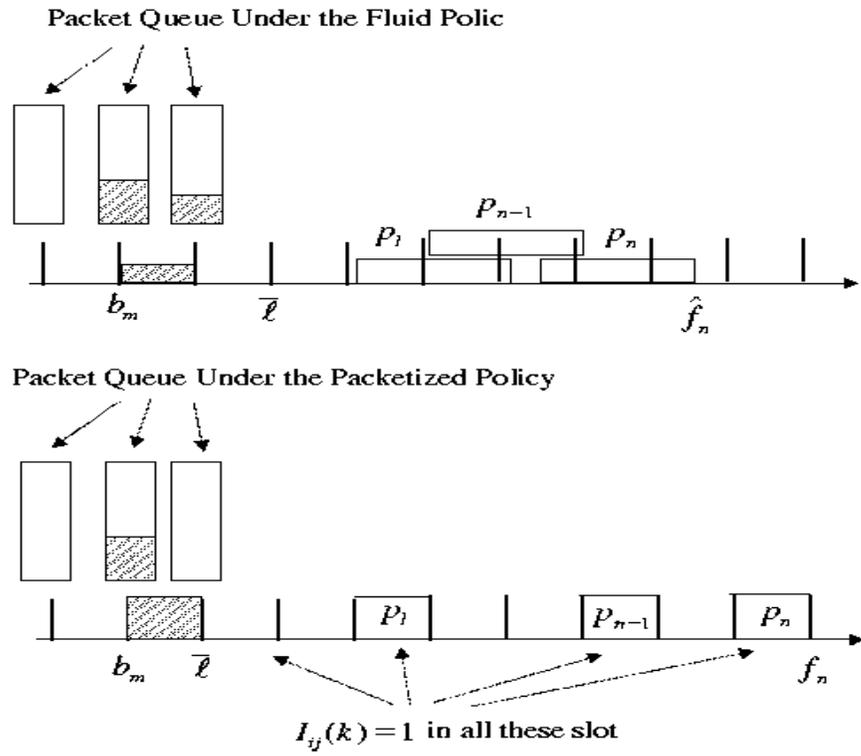


Figure 2.3: Arrangement of cells for Case 1 in the proof of Theorem 1.

Therefore,

$$\sum_{s=\bar{\ell}}^{\lceil \hat{f}_n \rceil} w_{ij}(s) \geq \sum_{s=\bar{\ell}}^{f_n} I_{ij}(s) \quad (2.5a)$$

$$= \sum_{s=\bar{\ell}}^{f_n-1} I_{ij}(s) + 1 \quad (2.5b)$$

$$\geq \sum_{s=\bar{\ell}}^{\lceil \hat{f}_n \rceil} I_{ij}(s) + 1 \quad (2.5c)$$

$$\geq \left\lfloor \sum_{s=\bar{\ell}}^{\lceil \hat{f}_n \rceil} w_{ij}(s) \right\rfloor + 1 \quad (2.5d)$$

Equality (2.5b) follows from the fact that since  $p_n$  was transmitted in slot  $f_n$ ,  $I_{ij}(f_n) = 1$ , inequality (2.5c) follows from (2.4), and inequality (2.5d) from (2.3).

Finally, we have the contradiction,

$$\sum_{s=\bar{\ell}+1}^{\lceil \hat{f}_n \rceil} w_{ij}(s) \geq \left\lfloor \sum_{s=\bar{\ell}+1}^{\lceil \hat{f}_n \rceil} w_{ij}(s) \right\rfloor + 1$$

*Case 2.* Suppose there is a cell  $p_l$ ,  $l < n$  such that

$$\hat{f}_n < \hat{f}_l$$

i.e., cell  $p_l$  leaves earlier than cell  $p_n$  under  $\pi_p$  and later than cell  $p_n$  under  $\pi_f$  (see Figure 2.4). Assume that  $p_l$  is the last cell before  $p_n$  with this property. Then, cells  $p_{l+1}, \dots, p_n$  leave earlier than, or at the same time as, cell  $p_n$  under both policies. This implies that cells  $p_{l+1}, \dots, p_n$  must have arrived at the earliest at the end of slot  $f_l$ . This is so, since otherwise, according to the definition of  $\pi_f$  and Assumption I, if one of the cells  $p_{l+1}, \dots, p_n$  was in the system at the beginning of slot  $f_l$ , it would have been transmitted earlier than  $p_l$  since its finishing time under  $\pi_f$  is smaller

than the finishing time of  $p_l$ . Therefore, we conclude that cells  $p_{l+1}, \dots, p_n$  arrive and are transmitted in the interval  $[f_l, f_n]$  under  $\pi_p$  and in the interval  $[f_l, \widehat{f}_n]$  under  $\pi_f$ . Using again the argument in case 1, we arrive at a contradiction. ■

It remains to show that the matrix  $I_{ij}(k)$  can be constructed in a non-anticipative manner, based on  $\pi_f$ . We will in fact construct a policy that in addition to (2.3) has the following property.

**Property I.** The following inequalities hold for all  $k$ .

$$\begin{aligned} \sum_{j=1}^2 w_{ij}(k) &\leq \sum_{j=1}^2 I_{ij}(k), \quad i = 1, 2 \\ \sum_{i=1}^2 w_{ij}(k) &\leq \sum_{i=1}^2 I_{ij}(k), \quad j = 1, 2. \end{aligned}$$

Define next  $I_{ij}(k)$  recursively as follows

$$I_{ij}(k+1) = \max_{\ell_{ij}(k+1) \leq l \leq k+1} \left\{ \left[ \sum_{s=l}^{k+1} w_{ij}(s) \right] - \sum_{s=l}^k I_{ij}(s), 0 \right\}, \quad k \geq 0, \quad (2.7)$$

where the notation  $\sum_n^m = 0$  when  $n > m$ , is used. Note that since  $\pi_f$  is non-anticipative,  $w_{ij}(k+1)$  can be computed at time  $k$  based only on the past history of the system. Since  $w_{ij}(s), I_{ij}(s), 0 \leq s \leq k$  are also known at time  $k$ ,  $I_{ij}(k+1)$  can indeed be computed at time  $k$ . In the next Lemma we show that the matrix  $I_{ij}(k+1)$  computed using (2.7), with a slight modification, satisfies (2.3).

**Lemma 2.3.2** *At slot  $k+1$ , compute  $I_{ij}(k+1)$  using (2.7). If it turns out that some row or column of  $I_{ij}(k+1)$  contains only zeros, then one of the elements of this row or column can be redefined to one, so that the resulting matrix remains*

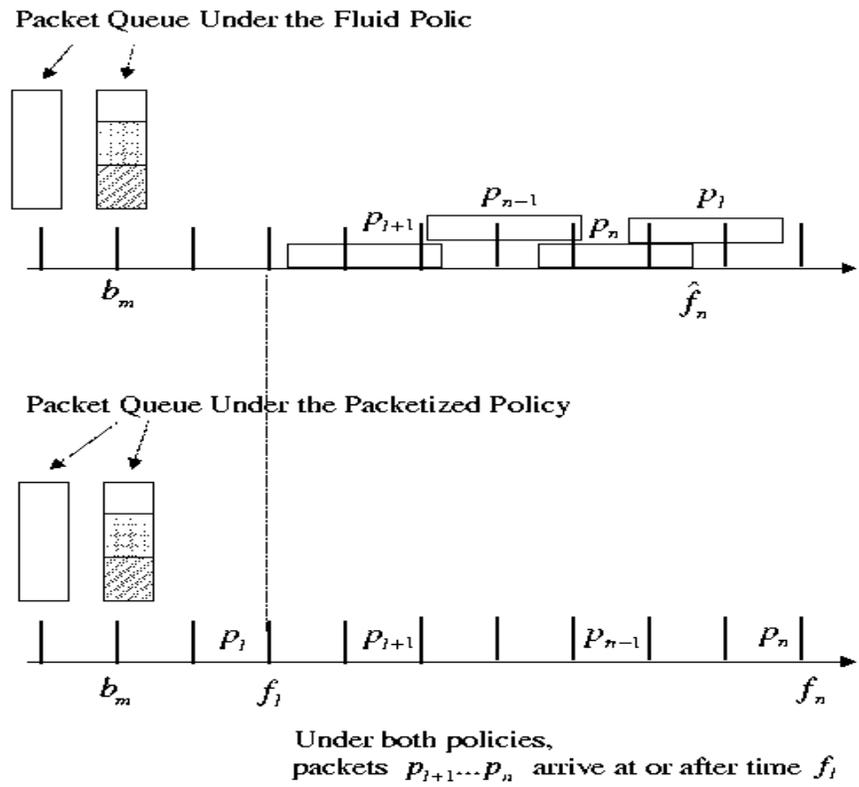


Figure 2.4: Arrangement of cells for Case 2 in the proof of Theorem 1.

a sub-permutation matrix. The matrix  $I_{ij}(k+1)$  so defined, satisfies (2.3) and property I.

*Proof:* Assume  $I_{ij}(k)$  satisfies (2.3) and Property I up to slot  $k$ . We show now that the same holds for slot  $k+1$ . First, we have to show that  $I_{ij}(k+1)$  is an integer sub-permutation matrix, i.e.,  $I_{ij}(k+1)$  takes values 0 or 1, and

$$\sum_{j=1}^2 I_{ij}(k+1) \leq 1, \quad i = 1, 2, \quad \sum_{i=1}^2 I_{ij}(k+1) \leq 1, \quad j = 1, 2. \quad (2.8)$$

To show that  $I_{ij}(k+1)$  takes values 0 or 1, notice that

$$0 \leq I_{ij}(k+1) \quad (2.9a)$$

$$\leq \max_{\ell_{ij}(k+1) \leq l \leq k+1} \left\{ \left| 1 + \sum_{s=l}^k w_{ij}(s) \right| - \sum_{s=l}^k I_{ij}(s), 0 \right\} \quad (2.9b)$$

$$= \max_{\ell_{ij}(k+1) \leq l \leq k+1} \left\{ 1 + \left| \sum_{s=l}^k w_{ij}(s) \right| - \sum_{s=l}^k I_{ij}(s), 0 \right\}. \quad (2.9c)$$

Inequality (2.9b) follows from the fact that  $w_{ij}(s) \leq 1$ . Suppose  $\ell_{ij}(k+1) = \ell_{ij}(k)$ .

Then from (2.3) we conclude that

$$\left| \sum_{s=l}^k w_{ij}(s) \right| - \sum_{s=l}^k I_{ij}(s) \leq 0, \quad \ell_{ij}(k+1) \leq l \leq k$$

and hence

$$\max_{\ell_{ij}(k+1) \leq l \leq k+1} \left\{ 1 + \left| \sum_{s=l}^k w_{ij}(s) \right| - \sum_{s=l}^k I_{ij}(s), 0 \right\} \leq 1, \quad (2.10)$$

Taking into account (2.9c) and the fact that  $I_{ij}(k+1)$  is integer-valued, we conclude that  $I_{ij}(k+1)$  takes the values 0 or 1. Suppose now that  $\ell_{ij}(k+1) \neq \ell_{ij}(k)$ . Then by the definition of  $\ell_{ij}(k)$ , we conclude that  $\ell_{ij}(k+1) = k+1$  and  $w_{ij}(k+1) = 0$ , hence it follows from (2.7) that  $I_{ij}(k+1) = 0$ .

In order to show (2.8) assume that for, say, row 1 we have

$$I_{11}(k+1) = I_{12}(k+1) = 1.$$

Then, be the definition (2.7) there must be  $\ell_1 \geq \ell_{11}(k+1)$  and  $\ell_2 \geq \ell_{12}(k+1)$  such that

$$\begin{aligned} \left\lfloor \sum_{s=\ell_1}^{k+1} w_{11}(s) \right\rfloor - \sum_{s=\ell_1}^k I_{11}(s) &= 1 \\ \left\lfloor \sum_{s=\ell_2}^{k+1} w_{12}(s) \right\rfloor - \sum_{s=\ell_2}^k I_{12}(s) &= 1 \end{aligned}$$

Assume without loss of generality that  $\ell_1 \geq \ell_2$ . Then, adding the previous equations, we have

$$2 \leq \left\lfloor \sum_{j=1}^2 w_{1j}(k+1) + \sum_{s=\ell_1}^k \sum_{j=1}^2 w_{1j}(s) + \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) \right\rfloor - \sum_{s=\ell_1}^k \sum_{j=1}^2 I_{1j}(s) - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s) \quad (2.11a)$$

$$\leq 1 + \left\lfloor \sum_{s=\ell_1}^k \sum_{j=1}^2 w_{1j}(s) + \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) \right\rfloor - \sum_{s=\ell_1}^k \sum_{j=1}^2 I_{1j}(s) - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s). \quad (2.11b)$$

Inequality (2.11a) follows from the fact that  $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor$  and inequality (2.11b) from the fact that  $\sum_{j=1}^2 w_{ij}(k+1) \leq 1$ . Hence, taking also into account that  $I_{ij}(s)$  are integers, we have

$$1 \leq \left\lfloor \sum_{s=\ell_1}^k \sum_{j=1}^2 w_{1j}(s) - \sum_{s=\ell_1}^k \sum_{j=1}^2 I_{1j}(s) + \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s) \right\rfloor.$$

Using Property I we conclude that

$$\sum_{s=\ell_1}^k \sum_{j=1}^2 w_{1j}(s) - \sum_{s=\ell_1}^k \sum_{j=1}^2 I_{1j}(s) \leq 0$$

and hence

$$1 \leq \left\lfloor \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s) \right\rfloor \quad (2.13)$$

If  $\ell_2 = \ell_1$  then  $\left\lfloor \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) \right\rfloor - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s) = 0$ , which contradicts (2.13). If  $\ell_2 < \ell_1$ , then necessarily  $\ell_{12}(k+1) = \ell_{12}(k) \leq \ell_2 < \ell_1$ , and since  $I_{ij}(s)$  satisfies

(2.3) for  $s \leq k$ , we have  $\left[ \sum_{s=\ell_2}^{\ell_1-1} w_{12}(s) \right] - \sum_{s=\ell_2}^{\ell_1-1} I_{12}(s) \leq 0$ , which again contradicts (2.13).

We also need to ensure that  $I_{ij}(k+1)$  satisfies Property I. If  $I_{ij}(k+1) = 1$ , then clearly Property I holds for column  $j$  and row  $i$ . Assume next that for a column or a row, say column 1, it is computed based on (2.7) that

$$I_{11}(k+1) = I_{21}(k+1) = 0.$$

In order to ensure that Property I holds for  $k+1$ , we claim that we can redefine one of  $I_{11}(k+1)$ ,  $I_{21}(k+1)$  to 1 without affecting the sub-modularity property of the matrix. To see this, notice that since  $I_{ij}(k+1)$  takes values 0 or 1 and satisfies (2.8), in column 2 there can be at most a single 1, say in position  $(1, 2)$ . We can therefore set  $I_{21}(k+1) = 1$  in order to ensure that Property I holds for column 1. Proceeding in this way, we can redefine some of the  $I_{ij}(k+1)$  if necessary, in order to ensure that Property I holds. Notice also that with this redefinition, the resulting matrix  $I_{ij}(k+1)$  still satisfies (2.3).

It remains to show that  $I_{ij}(s)$  satisfies (2.3) for  $s = k+1$ . Assume first that  $I_{ij}(k+1)$  has not been redefined. Since by the definition (2.7),

$$I_{ij}(k+1) \geq \left[ \sum_{s=l}^{k+1} w_{ij}(s) \right] - \sum_{s=l}^k I_{ij}(s), \quad \ell_{ij}(k+1) \leq l \leq k+1,$$

we easily conclude that

$$\max_{\ell_{ij}(k+1) \leq l \leq k+1} \left\{ \left[ \sum_{s=l}^{k+1} w_{ij}(s) \right] - \sum_{s=l}^{k+1} I_{ij}(s) \right\} \leq 0$$

If any of the  $I_{ij}(k+1)$  needs to be redefined, this redefinition only increases the value of  $I_{ij}(k+1)$  and hence (2.3) still holds. ■

## 2.4 Heuristic Algorithms

Let  $\pi_f$  be a feasible fluid policy that at every time slot  $k$  specifies the appropriate fluid scheduling matrix  $w(k)$ . We showed in the previous section that as long as  $\pi_f$  satisfies Assumption 1, a non-anticipative tracking packetized policy can be designed for a  $2 \times 2$  switch. Unfortunately, the approach used in the  $2 \times 2$  case does not work for the general case of  $N \times N$  switches. Recall that the validity of relation (2.3) was essential in the proof of Theorem 2.3.1. If we could construct permutation matrices  $I_{ij}(k)$  that satisfy (2.3) for an  $N \times N$  switch, then we could also construct a tracking policy for the general case. However, in Lemma 2.3.2 we heavily used the properties of a  $2 \times 2$  switch in order to prove relation (2.3). In the  $2 \times 2$  case, there are only two possible permutation matrices to consider and the simple construction in (2.7) is sufficient to obtain appropriate permutation matrices  $I_{ij}(k)$ . For  $N \geq 3$ , however, this construction does not always work. In fact, as we will see next, tracking policies do not always exist for  $N \geq 3$ .

Bonuccelli and Claudia present an example [5] for  $4 \times 4$  switches with fixed rate. This example shows that even an anticipative tracking policy may not exist for  $N \geq 4$ . Here we provide a different example for a  $3 \times 3$  switch where a non-anticipative tracking policy does not exist.

**Example :** Consider a  $3 \times 3$  switch. Suppose that the serving discipline of every link under the fluid policy is determined by the number of cells in each buffer and the priority of the cells as follows.

1. If less than one cells are queued in each link, these cells are served with equal rates. If there are links for which more than one cells are queued, all these links - and only these links - are served with equal rate.

2. In each of the links, higher priority cells are served first.

Suppose that cells  $p_1, \dots, p_9$  arrived at the beginning of the first time slot. Buffered cells of the fluid and packetized policy are depicted in Fig. 2.5. The figure shows the buffering at the beginning of first five time slots. In every input there are three parallel buffers (virtual queues) corresponding to the three outputs. As it is illustrated there are 9 cells, one in each virtual queue at time 1. Each of these cells have distinct input/output pairs and all have the same priority. Let matrix  $Q(k)$  specify the buffered cells under the packetized policy at time  $k$ . That is, element  $q_{ij}(k)$  of  $Q(k)$  specifies the set of cells that are buffered at input port  $i$  and are destined for output port  $j$  under the packetized policy. Assume that

$$Q(1) = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{bmatrix}.$$

According to rule 1 above, the service matrices of the fluid policy will be,

$$w(1) = w(2) = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}.$$

Without loss of generality, assume that the tracking policy selects the following two permutation matrices for the first two time slots.

$$S(1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad S(2) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Now assume that at the beginning of time slot 3, six new cells  $p_{10} \dots p_{15}$  arrived, so that the backlogged cells under the packetized policy at the beginning of slot 3

are,

$$Q(3) = \begin{bmatrix} p_{10} & p_{11} & p_3 \\ p_4 & p_{12} & p_{13} \\ p_{14} & p_8 & p_{15} \end{bmatrix}.$$

Hence in the figure at time slot 3, we have one cell in every queue under the packetized policy. The situation is different for the fluid policy,  $1/3$  of the previously arrived cells remain, and there are six new arrivals as well. Assume that  $p_{10}, \dots, p_{15}$  have equal priority, higher than the priority of the previous cells and hence, according to rule 2, they are placed ahead of the previous arrivals in the fluid policy queues. Based on rule 1, the serving rate of the fluid policy becomes,

$$w(3) = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \end{bmatrix}.$$

In order to ensure proper tracking, the tracking policy should also serve a feasible set of these high priority cells. For instance let,

$$S(3) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

At the beginning of time slot 4, assume that six new cells arrive, all with low priority, so that the queue matrix becomes,

$$Q(4) = \begin{bmatrix} 0 & p_{11}, p_{16} & p_3, p_{17} \\ p_4, p_{18} & 0 & p_{13}, p_{19} \\ p_{14}, p_{20} & p_8, p_{21} & 0 \end{bmatrix}.$$

Based on rule 1, the fluid policy would select the following serving rate matrix,

$$w(4) = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \\ 1/2 & 1/2 & 0 \end{bmatrix}.$$

Therefore, at the beginning of time slot 5 six cells, with indices (3,4,8,11,13,14) are fully served under the fluid policy. However, by that time the packetized policy can at most, serve three of them. We assume that the tracking policy serves cells  $p_{11}, p_{13}, p_{14}$ , and there remain cells  $p_3, p_4, p_8$  unserved. Similarly, it can be shown that for any of the other choices of  $J(3)$ , there is another possible set of new arrivals that makes it impossible for the packetized policy to track the fluid policy.

Since as the previous example shows it is impossible to construct non-anticipative policies in the general case, we are motivated to seek for heuristic algorithms with good but not perfect tracking properties. The design of the heuristic relies on two main concepts, port based tracking and critical links, that are discussed below. We design a simple tracking policy based on these concepts and illustrate its performance using simulation results.

### 2.4.1 Port Based Tracking

One way to implement a packetized tracking policy can be based on finding optimal weighted matchings in bipartite graphs. At slot  $k$ , for each input-output port pair  $(i, j)$ , a weight,  $v_{ij}(k)$ , is associated. This weight represents the amount of work on input-output port pair  $(i, j)$ , by which the fluid policy is ahead of the tracking policy, up to slot  $k$ . That is,

$$v_{ij}(k) = \max \left( \sum_{l=1}^k (w_{ij}(l) - S_{ij}(l)), 0 \right).$$

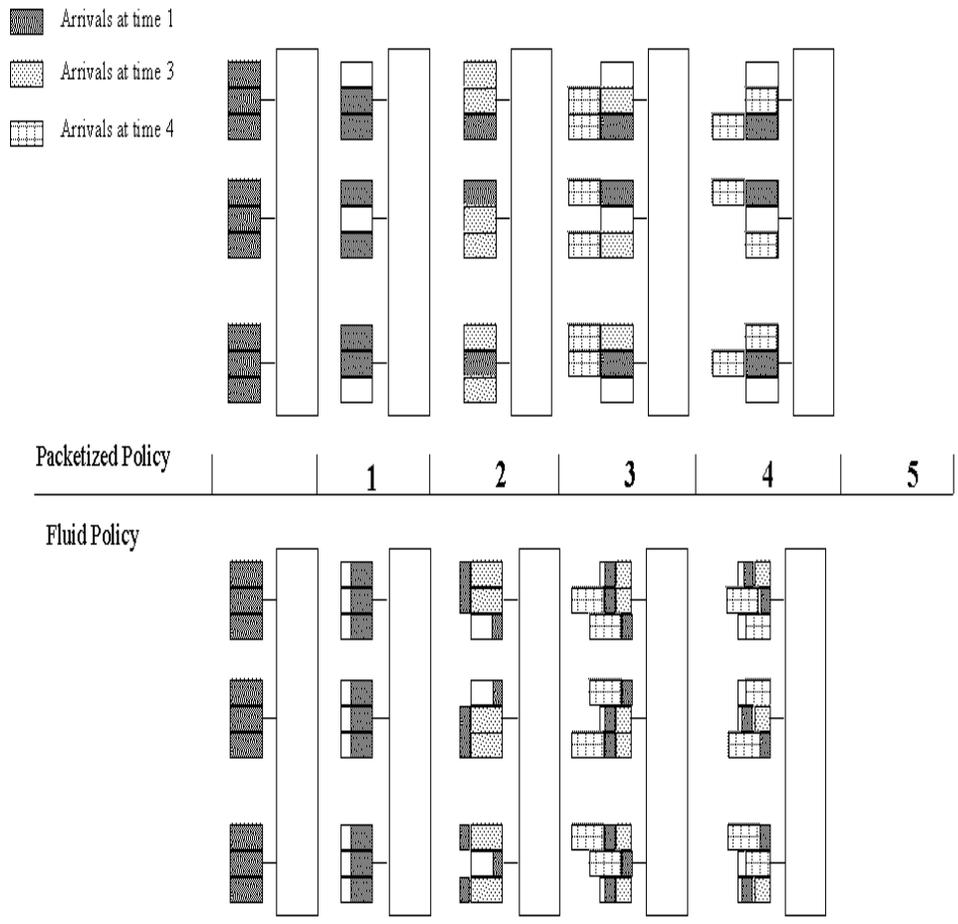


Figure 2.5: The backlogged cells for the fluid and packetized tracking.

We call these weights the tracking weights,  $v_{ij}(k)$ .

We view the switch as a bipartite graph, with nodes the input and output ports of the switch. The weight of link (input-output port pair)  $(i, j)$  is  $v_{ij}(k)$ . The weight of node (port)  $i$ ,  $u_i(k)$ , is the sum of the weights of the links that emanate from or terminate at that node. That is, for an input port  $i$  and an output port  $j$ , we have respectively,

$$u_i(k) = \sum_{j=1}^N v_{ij}(k), \quad (2.14)$$

$$u_j(k) = \sum_{i=1}^N v_{ij}(k). \quad (2.15)$$

Each sub-permutation matrix  $S(k)$  defining the tracking policy at slot  $k$ , corresponds to a matching in the bipartite graph. The selection of the appropriate matching can be based on the link and node weights introduced above. Ideally, the matchings should be chosen so that all these weights remain close to zero.

Bipartite matching algorithms have been extensively used in switch scheduling and they are mostly based on the maximum link (edge) weighted matching or on maximum size matching (that is, a matching that maximizes the number of links included in the matching) algorithms. Maximum link weighted matching algorithms are complex and computationally intensive, while maximum size matching algorithms often have poor performance. We concentrate here on algorithms based on optimal node weight related matchings [31]. As will be seen, algorithms based on these matchings have good performance, and they are simpler for hardware implementation. Two possible solutions are matchings that satisfy the following optimization criteria.

**Maximum node weight sum :** With this criterion, the matching whose node weight sum is maximum is chosen. Hence, at every step it is attempted to

find the matching whose node weight sum “lags” the most from the desired schedule. Since cells will be transmitted on the links of this matching, its “cost” will be reduced in the next time slot.

**Maximum Lexicographic Order of Node Weights:** In this approach, with each matching we assign a binary vector where each digit is associated to one node. Nodes are ordered according to their weight and nodes that are included in the matching are assigned bit one, while nodes not included in the matching are assigned bit 0. We then select the matching whose assigned vector is maximum in the lexicographic order (min-max fair) [4, Section 6.5.2]. In other words, if we consider the bit sequence as binary representation of a number with the bit associated to the greatest weight node to be the MSB, the maximum lexicographic order matching is the matching with greatest binary representation. In essence, we can not add any node to the maximum lexicographic order matching or replace any of its nodes with a node with greater weight. In this approach, it is attempted to select the matching whose nodes have individually large weights and hence are lagging the worst from the desired schedule.

It can be shown that both the above mentioned criteria are equivalent. This is due to the special structure of bipartite graphs. In the next lemma we prove the equivalence of the two criteria.

**Lemma 2.4.1** *Any matching that maximizes the node weight sum, maximizes the lexicographic order of weights and vice versa.*

*Proof:* let  $\overline{M}_1$  be a matching that maximizes the node weight sum and  $\overline{M}_2$  a matching that maximizes the lexicographic order of weights. Let also  $M_1, M_2$

be their respective set of nodes. Without loss of generality, we show that for any input node  $i_0 \in M_1 - M_2$ , there is a node  $i_1 \in M_2 - M_1$ , such that  $u_{i_0}(k) = u_{i_1}(k)$  and vice versa. This implies that the two matchings have the same node weight sum, and that they are equal in the lexicographic order.

Consider the graph  $G$  that consists of links that belong to one and only one of the two matchings. Note that the maximum degree of a vertex in  $G$  is two. Let  $i_0 \in M_1 - M_2$ . Then, degree of node  $i_0$  in  $G$  is one. Therefore, there is an (undirected) path in  $G$  that starts from node  $i_0$  and ends to a node  $i_1$  with degree one. We concentrate on this path.

The number of nodes in this path is either even or odd. If it is even then it follows from the definition of  $G$  that the last node in the path belongs to  $M_1$ , while all intermediate nodes belong to both matching. Thus, if we replace the alternative set of links in the path belonging to  $\overline{M}_2$  with those belonging to  $\overline{M}_1$ , two more vertices will be included to  $\overline{M}_2$ , and this contradicts with the optimality assumption of  $\overline{M}_2$ . Therefore, this case is impossible.

If the number of the nodes in the path is odd, then  $i_1 \in M_2 - M_1$ . In this case, we necessarily have  $u_{i_1}(k) = u_{i_0}(k)$ . Indeed, if  $u_{i_1}(k) < u_{i_0}(k)$  then as in the previous paragraph we can construct a matching that is better in the lexicographic order than  $\overline{M}_2$ . If on the other hand  $u_{i_1}(k) > u_{i_0}(k)$ , then we can similarly construct a matrix that is better than  $\overline{M}_1$  in the node weight sum criterion. ■

We call an optimal matching based on the above criterion, Maximum Node Matching (MNM). Notice that MNM is different from conventional maximum weighted matching, since the latter maximizes the sum of the weights of the links involved in the matching, while MNM maximizes the sum of the weights of the nodes involved in the matching. The difference is illustrated in Fig.2.6. Next, we

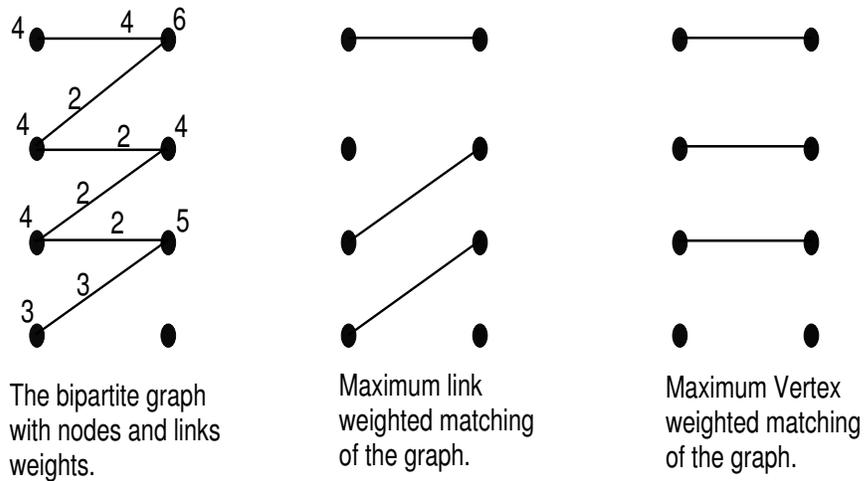


Figure 2.6: Example of node and link weight matching in bipartite graphs.

need to have an algorithm for finding a Maximum Node Matching of a bipartite graph. Note first that MNM is a maximum size matching. To see this, assume that the algorithm employed to find the maximum size matching is the augmented path algorithm for the associated maximum flow problem in an extended network [9].

This algorithm starts with an initial matching (flow on the extended network) and then at each iteration finds a flow augmenting path and a new flow in the extended network. Observe that at each iteration, due to the fact that the graph is bipartite, all the nodes in the original matching are still nodes of the new matching. Hence, assuming that the algorithm uses the MNM matching as its initial matching, the final matching includes all MNM nodes. On the other hand, the final matching cannot include additional nodes, since then its node weight sum would be larger than that of the MNM matching. Therefore, MNM matching should be a maximum size matching.

Assume now that we find a maximum size matching  $\overline{M}_1$ . We show next how

to obtain a MNM,  $\overline{M}^*$ , from  $\overline{M}_1$ . Given any matching  $\overline{M}$ , define an “alternating path” to be a path in the bipartite matching such that any two consecutive links in the path, one belongs to  $\overline{M}$  and the other does not belong to  $\overline{M}$ .

If  $\overline{M}$  is a maximum size matching, then it is easy to see that any alternating path such that one of its endpoints does not belong to  $\overline{M}$ , must contain an even number of links. Otherwise, we have found an augmented path, i.e. number of the links in the path that are not in the matching is one more than number of links that are in the matching. In that case we can replace those link that are in the matching with those that are not, and end up with a larger size matching. Obviously, this contradicts with the assumption that the original matching is maximum size. Hence, the other endpoint of the path belongs to  $\overline{M}$ . The algorithm for finding an MNM is based on the following lemma

**Lemma 2.4.2** *A maximum size matching  $\overline{M}_1$  is an MNM, if and only for any alternating path with endpoints  $i$  belonging to  $\overline{M}_1$  and  $j$  not belonging to  $\overline{M}_1$ , we have*

$$u_i(k) \geq u_j(k).$$

*Proof:* Let  $\overline{M}_1$  be an MNM. If there is an alternating path such that  $u_i(k) < u_j(k)$ , then we can replace the links of  $\overline{M}_1$  in the path with the links in the path not belonging to  $\overline{M}_1$ . The resulting matching will have larger node weight sum than  $\overline{M}_1$ , which contradicts the assumption that  $\overline{M}_1$  is MNM.

Assume now that for any alternating path with endpoint  $i \in \overline{M}_1$  we have  $u_i(k) \geq u_j(k)$ . Let  $\overline{M}^*$  be an MNM matching. Let  $G$  be the graph whose links are the links that belong to one and only one of  $\overline{M}_1$  and  $\overline{M}^*$ . Since both  $\overline{M}^*$  and  $\overline{M}_1$  are maximum matchings, using arguments similar to those used in the proof of Lemma 2.4.1, it can be seen that in  $G$  there can be either alternating cycles or

alternating paths with even number of links. If  $G$  contains only cycles, then  $\overline{M}_1$  has the same nodes as  $\overline{M}^*$  and hence the same node weight sum. Assume now that there is a path in  $G$  such that one of its end nodes,  $i$ , belongs to  $\overline{M}_1$  and the other,  $j$ , in  $\overline{M}^*$ . Since  $\overline{M}^*$  is MNM, we must have  $u_i(k) \leq u_j(k)$ . Since for any alternating path we have  $u_i(k) \geq u_j(k)$ , we conclude that  $u_i(k) = u_j(k)$ . We conclude that the node weights of  $\overline{M}_1$  are equal to the node weights of  $\overline{M}^*$  in the lexicographic order and hence they again have the same node weight sum. ■

Based on Lemma 2.4.2 we have the following algorithm for finding an MNM matching.

**Matching Algorithm:**

1. Find an initial maximum size matching  $\overline{M}_1$ .
2. Mark all vertices not in  $M_1$  as unexplored.
3. Pick one unexplored node  $i$ , and Search the graph for all possible endpoints  $j$  (belonging necessarily to  $\overline{M}_1$ ) of alternating paths originated from  $i$ . Let  $j_{\min}$  has minimum node weight among all discovered end points.
4. If  $u_{j_{\min}} < u_i$ , then replace in  $\overline{M}_1$ , the links in the path belonging to  $\overline{M}_1$  with the links in the path not belonging to  $\overline{M}_1$ .
5. Remove node  $i$  from the unexplored set.
6. If the set of unexplored nodes is empty, then  $\overline{M}_1$  is an MNM and process ends. Else, go to step 3.

The algorithm given above is iterative. There would be at most  $2N$  iterations since one node will be explored in each iteration. The complexity of each iteration

is  $N^2$ , since we have to search the graph and there are at most  $N^2$  links in the graph. Therefore, complexity of this algorithm is  $O(N^3)$ .

So far we have assumed that all lagging links (those with positive weights) are included in the bipartite graph, and that the weight of every node is the sum of weights of the links that emanate from the node. In the next section, we will discuss an enhancing mechanism that enables us to modify the weight of nodes and to exclude some of the graph links so that the matching algorithm can come up with better assignment configurations.

### 2.4.2 Critical Ports and Links

A critical port is a port for which a cell should be scheduled in the next time slot, in order not to miss a deadline in the future. As an example suppose that we are at the beginning of  $k$ th time slot. Assume that there are two cells, one that needs to be transmitted from node  $i$  to node  $j_1$  and the other from node  $i$  to node  $j_2$ . Assume also that both have deadline  $k + 2$ . Note that if we do not schedule any of these cells, no deadline will be missed in the  $k$ th time slot. However, we will definitely miss a deadline at the subsequent time slot,  $k + 1$ . We say that node  $i$  is a *critical node*, and links  $(i, j_1)$  and  $(i, j_2)$  are associated *critical links*. In general, a sufficient condition for a port to be critical at time  $k$  is to have at least  $p$  cells with deadlines less than or equal to  $k + p$ . Note that we are stating a sufficient condition. In other words, there might be some critical ports that cannot be detected well in advance using this criterion. However, for simplicity we concentrate on nodes that are critical according to the criterion defined above. Our goal is to detect critical nodes and increase their chance to be scheduled.

In case of the tracking policy, the deadlines of cells are implicitly given, and

are equal to the end of the time slot that the cell departs the switch under the fluid policy. We may not know the deadlines in advance, since the future rate of every link under the fluid policy depends on the future arrivals, which in general are not known. Nevertheless, we may have an approximate deadline for every cell based on back-logged traffic or the average arrival rate of the links

The next issue is to set an appropriate inspection horizon. Suppose that we are at time  $k$ . To detect the critical links, we have to account for the cells that should be scheduled in the next  $p$  time slots. We call  $p$  the “inspection horizon”. There is a trade-off involved here: increasing the inspection horizon helps us in detecting more critical links, but it increases the complexity of the algorithm as well.

After detecting a critical port, we know that we have to schedule one of the critical links associated with that port, otherwise we will miss the deadlines. To give priority to this node, we increase its weight by a constant, so that its weight exceeds all non-critical nodes weights. Therefore, these nodes are prioritized by the scheduler. To make sure that one of its critical links are scheduled, we remove all non-critical links that have a critical node as an endpoint. The algorithm for detecting critical nodes can be described as follows,

**Critical Node Detecting Algorithm:**

1. At every time step  $k$ , set  $p = 1$ .
2. For each node,  $i$ , find the number of cells that have to be sent in the next  $p$  time slots. These are the cells with deadlines at most  $k + p$ .
3. If for some node,  $i$ , the number of cells that should be sent in the next  $p$  time steps is larger than or equal to  $p$ , then  $i$  is critical. Moreover, the links emanating from  $i$ , over which at least a cell should be sent in the next  $p$  time

slots are critical links

4. Increment  $p$  and go back to step 2, if  $p \leq L_{\max}$  ( $L_{\max}$  is the inspection horizon).

The computationally expensive part of this algorithm is step 2. At that step we need to estimate the work done by the fluid policy in the next  $p$  time slots. In general the estimate depends on future arrivals and cannot be computed exactly. An approximate value can be obtained by assuming that there are no new arrivals in the system. There are cases of course, as in the example described in Section 2.2, where the design is done off-line and future arrival can be anticipated.

The scheduling process of a switch can be divided into two stages. In the first stage, the weight of the ports are calculated and the criticality of the ports is investigated. Once the service rates of the fluid policy are computed, the rest of computations for different ports of the switch can be done in parallel, and no interaction between them is necessary. In the next stage the computed weights for the nodes and the eligible links for every node are provided to the matching algorithm.

Finally we provide the scheduling algorithm that is based on the algorithms described above.

**Scheduling Algorithm:**

1. At every time slot  $k$  do the following steps.
2. Calculate node weights using (2.14), (2.15).
3. Insert all links with positive weights in the eligible links set.
4. Check for critical nodes and their associated critical links.

5. Increase the weight of every critical node to a value,  $C$ , where  $C$  is larger than the weights of all non-critical nodes. Remove all non-critical links of the critical nodes from the eligible links set.
6. Find the maximum node weighted matching. The result of the matching is the schedule for time slot  $k$ .

If there are several sessions that are transmitting cells in an input/output pair, the scheduling algorithm does not specify, which of them should be scheduled in the associated slot. The scheduler considers all these sessions as an aggregated session and works with the aggregate rate. Once a slot is assigned to an input/output pair, it is the responsibility of a local scheduler maintained at the input port to assign the space to one of the multiple sessions. In principle, any single link sharing algorithm may be used as local scheduler. Here we will use EDF scheduler. This hierarchal approach improves the scalability of the scheduler. Note that apart from the computations that may be needed to calculate rates under the fluid policy, the complexity of the scheduler does not depend on the number of sessions between every input/output pair, since all of them are considered as an aggregated session.

The proposed scheduling algorithm is used in next section to schedule fixed rate sessions, and its performance is evaluated through simulations.

### 2.4.3 Fix Rate Scheduler Simulation

We consider multiple fixed rate sessions arrive to all input ports of a switch. These sessions are similar to the periodic sessions introduced in Section 2.2. Each session  $m$ , has an integer period  $p_m$ . In any interval  $[kp_m, (k + 1)p_m - 1]$  one slot should be assigned to session  $m$ . If the slot is not assigned, we assume that one cell of that session is discarded. In effect, we are assuming that real time sessions

have strict deadlines, but can tolerate some cell loss. Moreover, the total capacity dedicated to real time sessions is less than the capacity of the switch. This does not necessarily mean that some of the capacity is wasted, since the remaining capacity could be dedicated to non-real time traffic. In fact, as will be seen, the capacity we considered allocated to real-time traffic in our simulations is much higher than the capacity normally assigned in today's networks.

The main input parameters to the simulation are the maximum port utilization and minimum overall utilization of the switch ports ,  $u_M$  and  $u_m$  respectively, and the switch size  $N$ . We denote the first two parameters as the utilization pair,  $(u_M, u_m)$ . In the first set of experiments, the inspection horizon  $L_{\max}$  is considered as an input and its effect is studied, while in the rest of experiments it is set to a constant value.

The sessions are generated as follows. A uniform random number generator is used to select the input and output ports for every session. The rate of a session is selected uniformly in the range of  $[1/1024, 68/1024]$ , so that the period of sessions is from 15 to 1024 slots. If the selected rate is such that one of the port load exceeds the maximum port rate, then the rate is reduced so that the overall load of that port equals the maximum load. The period of the session is then set to the ceiling of the inverse of the resulting rate. The above process is repeated 10000 times. This does not mean that there are 10000 sessions in each session set, since in some of the attempts either the input or the output port are fully loaded. Once set of sessions is generated, if the resulting average utilization of the switch ports exceeds  $u_m$  it is accepted, otherwise it is discarded and another session set is formed. The minimum session rate is set to  $1/1024$ , because for each session set, the simulation runs for 1024 time steps.

One of the main advantages of the heuristic algorithm is that its complexity is not a function of the number of sessions. The rate of all individual sessions with same input and output are added up and the arbiter looks at them as an aggregated session. Once a slot is assigned to a link, then there is a local scheduler that selects the session that is going to use that slot. In our case, we simply use an EDF scheduler for this purpose. If no slot is assigned to a session during one of its period interval, we assume that one of its cells is discarded. To study the effect of different parameters several experiments are carried out. Each experiment is specified by the values selected for switch size, utilization pair, and the inspection horizon. For each experiment 100 sets of sessions are generated, and for each session set 1024 time steps of simulation is performed. For every session, the percentage of discarded cells is calculated. The performance measure is the percentage of sessions with no discarded cell (0% loss ratio), and the percentage of sessions with 10% loss ratio. Three different aspects of the algorithm are studied, inspection horizon, switch size, and the utilization pair.

### **Inspection Horizon**

We introduced the concept of critical links as a way to detect and increase the chance of the links and ports that are more urgent to be scheduled. Obviously this increases the complexity of the scheduling algorithm. In fact, the additional computation load is a function of the selected inspection horizon. In the first series of experiments, we study the effectiveness of this procedure and the appropriate values for inspection horizon. The switch size is set to 32 and the utilization pair to (0.85, 0.8). The results are given in table 2.2. We can deduce that the detection of critical links can improve the capacity, and reduce the percentage of non-perfectly

$L_{\max}$	0% Ratio	10% Ratio
0	0.9322	0.9996
1	0.9732	0.9997
2	0.9786	0.9997
3	0.9797	0.9997
4	0.9803	0.9997
5	0.9800	0.9997

Table 2.2: Performance for different inspection horizons ( $U=(0.85, 0.8)$ ,  $N=32$ )

scheduled sessions by about 5%. Notice that for  $L_{max} = 0$  (no check), 0.068 of sessions have discarded cells, while for  $L_{max} = 1$ , this reduces to 0.027.

### Switch size

In this series of experiments, the inspection horizon is fixed to 5, and the utilization pair is (0.85, 0.8). Most of the heuristic algorithms provided for input queueing switches fail to give satisfactory result for moderate size switches [34]. The results of our simulation are given in table 2.3. We also observe some degree of performance degradation as the switch size increases. However, in all cases the 0% ratio is around 0.98. In fact, if we decrease the network load, we can even get better results. This is a very important feature of the algorithm, since it is vital for the algorithm to perform well for larger switch sizes.

### Utilization

In this series of experiments the effect of utilization or switch load is investigated. The switch size is set to 32, which is a moderate size switch. The results are given

<b>N</b>	<b>0% Ratio</b>	<b>10% Ratio</b>
8	0.99	0.9999
16	0.98	0.9997
32	0.98	0.9997
64	0.98	0.9997

Table 2.3: Performance for different switch sizes ( $U=(0.85, 0.8)$ )

<b>U</b>	<b>0% Ratio</b>	<b>10% Ratio</b>
(0.55, 0.5)	0.996	1
(0.65, 0.6)	0.993	0.9999
(0.75, 0.7)	0.989	0.9999
(0.85, 0.8)	0.980	0.9997
(0.95, 0.9)	0.952	0.9986

Table 2.4: Performance for different utility pairs ( $N=32$ )

in table 2.4. As we expect the performance of the system degrades as a function of utilization. However, with the exception of the utilization pair (0.95, 0.9), which is very high for a realistic system, the percentage of sessions without any cell loss is above 98%.

## 2.5 Summary and Conclusion

In this chapter, the notion of fluid policies and tracking policies are extended to the  $N \times N$  switches. These concepts are useful in the design of high speed input queued switches, where they can aid in the development of scheduling policies that provide guaranteed service to different applications. The existence of an ideal

tracking policy is proved for the special case of  $2 \times 2$  switches. For the general case, it is shown that perfect tracking policies do not exist, however a heuristic tracking policy is provided.

The design of tracking policies for a general  $N \times N$  switch is still an open question. The examples in Section 2.4 show that such a tracking policies cannot be designed without further constraints on the arrivals or on the policies themselves. This fact, together with the complexity that an ideal tracking policy entails, justify the need for less complicated heuristic tracking policies with good performance. The proposed heuristic algorithm is based on two useful notions, the Maximum Node Matching, and Critical Nodes. The scheduling is done in a hierarchical fashion. First the global scheduler selects the input-output pairs on which cells may be transmitted in a particular slot, and then a local scheduler assigns the slot to one of the sessions sharing the input-output pair. This approach makes the scheduler scalable in terms of the number of sessions. The simulation results are promising and illustrate that the algorithm can be useful in high speed networks and satellite switches where not only throughput but delay and jitter guarantees are desirable too.

## Chapter 3

# MNCM: A class of efficient maximal size matching scheduling algorithms with no speedup

In this chapter, we use the fluid model technique to establish some new results for the throughput of the input-buffered switches. In particular, we introduce a new class of deterministic maximal size matching algorithms that achieves 100% throughput. Dai and Prabhakar [13] have shown that any maximal size matching algorithm with speedup of 2 achieves 100% throughput. We introduce a class of maximal size matching algorithms that we call them the maximum node containing matching (MNCM) algorithms, and prove that they have 100% throughput with no speedup. We also introduce a new weighted matching algorithm, maximum first matching (MFM) with complexity  $O(N^{2.5})$  that belongs to MNCM. The MFM algorithm, to the best of our knowledge, is the lowest complexity deterministic algorithm that delivers 100% throughput. The only assumption on the input traffic is that it satisfies the strong law of large numbers.

Stability and throughput of input-buffered switches is a well studied problem. In the papers [37], [29] it is proved that the maximum weighted matching (MWM) algorithm can achieve 100% throughput. In [29] number of backlogged packets and maximum delay of waiting packets in each VOQ are considered as two potential weight functions. In another work [31], Mekkittikul and McKeown considered the case where weights are associated to the ports rather than the links and showed that the proposed algorithm, longest port first (LPF), achieves 100% throughput. Complexity of the LPF algorithm is also  $O(N^3)$ , even though for practical purpose it seems to be more favorable than the MWM [30]. Stability of these algorithms are all proven under the assumption of i.i.d. arrivals.

In chapter 2 we introduced MNM, a heuristic packetized tracking policy. In essence, the link weight function that we used there specifies how much the packetized policy is behind the fluid policy in serving a link. Intuitively, we can view the serving rate of the fluid policy as the virtual arrival process, and the weight function as the virtual backlog of the VOQs if we use the packetized tracking policy. Therefore, MNM is similar to the LPF algorithm; the only difference is MNM uses virtual backlog as the weight function and the LPF uses the real number of backlogged cells as the weight function.

In order to prove that a packetized policy based on the MNM algorithm can effectively track a fluid policy, we have to show that the virtual backlog process is stable and remain bounded. Recall that virtual backlog process measures how much the packetized policy lags the fluid policy. If the virtual backlog function is bounded, i.e. the MNM is rate stable, then the packetized policy is effectively tracking the fluid policy. The stability result of [31] is not applicable under these conditions, since those results are for Bernoulli i.i.d arrivals, where as virtual arrival

process is by definition not an i.i.d. process.

Dai and Prabhakar [13] have used the fluid model technique to prove that the maximum link weighted matching achieves 100% throughput for a very general set of input traffic patterns. The only assumption on the input traffic is that it satisfies the strong law of large numbers and it does not over-subscribe any input or output port. This result motivates us to use the same technique for the LPF and MNM algorithms.

In this chapter, we extend some of the results of [13], by proving that the LPF algorithm is also rate stable. In fact, our result is more general than that, and we introduce the maximum node containing matching (MNCM) algorithms. The MNCM is a new class of maximal size matching scheduling algorithms, that achieves 100% throughput with no speedup. The LPF and MNM algorithms are both in the MNCM. The norm function that is commonly used for the stability analysis is the norm 2 ( $\|\cdot\|_2$ ). In our stability proof for the MNCM, we use norm infinity ( $\|\cdot\|_\infty$ ) instead of it, and focus on the matching algorithms that function based on that. We also introduce the maximum first matching (MFM) algorithm that is in the MNCM class, and therefore has 100% throughput. The MFM algorithm employs the maximum size matching algorithms with  $O(N^{2.5})$  complexity rather than the maximum weighted matching algorithms with  $O(N^3)$  complexity. Recall that both MNM and LPF have  $O(N^3)$  complexity.

We also introduce another maximal deterministic matching algorithm with  $O(N^2)$  complexity, the maximal sorted matching (MSM). MSM is basically a trivial generalization of the iLPF algorithm introduced in [31]. MSM is not in the MNCM class, but for practical applications, we think that it performs similar to the MFM. Even though we were not able to prove that the MSM has 100% throughput, due

to its similarity to MFM, we think that for all practical purposes it achieves the 100% throughput. This conjecture is in accordance with our simulation results, where delay performance of the MSM matches performance of the MFM.

For the specific application of rate provisioning, we introduce the Maximum Size Unit Interval Matching (MSUIM) algorithm that can provide rate guarantees to all connections in a switch fabric. MSUIM is not in the MNCM class but it is based on the same concepts. In fact, we will show that the number of cells served under the MSUIM is at most  $N$  units behind an arbitrary feasible fluid policy.

This chapter is organized as follow. In section 3.1, we introduce our notation and model. In section 3.2, we review the link based model of the switch, and extend the link based fluid model proposed in [13] to a port based model. In section 3.3, we prove the main result of the paper, which is the stability of the MNCM class of matching. In section 3.4, we present the MFM and MSM algorithms and elaborate on their complexity. In section 3.5, we introduce the MSUIM algorithm and prove that it can provide rate guarantees. We conclude this chapter in section 3.6 with some simulation results, that demonstrate the delay performance of proposed matching algorithms.

### 3.1 Model and Definition

We consider input queued switches that serve fixed size packets (cells). Each input and output has the capacity of serving 1 cell per unit time. Since queues exist only at the input ports, the latter assumption implies that traffic of at most 1 cell per unit time can be transferred from the input ports to a given output port. To avoid HOL blocking, we consider that the buffer at an input is partitioned into  $N$  virtual output queues. The scheduling policy is basically a matching algorithm  $m$

that based on the state of the switch selects a matching between the inputs and outputs in every time slot. If input  $i$  is matched to output  $j$ , and the corresponding VOQ is not empty, a packet is sent from input  $i$  to output  $j$ . A matching can be represented by a permutation matrix  $\pi$ . Input ports are represented by the rows and output ports by the columns of this matrix, therefore input  $i$  is matched to output  $j$  if and only if  $\pi_{ij} = 1$ .

We assume that the cells arrive at the switch at the beginning of a time slot, and they depart the switch at the end of a time slot. A packet that has arrived at the beginning of time slot  $n$  can be scheduled at the same time slot and depart the switch at the end of time slot  $n$ . Let  $A_{ij}(n)$  be the number of packets that arrived at input  $i$  and are destined for output  $j$  up to time  $n$ . We assume that there are no arrivals before time 0, i.e.,  $A_{ij}(0) = 0$ . The arrival processes  $\{A_{ij}(\cdot), i, j = 1, \dots, N\}$  satisfy strong law of large numbers, that is with probability one,

$$\lim_{n \rightarrow \infty} \frac{A_{ij}(n)}{n} = \lambda_{ij} \quad i, j = 1, \dots, N. \quad (3.1)$$

$\lambda_{ij}$  is the arrival rate of cells destined from input  $i$  to output  $j$ . Similarly, we show the number of departed cells up to time  $n$ , from input  $i$  to output  $j$  with  $D_{ij}(n)$ . We consider the following definition for stability.

**Definition 3.1.1** *A switch operating under a scheduling algorithm is rate stable if, with probability one*

$$\lim_{n \rightarrow \infty} \frac{D_{ij}(n)}{n} = \lambda_{ij} \quad i, j = 1, \dots, N, \quad (3.2)$$

*for any arrival process that satisfies (3.1).*

**Definition 3.1.2** *A scheduling algorithm is efficient if (3.2) holds for any arrival*

process satisfying the feasibility conditions,

$$\begin{aligned} \sum_{i=1}^N \lambda_{ij} &\leq 1, \\ \sum_{j=1}^N \lambda_{ij} &\leq 1. \end{aligned} \tag{3.3}$$

Let  $Z_{ij}(n)$  be the number of backlogged packets in  $VOQ_{ij}$  at time  $n$ , hence

$$Z_{ij}(n) = A_{ij}(n) - D_{ij}(n). \tag{3.4}$$

We will introduce and analyze some scheduling algorithms that work based on the port level parameters. Port level parameters are defined to be aggregate (summation) of their corresponding link level parameters. We number the ports of the switch from 1 to  $2N$  where index sets  $\{1, \dots, N\}$ , and  $\{N+1, \dots, 2N\}$  corresponds to input and output ports respectively. Let  $(i, j)$ ,  $i, j \in \{1, \dots, N\}$ , indicates one of the switch links and  $k \in \{1, \dots, 2N\}$  one of the switch ports. We say link  $(i, j)$  goes to port  $k$  and show it as  $(i, j) \rightarrow k$ , if  $k$  and either  $i$  or  $j$  are associated to the same physical port, i.e. either  $k = i$  or  $k = j + N$ . We define the port arrival process  $H(n) = \{H_k(n), k = 1, \dots, 2N\}$ , departure process  $E(n) = \{E_k(n), k = 1, \dots, 2N\}$ , the backlogged process  $B(n) = \{B_k(n), k = 1, \dots, 2N\}$ , and the port arrival rate  $r = \{r_k, k = 1, \dots, 2N\}$ ,

$$\begin{aligned} H_k(n) &= \sum_{(i,j):(i,j) \rightarrow k} A_{ij}(n), \\ E_k(n) &= \sum_{(i,j):(i,j) \rightarrow k} D_{ij}(n), \\ B_k(n) &= \sum_{(i,j):(i,j) \rightarrow k} Z_{ij}(n), \\ r_k &= \sum_{(i,j):(i,j) \rightarrow k} \lambda_{ij}. \end{aligned} \tag{3.5}$$

In the next section, we provide the equations that govern evolution of the port level parameters.

## 3.2 Link and Port Fluid Models

The fluid model for the links of an input buffered switch is given in [13]. We first review those relations and then extend them to derive the port based fluid model. Consider an input-buffered switch that employs scheduling algorithm  $m$ . Suppose that  $T_\pi^m(n)$  be the total time that permutation matrix  $\pi$  is used up to time  $n$ . The following equations describe the link level discrete dynamic of the switch, for  $n \geq 0$ , and  $i, j = 1, \dots, N$ ,

$$Z_{ij}(n) = Z_{ij}(0) + A_{ij}(n) - D_{ij}(n),$$

$$D_{ij}(n) = \sum_{\pi \in \Pi} \sum_{l=1}^n \pi_{ij} 1_{\{Z_{ij}(l) > 0\}} (T_\pi^m(l) - T_\pi^m(l-1)), \quad (3.6)$$

$$\sum_{\pi \in \Pi} T_\pi^m(n) = n,$$

where  $\Pi$  is the set of all  $N \times N$  permutation matrices. The first equation describes the basic relation between arrival, departure and backloged process. The second equation counts the number of total departures from input  $i$  to output  $j$  by counting number of times that a permutation matrix with a one in  $(i, j)$  position is used, while there were a backloged packet in the corresponding VOQ.

The port dynamics of a switch can be derived from link dynamics. Every port

$k = 1, \dots, 2N$ , dynamic is basically summation of its corresponding link dynamics,

$$B_k(n) = B_k(0) + H_k(n) - E_k(n),$$

$$E_k(n) = \sum_{\pi \in \Pi} \sum_{l=1}^n \sum_{(i,j):(i,j) \rightarrow k}^N (\pi_{ij} 1_{\{Z_{ij}(l) > 0\}} (T_{\pi}^m(l) - T_{\pi}^m(l-1))), \quad (3.7)$$

$$\sum_{\pi \in \Pi} T_{\pi}^m(n) = n.$$

Now we describe a deterministic continuous fluid model of a switch operating under some matching algorithm  $m$ . The link level fluid model that is used in [13] is,

$$Z_{ij}(t) = Z_{ij}(0) + \lambda_{ij}t - D_{ij}(t) \geq 0,$$

$$\dot{D}_{ij}(t) = \sum_{\pi \in \Pi} \pi_{ij} \dot{T}_{\pi}^m(t) \geq 0, \quad \text{if } Z_{ij}(t) \geq 0, \quad (3.8)$$

$$\sum_{\pi \in \Pi} T_{\pi}^m(t) = t.$$

We can derive the port level fluid model of the switch from relation 3.8 and 3.5, or directly from 3.7.

$$B_k(t) = B_k(0) + r_k t - E_k(t) \geq 0,$$

$$\dot{E}_k(t) = \sum_{\pi \in \Pi} \sum_{(i,j):z_{ij}(t) > 0 \& (i,j) \rightarrow k} \pi_{ij} \dot{T}_{\pi}^m(t) \geq 0, \quad (3.9)$$

$$\sum_{\pi \in \Pi} T_{\pi}^m(t) = t.$$

Note that (3.9) is not an independent complete set of equations that describes the dynamic behavior of the port level fluid model for the switch, since in the second equation we still use link level parameters ( $z_{ij}(t)$ ) of the switch.

The matching algorithm that is used for scheduling provides additional fluid model equations. For instance, if we use the conventional maximum weighted matching algorithm and consider number of backlogged cells of every link,  $Z_{ij}(t)$  as its weight, there will be one additional fluid equation for every link  $(i, j)$  [13],

$$\dot{T}_\pi^m(t) = 0 \quad \text{if} \quad \langle \pi, Z(t) \rangle < \langle \pi', Z(t) \rangle \quad \text{for some } \pi' \in \Pi . \quad (3.10)$$

In other words, at time  $t$ , under the maximum weight matching algorithm, a matching  $\pi$  that has less weight than another matching  $\pi'$  is not employed.

Here, we consider weighed matching algorithms that function based on the port weight rather than link weights. Weight of a port is total weight of all links connected to it. The LPF [31] is one of these algorithms, where we consider the backlogged traffic in every port as the weight of that node. The maximum node matching (MNM) [35] that we introduced in chapter 2 is another example, where weights are amount of service that scheduler owes every link, according to the reserved rate of that port. The only difference between the MNM and LPF is in the weight function that is used for links, but they have the same complexity. We proved that the obtained matching under the MNM is a min-max fair, and it has the maximum lexicographic ordering. In other words, it is impossible to add a new port to the matching, without removing a higher weight node from the matching. This is also true in the LPF for the corresponding weight vectors. This property can be used to write an additional fluid equation for the LPF, however we consider a generalized class of matching algorithms, MNCM that includes the LPF, and derive some interesting result for this general class of matching algorithms. The significance of the MNCM class becomes more clear later, when we introduce a deterministic matching algorithm in the MNCM that has lower complexity, and has a very good delay performance. First we need to define the MNCM:

**Definition 3.2.1** *A maximal size matching algorithm belongs to the MNCM class if and only if it always uses a matching  $m$  that contains all nodes that their weight are greater than  $(1 - 1/N)B_{\max}$ , where  $B_{\max}$  is the maximum weight.*

For now, we assume that the MNCM class is non-empty, and for any combination of the link weight values there exists at least one matching that is in the MNCM. Under this assumption, the LPF turns out to be in the MNCM. Recall that the LPF is a min-max matching algorithm. This means that if a node  $i$  is not in the LPF matching, there is no matching  $m$  that contains node  $i$  and all nodes that are in the LPF matching and their weight is larger than  $i$ .

In order to prove that the LPF is in the MNCM, we assume that it is not, and reach a contradiction. Assume that the LPF is not in the MNCM, then there should be a node  $i$  that is not in the LPF but its weight is greater than  $(1 - 1/N)B_{\max}$ . On the other hand, we know that there exists a matching  $m$  in the MNCM that contains node  $i$  and all other nodes that their weight is larger than  $i$ . This contradicts with the min-max property of the LPF, and hence the LPF should be in the MNCM. In the next section, we prove that any MNCM scheduler is efficient, and this immediately proves that the LPF is efficient. Moreover, we show that there are some less complex algorithms in the MNCM class too.

Recall that the matching algorithm that is used for scheduling results in additional fluid model equations. For any matching in the MNCM the following additional fluid equation holds. For any port  $k$  such that  $B_k(t) > (1 - 1/N)B_{\max}$  we have,

$$\sum_{\pi \in \Pi} \left( \sum_{(i,j): z_{ij}(t) > 0 \& (i,j) \rightarrow k} \pi_{ij} \right) \dot{T}_{\pi}^m(t) = 1 \quad \text{if } B_k(t) > (1 - 1/N)B_{\max} \quad (3.11)$$

Equation (3.11) says that all ports that their weight is greater than  $(1 - 1/N)B_{\max}$  are fully served under an MNCM policy. In fact, we are restating the

definition of the MNCM policy here. We can do that since the threshold value is a linear function of the node weights. For instance, if the scheduling policy was to include all nodes that their weight is greater than  $B_{\max} - 1$ , we could not use the same criterion in the corresponding fluid model, since the threshold is not a linear function of the node weights.

In order to make this clear, it would be helpful to go over some basic concepts of the fluid model technique and show how the fluid model variables such as  $B(t)$  are derived from the original queueing model parameters such as  $B(n)$  [12]. Starting from the discrete process  $B(n)$ , we define its continuous time extension  $\widehat{B}(t)$  by linear interpolation. The fluid model focuses on the behavior of,

$$\lim_{k \rightarrow \infty} \frac{\widehat{B}(kt)}{k} = B(t), \quad (3.12)$$

that is called the fluid limit of the queue length vector. Notice that if the weight of a node  $i$ ,  $B_i(n)$  is greater than  $(1 - 1/N)B_{\max}(n)$ , then its fluid limit counterpart  $B_i(t)$  is also greater than  $(1 - 1/N)B_{\max}(t)$ . In contrary, if  $B_i(n) > B_{\max}(n) - 1$ , we can **not** conclude that  $B_i(t) > B_{\max}(t) - 1$ .

Before ending this section we need to define the function  $f(B(t))$ . Function  $f(B(t))$  is defined from  $R^{2N} \rightarrow R$  as,

$$f(B(t)) = \max \{B_1(t), \dots, B_{2N}(t)\} \quad (3.13)$$

This function plays a role similar to the Lyapanov function in the stability proofs. Under a fluid model,  $B(t)$  is differentiable, and consequently,  $f(B(t))$  is continuous but not necessarily differentiable. In paper [13], norm 2 of the weight vector is used in definition of the function  $f(t)$ . Therefore,  $f(t)$  is differentiable, but here we have used norm infinity and the function  $f(t)$  is not differentiable anymore.

### 3.3 Stability Results

Our main objective here is to prove the following theorem,

**Theorem 3.3.1** *A switch operating under an MNCM matching algorithm is efficient.*

To prove theorem 3.3.1, we will use the following theorem that is proved in [13].

**Theorem 3.3.2** *A switch operating under a matching algorithm is rate stable if the corresponding fluid model is weakly stable.*

A fluid model is weakly stable if for every fluid model solution  $(E, T, B)$  with  $B(0) = 0$ ,  $B(t) = 0$  for all  $t \geq 0$ .

Intuitively, from equation 3.12 we can conclude that in the fluid model  $B(0) = 0$ , whenever there is a finite number of customers in the system at time 0. Moreover, if the length of the queues  $B(n)$  stays finite with probability one, the fluid limit variable  $B(t)$  should remain 0. This is consistent with the definition of weakly stable fluid model.

**Proof of theorem 3.3.1:** Let  $(E, T, B)$  be a solution to equations (3.9), (3.11) with  $B(0) = 0$ . In order to prove that  $B(t)$  remains zero for  $t > 0$ , it suffices to prove that  $f(B(t))$  remains zero. Therefore, it is sufficient to show that if the function  $f(B(t)) > 0$ , it will be absolutely decreasing.

Suppose that  $f(B(t_0)) = L > 0$ . Due to continuity of  $f(B(t))$ , there exists  $\delta_0 > 0$  such that every node that has the maximum weight for some  $t \in [t_0, t_0 + \delta_0]$ , its weight remains in the interval  $((1 - 1/N)f(B(t)), f(B(t)))$  for all  $t \in [t_0, t_0 + \delta_0]$ . For example, suppose that the node  $n_{\max}(t_0)$  has the maximum weight at time  $t_0$ . It is possible that this node does not have the maximum weight in the

neighborhood of  $t_0$ . However, due to continuity, its weight remains in the  $((1 - 1/N)f(B(t)), f(B(t)))$  interval for some time interval  $[t_0, t_0 + \delta]$ .

Let  $N(t_0, t_0 + \delta_0)$  be the set of the nodes that their weight is maximum for some  $t$  in  $[t_0, t_0 + \delta]$ . By the definition of the MNCM and the way  $\delta_0$  is set, all nodes  $i$  in  $N(t_0, t_0 + \delta_0)$  are fully utilized for all  $t$  in this interval, since their weight is always in the  $((1 - 1/N)f(B(t)), f(B(t)))$  region. Therefore, their backlog function is decreasing and consequently,  $f(B(t))$  which is the maximum weight of the nodes in  $N(t_0, t_0 + \delta_0)$  will be decreasing for all  $t \in [t_0, t_0 + \delta_0]$  as well.

Using the same argument as above, given any initial non-zero backlogged value at time  $t_0$  it is possible to partition the time line into the intervals,  $[t_k, t_{k+1}]$ ,  $k = 0, 1, \dots$ , ( $t_{k+1} - t_k = \delta_k > 0$ ) such that  $f(B(t))$  is either zero or absolutely decreasing in all intervals.

Therefore, we have proved that  $Z(t) = 0$  and fluid model is weakly stable. Using theorem 3.3.2, we can conclude that the MNCM policies are efficient. ■

The novelty of our approach is to consider the maximum norm in definition of function  $f(\cdot)$ . In the previous approaches, usually a second norm function was used in definition of  $f(\cdot)$ .

Our proof is not complete yet. We have to prove that there is always a matching  $m(t)$  that belongs to the MNCM. The matching  $m(t)$  is in the MNCM if and only if it contains all nodes that have their weight in the interval  $((1 - 1/N)f(B(t)), f(B(t)))$ . The existence of such a matching  $m(t)$  is proved in the following lemma.

**Lemma 3.3.3** *Consider an  $N \times N$  input buffered switch fabric. For all values of  $f(B(t)) > 0$ , all nodes that have their weight in  $((1 - 1/N)f(B(t)), f(B(t)))$  can be included in a matching  $m(t)$ .*

**Proof of lemma 3.3.3:** Without loss of generality, assume that there is an input node such that its weight is in the  $\epsilon$ -neighborhood of  $f(B(t))$  and can not be included in the matching. We show that  $\epsilon \geq (1/N)f(B(t))$ . Let  $A$  be the set of all input nodes that their weight is in  $\epsilon$ -neighborhood of  $f(B(t))$  and  $E$  be the set of all output nodes. Using Hall's theorem<sup>1</sup>, since there is not a perfect matching between  $A$  and  $E$  that contains all nodes in  $A$ , we can conclude that there is a subset  $S$  of nodes in  $A$  such that its cardinality is greater than its neighbor set  $NS$ ,

$$|NS| \leq |S| - 1. \quad (3.14)$$

Note that  $NS$  is the set of all output nodes that have a common link with at least one node in  $S$ . Let  $W(S)$  be the total weight of all nodes in  $S$ ; by definition we have,

$$(f(B(t)) - \epsilon) |S| \leq W(S) \leq f(B(t)) |S| \quad (3.15)$$

Since  $NS$  is connected to all links connected to  $S$ ,

$$W(NS) \geq W(S). \quad (3.16)$$

From relations 3.15 and 3.16,

$$W(NS) \geq (f(B(t)) - \epsilon) |S| \quad (3.17)$$

---

<sup>1</sup>Hall's Theorem: There exists a perfect matching that contains all input nodes  $A$  in a bipartite graph if and only if  $|S| \leq |NS|$  for every subset  $S \subseteq A$ , where  $NS$  is the set of neighbor nodes of set  $S$ .

Therefore, there is a node  $k \in NS$  such that,

$$\begin{aligned}
u_k(t) &\geq \frac{W(NS)}{|NS|} \\
&\geq \frac{(f(B(t))-\epsilon)|S|}{|NS|} \\
&\geq \frac{(f(B(t))-\epsilon)|S|}{|S|-1} \\
&\geq \frac{(f(B(t))-\epsilon)(N)}{N-1} \\
&= \frac{(f(B(t))-\epsilon)(N)}{N-1} - f(B(t)) + f(B(t)) \\
&= \frac{N}{N-1} \left( \frac{f(B(t))}{N} - \epsilon \right) + f(B(t))
\end{aligned} \tag{3.18}$$

Second relation is from 3.17, third relation is from 3.14 and fourth relation is due to the fact that  $|S| \leq N$ .

If  $\epsilon(N, f(B(t))) < f(B(t))/N$  from 3.18 we have

$$u_k(t) > f(B(t)). \tag{3.19}$$

This contradicts the definition of  $f(B(t))$ , hence there always exists a matching  $m$  that contains all nodes that their weight is in epsilon neighborhood of  $f(B(t))$  for  $\epsilon(N, f(B(t))) < f(B(t))/N$ . ■

### 3.3.1 Comparison of the MNCM and LPF

It would be instructive to compare the MNCM with the LPF. We know that the LPF is min-max fair, i.e., weight of the node that has largest weight among nodes

that are not in the matching is minimized. Therefore, LPF not only contains all nodes that their weight is in the interval  $((1 - 1/N)f(B(t)), f(B(t))]$ , but it will also try to contain all other nodes such that the priority is always given to a node with larger weight. In contrary, the MNCM only focuses on those nodes that their weight is in the interval  $((1 - 1/N)f(B(t)), f(B(t))]$  and neglects other nodes. In fact, for the MNCM we have identified a group of nodes that their weight is larger than others and we know that it is possible to contain all of them in a matching. Moreover, we have proved that for the stability it is sufficient to contain only these nodes and we really do not care about other nodes. Intuitively, it makes sense that for stability the only concern is for those nodes that their weight is close to the maximum value. This fact should enable us to come up with rate stable algorithms that has lower complexity algorithms than the LPF.

We can also view the MNCM as a quantized version of the LPF. Consider a quantizer for the node weights that has two levels,  $f(B(t))$  and 0. All weight values that are above  $(1 - 1/N)f(B(t))$  are quantized to  $f(B(t))$  and all other weights are quantized to 0. It should be clear that if we use the LPF with the quantized weights the result would be an MNCM scheduler.

In the next section, we introduce a matching algorithm that is in the MNCM and has lower complexity than all previous suggested deterministic algorithms including the LPF.

### 3.4 Maximum First Matching (MFM)

In the previous section, we proved that the MNCM algorithms are efficient. In the MFM algorithm, we find two matchings that contain respectively, all input and output nodes that their weight is in the  $((1 - 1/N)f(B(t)), f(B(t))]$  interval.

We call these nodes the marked nodes. Note that existence of such matchings is proved in lemma 3.3.3.

The LPF algorithm that is introduced in [29] is an example of MNCM algorithms. LPF works on non-matched nodes one by one starting with the node that has highest weight. To find the matching, for each one of the nodes we need to search all  $N^2$  links of the bipartite graph. Since there are  $2N$  nodes, the complexity of the LPF turns out to be  $O(N^3)$ . Basically, this algorithm is a modification of the Edmonds Karp max-flow algorithm [15] and its complexity is the same as that.

The Hopcroft and Karp maximum size matching algorithm [23] has  $O(N^{2.5})$  complexity. In this algorithm the nodes are introduced into the matching simultaneously; this is not possible in the LPF algorithm.

In the MFM algorithm, we replace the maximum weighted matching algorithm to a limited number of the maximum size matching algorithms. In this way, we can use the Hopcroft and Karp algorithm to obtain the maximum size matching in each step, and reduce the complexity of the algorithm. We can do this since our objective is not to find the maximum weighted matching anymore, but to have a matching that contains all node with the maximum weight. The details of the algorithm is as follow (complexity of each step is given at the end of each step),

**Algorithm 3.4.1** (*MFM*):

1. Sort all input and output nodes according to their weight , and mark all nodes that their weight is greater than  $(1 - 1/N)f(B(t))$  ( $O(N \log N)$ ).
2. Find a matching  $M_1$  that contains all marked input nodes ( $O(N^{2.5})$ ).
3. Find a matching  $M_2$  that contains all marked output nodes ( $O(N^{2.5})$ ).

4. Combine  $M_1$  and  $M_2$  to get a merged matching containing all critical nodes in  $M_1 \cup M_2$  ( $O(N)$ ).
5. Perform a simple sorted maximal size matching algorithm on the rest of nodes not in the matching ( $O(N^2)$ ).
6. Combine matchings of steps 4 and 5; result is the MFM matching.

◇

In step 1, all nodes are sorted according to their weights. Since there are  $2N$  nodes the complexity of this step is  $O(N \log N)$ . In step 2, we consider all marked input nodes, together with all output nodes and find a maximum size matching,  $M_1$ , in the corresponding graph. Note that in lemma 3.3.3, we concluded that such a matching always exist. The complexity of this step is  $O(N^{2.5})$  which is the complexity of finding a maximum size matching. Step 3 is similar to step 2, but here we find a matching,  $M_2$  that covers all marked output nodes. In step 4, we combine the two matchings to find a matching that covers the input nodes covered in  $M_1$  and the output nodes covered in  $M_2$ .

It is useful to elaborate more on step 4. Consider the bipartite graph that contains only those links that are in  $M_1$  or  $M_2$  ( $M_1 \cup M_2$ ). Our objective is to find a matching in the bipartite graph  $M_1 \cup M_2$  that contains all of the marked nodes. The maximum degree of a node in the combined graph is 2, since there are at most two links connected to every node (one from  $M_1$  and one from  $M_2$ ). This bipartite graph can be divided into disjoint sub-graphs. In each subgraph, we have to obtain an optimal matching. The subgraphs can be classified into four classes and we explain the process of obtaining a matching for each of them. In general,

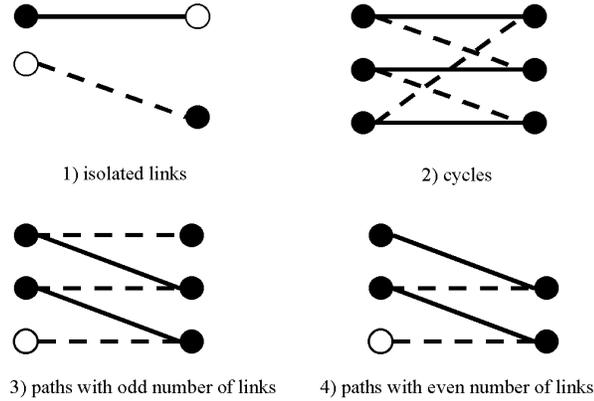


Figure 3.1: Four possible forms of sub-graphs in a combined matching. Links of  $M_1$  are shown with solid lines and  $M_2$  with dashed lines. Marked nodes are shown as black nodes.

we have to select a group of links, belonging to either  $M_1$  or  $M_2$ , that construct a matching in the sub-graph as follow (Fig.3.1):

1. **Single link subgraph:** If there is an isolated link connecting two nodes, include that link into the matching.
2. **Cycle subgraph:** Since the graph is bipartite, cycles have even number of links that alternatively belong to  $M_1$  and  $M_2$ . We can select either set of the links for matching, since both cover all nodes in the cycle.
3. **Path subgraph with odd number of links:** Here, we have an alternate path, that is an augmented path in the matching terminology. Basically one set of links either those belonging to  $M_1$  or  $M_2$  has one more element. That set covers all the nodes in the sub-graph, and thus should be selected for the matching. Both end nodes of the path belong to same matching and one

of them is in the input side and the other in the output side. The set of alternating links that is connected to the end nodes should be included in the matching.

4. **Path subgraph with even number of links:** Without loss of generality, assume such a path that starts from an input node. Obviously, since there are an even number of the links, this path ends also at an input node. One of these nodes belongs to  $M_1$ , and therefore is marked the other belongs to  $M_2$  and is not a marked node (because it is not in  $M_1$ ). If we select those links that belong to  $M_1$ , only the unmarked end node will be excluded, which is not important.

Therefore, we can employ the following general rule for the paths.

*If a path starts from an input (output) node that is marked, include those set of alternating links in the matching that contains the specified marked node.*

Therefore, we have to take the following sub-steps in step 4 to merge  $M_1$  and  $M_2$ :

**Algorithm 3.4.2** (*Merge (step 4)*):

**4.1** *Search for an input or output marked node  $i$  that is not selected yet and is connected to only one link in  $M_1 \cup M_2$ . If such a node does not exist go to step 4.4.*

**4.2** *In the path that is originated from the node  $i$ , select the set of the alternating links that contain the link connected to node  $i$  and include it into the matching. Discard the other alternating set of the links.*

**4.3** *Flag all nodes that are in the path covered in 4.2 as searched and go back to step 4.1.*

**4.4** *Search for an input or output node  $i$  that is not selected yet and is connected to two links in  $M_1 \cup M_2$ . If such a node does not exist, the desired matching is found and the search is finished.*

**4.5** *In the cycle containing node  $i$  select one set of alternating links (arbitrary) and include them into the matching and discard the other set.*

**4.6** *Flag all nodes that are in the cycle covered in 4.5 as searched and go back to step 4.4.*

In the merge algorithm steps 4.1 to 4.3 search for paths and steps 4.4 to 4.6 search for cycles. To determine the complexity of step 4, notice that in the merge algorithm described above we are searching the bipartite graph  $M_1 \cup M_2$ . This graph has  $2N$  links and  $2N$  nodes and therefore the search complexity is  $O(N)$ . The ultimate matching covers all critical marked nodes.

In step 5, we perform a sequential maximal matching on the nodes that are not in the matching. Starting from the node with the maximum weight, we scan all its neighbors starting from the one with the maximum weight. If we find a neighbor that is not in the matching we match and include the corresponding link to the matching list. The number of nodes that we have to scan in this step is less than  $2N$ , and for each one we have to check at most  $N$  neighbors. This has a complexity of  $O(N^2)$ .

One may think there should be algorithms that are practically efficient and has lower complexity than the MFM. the maximal sorted matching (MSM) can be considered as a first attempt toward such an algorithm. The MSM is basically very

similar to iLPF algorithm that is introduced in [31]. Similar to the MFM, MSM works on a sorted list of the nodes, however it does not treat the marked nodes separately to ensure that they are contained into the matching. The MSM scans all nodes starting with the one with maximum weight and going down the sorted list, and tries to include the scanned (primary) nodes into the matching. At every step it scans all neighbor nodes (starting from the neighbor node with maximum weight) of the primary node until it finds a free (not matched) node or until it scans all neighbors. If it finds a free neighbor, it considers it as the secondary node and match the primary and secondary nodes and add the connecting link to the matching. This is essentially similar to the step 5 of the MFM and results in an  $O(N^2)$  complexity algorithm.

**Algorithm 3.4.3** (*MSM*):

1. Sort all input and output nodes according to their weight  $O(N \log N)$ .
2. Perform a simple sorted maximal matching algorithm on all of the nodes ( $O(N^2)$ ).

◇

Although we can not prove that MSM achieves 100% throughput, in simulations its performance was identical to MFM. We review some of the simulation results in the next section.

### 3.5 Rate Provisioning and the MSUIM Scheduling Algorithm

In this section, we introduce the Maximum Size Unit Interval Matching (MSUIM) scheduling algorithm that can provide rate guarantees to the backlogged connections. This result is useful by itself and is also insightful for understanding the stochastic stability results of the MNCM. Even though the MSUIM is not in the MNCM, it is based on the same concepts and for the purpose of rate guarantees it appears to be more suitable.

Consider that a variable guaranteed rate matrix  $G(n)$  is given. Our objective is to provide the reserved rates  $g_{ij}(n)$  to the cells going from the input  $i$  to the output  $j$ , whenever there are backlogged cells for that connection. We consider a credit based scheduling algorithm, where  $v_{ij}(n)$ , credit of the link  $(i, j)$ , reflects how much service we owe to that link at time  $n$ . When a link  $(i, j)$  is backlogged, its credit increments by its guaranteed rate  $g_{ij}(n)$  and decrements by one if that link is scheduled at the time slot  $n$ .

Furthermore, for simplicity we assume that all links are always backlogged and a link can not be scheduled if its credit is less than one. Therefore, the credit matrix evolves according to the following linear relation,

$$V(n+1) = V(n) + G(n) - S(n). \quad (3.20)$$

$S(n)$  is the scheduling matrix at time  $n$  and  $s_{ij} = 1$  if link  $(i, j)$  is scheduled at time  $n$  and is zero otherwise.

To avoid confusion with the number of backlogged cells, we use  $u_k(n)$   $k = 1, \dots, 2N$  to show the node weight in this section:

$$u_k(n) = \sum_{(i,j):(i,j) \rightarrow k} v_{ij}(n). \quad (3.21)$$

Now, we define the MSUIM scheduling algorithms that we use in this section. Note that weight of a node is sum of the credits of links connected to it.

**Definition 3.5.1** *A maximal size matching algorithm belongs to the MSUIM class if and only if it always uses a matching  $m$  that contains all nodes that their weight are greater than  $u_{\max} - 1$ , where  $u_{\max}$  is the maximum weight.*

Therefore, for the MSUIM, we have to find the maximum weight of all nodes  $u_{\max}$  and mark all nodes that their weight is in the interval  $[u_{\max} - 1, u_{\max}]$ . Next, we have to find a matching that contains maximum number of marked nodes.

It is not always possible to contain all marked nodes in the MSUIM but as we proved in lemma 3.3.3 for large enough maximum weights this becomes possible.

The guaranteed rate matrix is feasible if,

$$\begin{aligned} \sum_{i=1}^N g_{ij} &\leq 1 \quad j = 1 \cdots N, \\ \sum_{j=1}^N g_{ij} &\leq 1 \quad i = 1 \cdots N. \end{aligned} \quad (3.22)$$

We will show that if the guaranteed rate matrix is inside the feasibility region, the credit matrix remains bounded below  $N + 1$ . Hence, the algorithm can provide variable rate guarantees.

We define function  $f(n)$  to be the equal to the maximum weight node.

$$f(n) = \max \{u_1(n), \dots, u_{2N}(n)\}. \quad (3.23)$$

**Theorem 3.5.2** *Consider an  $N \times N$  input-buffered switch with all virtual output queues backlogged and a given variable guaranteed rate matrix  $G(n)$  inside the*

*feasibility region. If the scheduling policy is MSUIM and credits of links are used as the weight function then credits of all links remain bounded below  $N + 1$ .*

**Proof of theorem 3.5.2:** We show that for  $f(n) > N$ , we can conclude  $f(n+1) < f(n)$ . Due to the feasibility of the matrix  $G(n)$  weight of nodes can not increase by more than one at every cell time and since  $f(0) = 0$ , the maximum weight of a node is always less than  $N + 1$ . Therefore, maximum credit of a node is always less than  $N + 1$  and the algorithm can provide rate guarantees.

The proof is very similar to lemma 3.3.3. Let  $f(n) > N$ , we show that there exists a matching that contains all marked nodes, i.e. all nodes that their weight is in  $[f(n) - 1, f(n)]$ . Suppose that this is not true and without loss of generality, there is a marked input node that can not be included in the matching. Using Hall's theorem, this means that there is a subset of the input marked nodes  $S$  such that its cardinality is greater than its neighbor set  $NS$ ,

$$|NS| \leq |S| - 1. \tag{3.24}$$

Let  $W(S)$  be the weight of all nodes in  $S$ , by definition we have,

$$(f(n) - 1) |S| \leq W(S) \leq f(n) |S| \tag{3.25}$$

Since  $NS$  is connected to all links connected to  $S$ ,

$$W(NS) \geq W(S). \tag{3.26}$$

From relations 3.25 and 3.26,

$$W(NS) \geq (f(n) - 1) |S| \tag{3.27}$$

Therefore, there is a node  $k \in NS$  such that,

$$\begin{aligned}
u_k(n) &\geq \frac{W(NS)}{|NS|} \\
&\geq \frac{(f(n)-1)|S|}{|NS|} \\
&\geq \frac{(f(n)-1)|S|}{|S|-1} \\
&\geq \frac{(f(n)-1)(N)}{N-1} && (3.28) \\
&= \frac{(f(n)-1)(N)}{N-1} - f(n) + f(n) \\
&= \frac{N}{N-1} \left( \frac{f(n)}{N} - 1 \right) + f(n) \\
&> f(n)
\end{aligned}$$

This contradicts by the definition of  $f(n)$ . Hence, for  $f(n) > N$  all marked nodes can be included in the matching. Therefore, if  $k$  is a marked node,

$$u_k(n+1) = u_k(n) - 1 + \sum_{(i,j) \rightarrow k} g_{ij} < u_k(n) \leq f(n), \quad (3.29)$$

and if  $k$  is not a marked node ( $u_k(n) < f(n) - 1$ ),

$$u_k(n+1) \leq u_k(n) + \sum_{(i,j) \rightarrow k} g_{ij} < u_k(n) + 1 < f(n). \quad (3.30)$$

From relations 3.29 and 3.30, we can conclude that,

$$f(n+1) < f(n). \quad (3.31)$$

Therefore, if  $f(n) \leq N$  we know that  $f(n+1) < N+1$  by definition and if  $N < f(n) < N+1$  from 3.31 we can conclude that  $f(n+1) < f(n) < N+1$ , and therefore,  $f(n)$  does not exceed  $N+1$ . ■

It should be clear that we can reach the same result with a scheduling algorithm in MNCM since for  $f(n) \geq N$  the corresponding interval of the MNCM covers the interval of the MSUIM.

## 3.6 Simulations

In this section we present our simulation results. The main objective is to study delay performance of the proposed scheduling algorithms. For practical purposes it is not sufficient for a scheduling algorithm to have 100% throughput. Therefore, it is essential to compare and study the delay performance of different scheduling algorithms.

We consider a  $32 \times 32$  input-buffered switch. For each experiment the average throughput,  $\rho$ , of the switch is given. A random  $32 \times 32$  rate matrix is generated such that the aggregate rate of every input (row summation) and output (column summation) is  $\rho$ .

The rate matrix is generated iteratively. In each iteration a new flow between two randomly selected input and output nodes is generated. After selecting the input node  $i$ , and output node  $j$ , of a flow the maximum allowable rate (MAR) for that flow is set to minimum of:

- Maximum flow rate that is set to 0.1 in our experiments.
- Difference between  $\rho$  and aggregate rate of input port  $i$ . Aggregate rate of  $i$  is summation of all elements of rate matrix in row  $i$ .
- Difference between  $\rho$  and aggregate rate of output port  $j$ . Aggregate rate of  $j$  is summation of all elements of rate matrix in column  $j$ .

Next, a uniform random number between zero and MAR is generated as the rate of that flow and it is added to the  $(i, j)$  element in rate matrix. Basically, MAR is set so that the aggregate rate of each port does not exceed  $\rho$ . This procedure is repeated until aggregate rates of all ports become very close to  $\rho$ . In this way, we are able to create a non-uniform rate matrix. This matrix is used to generate i.i.d. Bernoulli arrival patterns for all of the connections.

We have studied four different systems which are three input-buffered switches with matching algorithms LPF, MFM, MSM, and an output buffered switch. The first three systems were simulated, but for the output-buffered system we modelled it as an M/D/1 system. We have elaborated on the first three, and proved here that LPF and MFM achieve 100% throughput. The output-buffered system is mainly included as a benchmark.

The main performance measure that we have considered here is delay versus throughput, and it is plotted for all of the scheduling algorithms in Fig. 3.2. For each scheduler the simulation is stopped, when the throughput reaches 1.

As we expected all of the systems achieve 100% throughput. Delay performance of LPF is slightly better than MFM and MSM, but recall that it is more complex as well. The performance of MFM and MSM is not distinguishable. Although we have not been able to prove the stability of MSM, the simulations reveal that for practical purposes it functions as 100% throughput algorithm, with suitable delay performance. Recall that MSM complexity is  $O(N^2)$ , which is lower than other proposed algorithms.

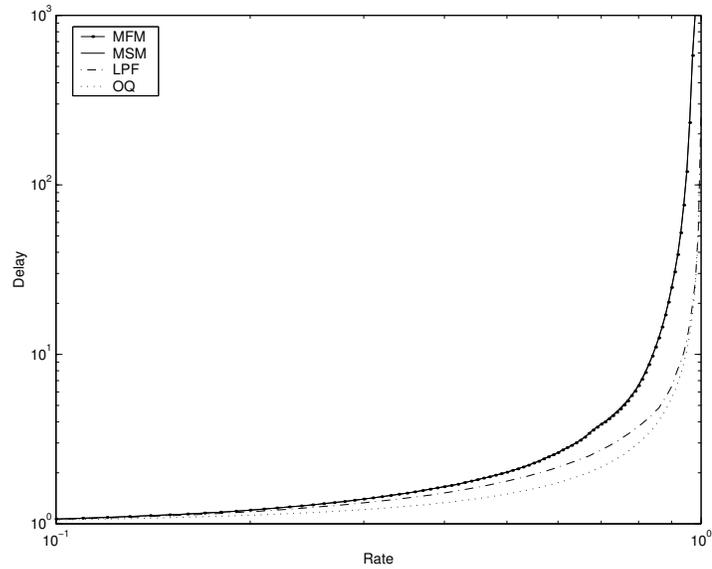


Figure 3.2: Average Delay v.s. Throughput for different matching algorithms. The output queueing system is modelled as an M/D/1 system.

### 3.7 Summary

In this chapter, a new class of weighted matching algorithms for scheduling in input-buffered switches is introduced. Using the fluid model techniques, we were able to prove that they achieve 100% throughput. Basically, it is shown that to achieve 100% throughput it suffices to include only nodes that their weight is in  $\epsilon$ -neighborhood of the maximum weight, where  $\epsilon = u_{\max}(1 - 1/N)$ . This result can lead us to the development of less complex and more efficient scheduling algorithms for the input-buffered switches. Two particular matching algorithms, the MSM and MFM that can be considered for practical systems, are also introduced. The MFM is proven to achieve 100% throughput. Furthermore, the MSUIM scheduling algorithm is proposed and it is proved that it can provide rate guarantees and its maximum service deviation from guaranteed rate is bounded to  $N$ .

# Chapter 4

## Analysis and design of self-randomized max-min fair schedulers

In this chapter, we consider the self-randomized scheduling policies for input buffered switches. A scheduling policy is said to be self-randomized if the randomized component (process) of the randomized policy is replaced by a pseudo-random process that is a function of the cell arrival process. We use fluid model techniques to show that the self-randomized scheduling algorithms deliver 100% throughput. The only assumption on the arrival pattern is that it satisfies strong law of large numbers, and no input or output port is oversubscribed. We provide a general architecture for the design of the self-randomized algorithms, and introduce two algorithms that consider number of backlogged cells as the weight function. We then introduce the concept of the max-min fair self-randomized scheduling algorithms. The idea here is to introduce self-randomized algorithms that can provide QoS by sharing the switch bandwidth proportional to the assigned weights. We

introduce three self-randomized scheduling algorithms, and prove that they are max-min fair. In order to study and compare the performance of the proposed scheduling algorithms, several simulations are carried out and their results are provided and discussed.

In chapters 2 and 3, we introduced and elaborate on the deterministic scheduling algorithms for the input-buffered switch fabrics. Our objective was to develop new algorithms that achieve 100% throughput, can provision rate, and have low complexity. For instance, the MNCM class of matchings that was introduced in section 3, achieves 100% throughput. Moreover, if we use the link weight function that was introduced in chapter 2 in equations 2.14, 2.15, MNCM can track appropriate fluid policy, and thus to provision rate. We also introduced the MFM algorithm in MNCM class with  $O(N^{2.5})$  complexity and the MSIM algorithm that is not in MNCM but can provide rate guarantees. Even though MFM is the lowest complexity deterministic scheduling algorithm that achieves 100% throughput, it is still desirable to develop algorithms with lower complexity.

Tassiulas introduced a new class of randomized scheduling algorithms in [36]. These algorithms have linear complexity  $O(N)$ , and at the same time achieve 100% throughput. The basic idea is instead of finding the scheduling matching, to select it from a set of candidate matchings. A simple randomized procedure is used to obtain or update the candidate set in every time slot. A completely randomized set results in a very low performance system. The candidate matching set in [36] contains two matchings, one randomly selected matching and the matching that was used in the previous time slot. The candidate matching with higher total weight of links is selected, where weight of the link is number of backlogged cells. The stability proof is for i.i.d. arrival. Intuitively, since state of the switch (number

of backlogged cells) changes slightly in every time slot, a good matching in previous time slot is also a good matching for the next time slot.

Although this randomized algorithm is stable, it does not have a good delay performance. Paper [19] proposed several modifications to the basic randomized algorithm to improve its delay performance. One of the contributions of [19] is introduction of the self-randomized algorithms. A scheduling policy is said to be self-randomized if the randomized component of the randomized policy is replaced by a pseudo-random process that is a function of the cell arrival process.

In this chapter, we focus on the self-randomized scheduling algorithms and introduce some new algorithms that have better delay performance than the previously proposed algorithms. To that end, we introduce a general architecture for the self-randomized schedulers and propose two specific scheduling algorithms that use the number of backlogged cells as the weight of the links. We use the fluid model techniques to establish some new results for the throughput of the randomized scheduling algorithms. More specifically, we prove that the original randomized algorithm of [36], and all of the self-randomized scheduling algorithms that are introduced in [19] delivers 100% throughput. The only assumptions on the input traffic is that it satisfies SLLN, and is admissible. To the best of our knowledge this is the first time that the fluid model is applied to the randomized scheduling algorithms, and all previous stability results were contained to i.i.d assumption.

None of the randomized scheduling algorithms that were introduced previously was capable of QoS provisioning. We employ the notion of the max-min fair to the self-randomized schedulers to develop self-randomized schedulers that can provision rate. The basic idea behind a max-min fair scheduler is to allocate bandwidth

among flows proportional to their weights, and if a flow can not utilize its bandwidth, because of the constraints elsewhere in the network, then the residual bandwidth is distributed proportionally among others [22]. Tassiulas and Sarkar [38] has recently introduced a max-min fair scheduler that is not originally proposed for input-buffered switches, but is applicable to them too. We can categorize this scheduling algorithm as a deterministic MWM scheduling algorithm that weight of links (flows) are number of tokens allocated to them. Tokens are distributed among links with their max-min rate.

We extend the idea of max-min fair scheduling algorithms and introduce the concept of max-min fair self-randomized scheduling algorithms. Three different self-randomized max-min fair algorithms are introduced in this chapter. For each algorithm, it is proved that it delivers the max-min rate to all of the links. Furthermore, their delay performance is studied and compared using simulation results.

The rest of this chapter is organized as follows. Section 4.1 defines and introduces the basic concepts that are used in the rest of the paper. Section 4.2 contains proof of the main stability result for the randomized scheduling algorithms. Section 4.3 introduces the general architecture that is adopted for self-randomized schedulers in this chapter, and elaborate on some of its basic features. Section 4.4 introduces two scheduling algorithms based on backlogged weight function, and provides analytical as well as simulation results for them. Section 4.5 contains theoretical and simulation results for the max-min fair randomized scheduling algorithm.

## 4.1 Models and Definitions

The basic assumptions and model are similar to what defined in section 3.1, so we do not repeat it here. We start with the discrete dynamic relations that govern the input buffered switch behavior. We adopt the notation that is used in [13]. Consider an input-buffered switch that employs scheduling algorithm  $m$ . Suppose that  $T_\pi^m(n)$  be the total time that permutation  $\pi$  is used up to time  $n$ . The following equations describe the switch dynamics,

$$\begin{aligned} Z_{ij}(n) &= Z_{ij}(0) + A_{ij}(n) - D_{ij}(n), \\ D_{ij}(n) &= \sum_{\pi \in \Pi} \sum_{l=1}^n \pi_{ij} \mathbf{1}_{\{Z_{ij}(l) > 0\}} (T_\pi^m(l) - T_\pi^m(l-1)), \\ \sum_{\pi \in \Pi} T_\pi^m(n) &= n. \end{aligned} \tag{4.1}$$

The first equation describes the basic relation of arrival, departure and backlogged packets. The second equation counts the number of total departures by counting number of times that a permutation matrix with a one in  $(i, j)$  position is used, when there were a backlogged cells at that VOQ.

The matching algorithm that is used adds more equations to (4.1). For instance, if we use the conventional MWM, with number of backlogged cells as the link weight, there will be one more equation for every link  $(i, j)$  [13],

$$T_\pi^{MWM}(n) - T_\pi^{MWM}(n-1) = 0 \quad \text{if} \quad \langle \pi, Z(n) \rangle < \langle \pi', Z(n) \rangle \quad \text{for some} \quad \pi' \in \Pi. \tag{4.2}$$

The above equation says that under the maximum weight matching algorithm, a matching  $\pi$  that has less total weight than another matching  $\pi'$  is not used.

### 4.1.1 Randomized Scheduling Algorithms and Fluid model

Consider the following scheduling algorithm,

**Algorithm 4.1.1** (*Tass*)[36]:

- Let  $S(n)$  be the schedule used at time  $n$ .
- At time  $n + 1$  choose a matching  $R(n + 1)$  uniformly at random from the set of all permutation matrices  $\pi \in \Pi$ .
- $S(n + 1) = \arg \max_{s \in \{S(n), R(n+1)\}} \langle s, Z(n + 1) \rangle$

At every time slot, a random matching is generated, and its weight is compared to the weight of the matching that is used in previous time slot. The matching with highest weight is selected for scheduling in the present time slot. Paper [36] proves that the above algorithm is stable under any Bernoulli i.i.d. admissible input. This proof is the basis for all subsequent stability proofs for other randomized scheduling algorithms. Here, we generalize the stability proof for any admissible arrival process. In other words, we prove that *TASS* scheduling algorithm is efficient. To that end, we introduce Basic Randomized (*BR*) scheduling algorithm.

**Algorithm 4.1.2** (*BR*):

- Let  $S(n)$  be the schedule used at time  $n$ .
- At time  $n + 1$  choose a matching  $R(n + 1)$  uniformly at random from the set of all permutation matrices  $\pi \in \Pi$ .
- $S(n + 1) = \begin{cases} R(n + 1) & \text{if } R(n+1) \text{ is MWM for } Z(n+1) \\ S(n) & \text{otherwise} \end{cases}$

BR algorithm does not have any practical merit since its performance is worse than MWM but its complexity is higher. Note that in step 3 of BR algorithm we need to determine the MWM scheduling. The BR algorithm is used to prove the

basic stability results for randomized scheduling algorithms. Let  $\eta(n)$  be a binary random variable such that,

$$\eta(n) = \begin{cases} 1 & \text{if } R(n) \text{ is MWM for } Z(n) \\ 0 & \text{otherwise} \end{cases} \quad (4.3)$$

Let,  $t_\pi^m(n) = T_\pi^m(n) - T_\pi^m(n-1)$ . We can add the following equation to (4.1) for the *BR* algorithm.

$$t_\pi^{BR}(n) = \eta(n)t_\pi^{MWM}(n) + (1 - \eta(n))t_\pi^{BR}(n-1) \quad (4.4)$$

Equation (4.4) is a difference equation that needs an initial condition at time 0 to fully characterize the *BR* scheduling policy. We assume that probability of scheduling a link at time 0 is at least equal to probability of cell arrival on that link,

$$\text{Prob}[\pi_{ij} = 1] \geq \lambda_{ij}. \quad (4.5)$$

Next, we review a deterministic continuous fluid model of a switch operating under some matching algorithm  $m$ . We use the same model that we used before in 3.8, and is originally given in [13],

$$\begin{aligned} Z_{ij}(t) &= Z_{ij}(0) + \lambda_{ij}t - D_{ij}(t) \geq 0 \\ \dot{D}_{ij}(t) &= \sum_{\pi \in \Pi} \pi_{ij} \dot{T}_\pi^m(t), \quad \text{if } Z_{ij}(t) > 0 \\ \sum_{\pi \in \Pi} T_\pi^m(t) &= t \end{aligned} \quad (4.6)$$

In order to translate (4.4) into fluid model, we first write it in the following form,

$$(1 - \eta(n))(t_\pi^{BR}(n) - t_\pi^{BR}(n-1)) = \eta(n)(t_\pi^{MWM}(n) - t_\pi^{BR}(n)) \quad (4.7)$$

Using Relations (4.7, 4.5), and assuming,  $\text{Prob}[\eta(n) = 1] = \epsilon > 0$ , we can add the following fluid equation for BR algorithm,

$$\begin{aligned} t_{\pi}^{BR}(t) &= \frac{\epsilon}{1-\epsilon}(t_{\pi}^{MWM}(t) - t_{\pi}^{BR}(t)), \\ \sum_{\pi \in \Pi} \pi_{ij} t_{\pi}^{BR}(0) &\geq \lambda_{ij}. \end{aligned} \tag{4.8}$$

Adding an equation to the fluid model is related to fluid limits and is discussed in [12]. Basically, the time difference of  $t_{\pi}^{BR}(n)$  is turned into its derivative and expectation of the random process  $\eta(n)$  has replaced  $\eta(n)$  itself. Note that we are assuming that  $\eta(n)$  satisfies the strong law of large numbers.

Recall that  $t_{\pi}^m(n) = T_{\pi}^m(n) - T_{\pi}^m(n-1)$ , and consequently in the fluid model we have  $t_{\pi}^m(t) = \dot{T}_{\pi}^m(t)$ . Using (4.6), (4.8), we can conclude,

$$\begin{aligned} \ddot{D}_{ij}^{BR}(t) &= \frac{\epsilon}{1-\epsilon}(\dot{D}_{ij}^{MWM}(t) - \dot{D}_{ij}^{BR}(t)), \\ \dot{D}_{ij}^{BR}(0) - \lambda_{ij} &\geq 0. \end{aligned} \tag{4.9}$$

Relation (4.9) provides additional equations that characterizes the BR scheduling algorithm fluid model.

## 4.2 Stability of Randomized Schedulers

In this section, we use the fluid model calculus to prove the stability of randomized scheduling algorithms. Our main objective here is to prove the following theorem,

**Theorem 4.2.1** *The Basic Randomized scheduling algorithm is efficient.*

To prove theorem 4.2.1, we will use the following theorem that is proved in [13] and we used it in chapter 3.

**Theorem 4.2.2** *A switch operating under a matching algorithm is rate stable if the corresponding fluid model is weakly stable.*

Recall that a fluid model is weakly stable if for every fluid model solution  $(D, T, Z)$  with  $Z(0) = 0$ ,  $Z(t) = 0$  for all  $t \geq 0$ . In [13] it is proved that the maximum weighted matching algorithm is rate stable, by proving that for the function  $V(t) = \langle Z(t), Z(t) \rangle$ ,  $\dot{V}(t) = 2\langle \dot{Z}(t), Z(t) \rangle \leq 0$ ,  $t \geq 0$ . Therefore, if  $Z(0) = 0$ , we can conclude that  $Z(t) = 0$ ,  $t \geq 0$ , and consequently MWM algorithm is rate stable. Since the MWM is rate stable, for  $Z(0) = 0$ ,

$$\dot{D}_{ij}^{MWM}(t) = \lambda_{ij} \quad t \geq 0 \quad (4.10)$$

We use equation (4.10) and relation between *MWM* and *BR* scheduling algorithms that is formulated in (4.9) to prove theorem 4.2.1.

**Proof of theorem 4.2.1:** We prove that starting with  $Z(0) = 0$ , we can conclude that  $\dot{D}_{ij}^{BR}(t) = \lambda_{ij}$  for all  $t \geq 0$  is the only solution, and therefore  $Z(t) = 0$  for  $t \geq 0$ . To that end, we rewrite the main differential equations and initial conditions that describe evolution of  $D^{BR}(t)$ ,

$$\begin{aligned} \ddot{D}_{ij}^{BR}(t) &= \frac{\epsilon}{1-\epsilon}(\dot{D}_{ij}^{MWM}(t) - \dot{D}_{ij}^{BR}(t)), \\ \dot{D}_{ij}^{BR}(0) &= \lambda_{ij}, \end{aligned} \quad (4.11)$$

$$\dot{D}_{ij}^{MWM}(t) = \lambda_{ij} \text{ if } Z(t)=0$$

First equation is from relation (4.9) repeated. Second equation is the result of (4.9) and the fact that departure rate at time zero is bounded by the arrival rate. Finally, from rate stability of MWM we can conclude the last relation.

It is easy to verify that  $\dot{D}_{ij}^{BR}(t) = \dot{D}_{ij}^{MWM}(t) = \lambda_{ij}$  is the only solution for  $t \geq 0$  that satisfies all the equations above, and therefore for all fluid model solutions, we have  $Z(t) = 0$ . ■

Next, we use stability result proved in theorem 4.2.1 to prove that the original randomized algorithm 4.1.1 is also rate stable.

**Lemma 4.2.3** *TASS scheduling algorithm is efficient.*

**Proof of lemma 4.2.3:** Proof is very similar to proof of theorem 4.2.1, so we skip the details. First, we need to derive the additional fluid model relations that describe TASS algorithm. Suppose that  $H = \{m_1, \dots, m_{k(n)}\}$  be the set of matchings with higher weight than  $m_{TASS}(n-1)$  at time  $n$ , where  $m_{TASS}(n-1)$  is the matching used at time  $n-1$ . Let  $\eta_m(n)$  be the binary variable that is one if we randomly select matching  $m$  and use it at time  $n$  and  $P_m$  is a policy that uses matching  $m$ . We can write,

$$t_\pi^{TASS}(n) = \eta_\pi(n)t_\pi^{P_\pi}(n) + \left(1 - \sum_{\pi' \in \Pi} \eta_{\pi'}(n)\right) t_\pi^{TASS}(n-1) \quad (4.12)$$

Next, we sum up overall matchings  $\pi$ ,

$$\sum_{\pi \in \Pi} t_\pi^{TASS}(n) = \sum_{\pi \in \Pi} \eta_\pi(n)t_\pi^{P_\pi}(n) + \left(1 - \sum_{\pi' \in \Pi} \eta_{\pi'}(n)\right) \sum_{\pi \in \Pi} t_\pi^{TASS}(n-1) \quad (4.13)$$

In order, to derive the fluid model equivalent of 4.13, we have to first characterize  $\eta_\pi(n)$ .  $\text{Prob}[\eta_\pi(n) = 1] = \epsilon > 0$  if  $\pi \in H$  and it is zero otherwise. Similar to BR algorithm case in relation 4.8, we can derive the fluid model equation for TASS algorithm,

$$(1 - k(t)\epsilon) \sum_{\pi \in \Pi} \dot{t}_\pi^{TASS}(t) = \epsilon \sum_{\pi \in H} t_\pi^{P_\pi}(n) - k(t)\epsilon \sum_{\pi \in \Pi} t_\pi^{TASS}(t) \quad (4.14)$$

$k(t)$  is the number of matchings that has a higher weight than the matching used at time  $t$ . Similar to the *BR* algorithm, for the *TASS* algorithm the following differential equation can be added to the basic fluid model equations,

$$\ddot{D}_{ij}^{TASS}(t) = \frac{\epsilon}{1-k(t)\epsilon} \left( \dot{D}_{ij}^{P_{m_1}(t)}(t) + \dots + \dot{D}_{ij}^{P_{m_k(t)}(t)}(t) - k(t)\dot{D}_{ij}^{TASS}(t) \right),$$

$$\dot{D}_{ij}^{TASS}(0) = \lambda_{ij}. \tag{4.15}$$

$\dot{D}_{ij}^{P_{m_1}(t)}(t), \dots, \dot{D}_{ij}^{P_{m_k(t)}(t)}(t)$  represents matchings that their weight is larger than  $\dot{D}_{ij}^{TASS}(t)$ . Substituting the first two equations of (4.11) with (4.15) and following the same argument as in theorem (4.2.1), we can show that  $\dot{D}_{ij}^{TASS}(t) = \dot{D}_{ij}^{MWM}(t) = \lambda_{ij}$  is the only solution that satisfies the fluid model differential equation. Therefore, *TASS* algorithm is also efficient. ■

The methodology that is used in lemma 4.2.3 is quite general and can be used to prove the stability of the randomized algorithms that will be introduced in the following sections.

### 4.3 General Architecture

In this section, we introduce the general architecture that will be used in all of self-randomized scheduling algorithms of this chapter. The scheduler structure is shown in Figure 4.1 and it is divided into two stages. First stage has a memory bank that buffers  $K$  previously used matchings. The first stage also contains the circuitry that selects a candidate matching out of the  $K$  matchings. The selection module computes weight of all  $K$  contending matchings in parallel, and selects the one with highest weight. Weight of a matching is summation of its link weights. The candidate matching is passed to the second stage.

Randomized scheduling algorithms usually store and process the matching that is used in the previous time slot; we have extended this idea by taking into account

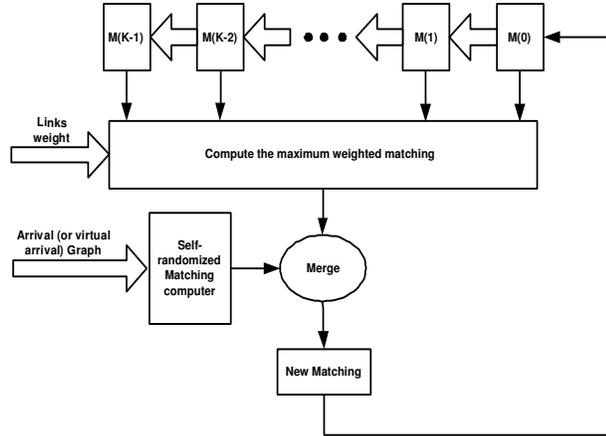


Figure 4.1: The generic block diagram adopted for self-randomized schedulers.

matchings that are used in  $K$  previous time slots. Memory or state notion plays a critical role in the stability of randomized schedulers. Saving the previous matching and using it as a new candidate is the main reason for the stability of Tassiulas algorithm, and for that reason it has been retained in all proposed self-randomized algorithms.

The concept of maintaining  $K$  previous matchings is also in accordance with some of the proposed deterministic schemes. It is proved in [6] that fixed serving rates can be obtained, using a periodic sequence of matching matrices for scheduling. The rate matrix is decomposed to a sequence of permutation matrices, that should be repeated in a periodic form. Based on the same result, paper [24] has proposed a rate provisioning scheme that uses a weighted fair queueing scheduler to select one matching from a fixed set of matchings. Increasing the memory order of the system enables us to reuse the same set of matchings, and can improve the performance, specially if we intend to do rate provisioning.

Main functions of the second stage are, to generate a matching in the self-randomized block, and to merge that matching with the candidate matching out

of the first stage. The end result of the merging block is the scheduling matching for that time slot. This matching is also passed to the first stage and is stored in the memory bank for future use in subsequent time slots.

The overall quality and performance of the scheduler is very much dependent on the procedure that is used to generate the self-randomized matching. Self-randomized matching is generated based on the arrival of cells, and/or virtual cells (tokens) arrival pattern for QoS provisioning. We use some simple techniques for that purpose, that will be discussed later.

In this architecture, scheduling encompasses weight calculation for  $K$  matchings at the first stage, and one merging at the second stage. This structure is based on the fact that weight of a matching can be simply calculated in hardware, where as merging of two matchings is not a simple scalable task for hardware.

The self-randomized scheduling algorithms that we introduce in the rest of the paper are based on the general architecture that is described here. Each of these algorithms is completely defined by two main characteristics,

1. The weight function,
2. The self-randomized matching generator.

The weight function specifies the fundamental features and objectives of the algorithm, while the self-randomized matching generator determines how well and fast scheduler responds and adapts to the changes in the arrival pattern.

## 4.4 Backlogged weight function

In this section, we introduce B1, and B2, two self-randomized scheduling algorithm that their weight function is number of backlogged cells. This is the most common

weight function for scheduling in switches [29] [31] [36], even though it has its own shortcomings. From a theoretical stand point, most of the stability results are originally derived for backlogged weight function. From a practical stand point, it is simple to compute and update weights of links. Moreover, in terms of average delay and number of backlogged cells, this weight function provides a good performance.

However, backlogged weight function has its own drawbacks; first of all it is not starvation free, i.e., it is possible that some finite number of cells experience infinite delays [29]. Secondly, it lacks a mechanism to provide QoS. In section 4.5, we introduce an alternative weight function that resolves these problems.

#### **4.4.1 B1 : Self-Randomized Based on Instantaneous Arrivals**

B1 is similar to the SERENA algorithm that is introduced in [19]. We use it as a bench mark, and as a means to discuss some of the features of the self-randomized algorithms. As mentioned before, number of backlogged cells is the weight function of this scheme. The arrival graph is the input to the self-randomized matching computer block. The arrival graph at time  $t$  is a bipartite graph that has a link between node  $i$  and  $j$  if there is an arrival at time  $t$  in  $i$  destined for  $j$ . The self-randomized matching generator extracts a matching graph out of the arrival graph. The extracted matching will be merged with the candidate matching out of the first stage (Fig.4.1).

Arrival pattern is not a matching since there can be multiple arrivals with the same destination. Self-randomized matching block, extracts a matching from this graph by selecting the highest weight link for every output node. This process can be done in parallel for all output ports. The exact same procedure is used in

SERENA [19].

**B1 characterization:**

- **Weight function:** Buffered cells for each link.
- **Self-Randomized Generator:** Works based on the recent cell arrival graph.

*B1* is a simple extension of the SERENA algorithm, that uses  $K$  previous matchings rather than only one previous matching.

#### 4.4.2 B2 : Self-Randomized Based on Total Arrivals

In *B1*, every new arrival gets one chance to be included in the self-randomized matching. However, links may lose their chance, either to other new arrivals in the process of generating the self-randomized matching, or in the merging phase. If a link loses its chance, it should wait for another new arrival to get another opportunity. Therefore, even under low utilization, it is possible that some of the cells incur high delays. In fact, the situation is worse for very low arrival rates, since the average waiting time for the next arrival is higher. We intend to resolve this problem in *B2*.

*B2* uses the total arrival graph as the source for extraction of self-randomized matching. In the total arrival graph there is a link for all previous cell arrivals that are not scheduled yet. Total arrival graph evolution is based on link insertion, and ejection rules in every cell time.

- **Insertion :** Any link with new arrival that is not in the total arrival graph is added to the total arrival graph.
- **Ejection :** Links that are scheduled are removed from the total arrival graph.

At every cell time a matching is extracted from the total arrival graph. The extraction process is a trivial extension of the process that is used in B1. Similar to B1, for every output port the link with largest weight is selected from the total arrival graph. In B2, there may be some contention at the input ports between the selected links. In the next step, if there are multiple selected links connected to the same input port, the one with highest weight is selected. The outcome of this two step selection process is a matching that is used as the self-randomized matching.

Only those links that are included in the final scheduling matching are ejected from the total arrival graph. Therefore, a new arrival link is continuously considered for the self-randomized matching, until it is scheduled. We expect that the delay performance of B2 for the links with low arrival rates be better than B1, because they get more chances to be scheduled.

**B2 characterization:**

- **Weight function:** Buffered cells for each link.
- **Self-Randomized Generator:** Works based on total cell arrival graph.

We end this section by stating the stability theorem for the *B1* and *B2* algorithms introduced here.

**Corollary 4.4.1** *B1 and B2 schedulers are efficient.*

**proof of corollary 4.4.1:** *B1* and *B2* can be both considered as special cases of the *TASS* scheduling algorithm. Note that in *B1* and *B2* either the previous time slot matching is selected or another randomly selected matching with a higher weight is selected. The mechanism of selecting the randomized matching is more complicated, but still there is a positive probability  $\epsilon > 0$  that the MWM is

selected. Therefore, the basic properties of the *TASS* algorithm that is used in proof of lemma 4.2.3 is maintained in these algorithms and with the same reasoning we can prove that they are efficient. ■

### 4.4.3 Simulation Results

In this section, we use simulation results to study and discuss the delay performance of *B1* and *B2* scheduling algorithms. Simulations are done for a 16x16 input-buffered switch, and the memory length for the buffered matchings  $K$  are set to 1, 16, and 32. The arrival process for all links are i.i.d Bernoulli, and the arrival rate for the link between input  $i$  and output  $j$  is  $\lambda_{ij}$ .

Three different load distribution models, uniform, diagonal, and log-diagonal between the links are studied. The simplest one is uniform, where the total load  $\rho$  of every input port is uniformly distributed among its outgoing links. The diagonal is the most compact form, where the arrival rates are  $\lambda_{ii} = 2\rho/3$  and  $\lambda_{|i+1|} = \rho/3$ ,  $i = 1, \dots, N$  and 0 for all other links, where  $|i| = (i \bmod N)$ . Therefore, there are only two non-zero elements in every row. In the log-diagonal model  $\lambda_{i|j|} = 2\lambda_{i|j+1|}$ ,  $i = 1 \dots, N, j = i, \dots, N + i$ .

The simulations results for *B1* are shown in Figures 4.2, 4.3, 4.4, for Diagonal, Log-diagonal and Uniform destination distributions respectively. In all cases, increasing memory size  $K$  improves the delay performance of the switch. Improvement is more significant for uniform distribution. The uniform distribution matrix, has the maximum entropy, and correlation between subsequent arrivals is minimal. Therefore, for low arrival rates, the previous matching is usually not a good candidate for the present time slot, and having  $K$  alternative matchings significantly increases probability of selecting a good matching.

The other interesting and non-desirable phenomena is having large delays at very low rates. This is to some extent, because of the self-randomized matching generator that is used. Note that new links are introduced to the system through the self-randomized graph, and a link is introduced to the self-randomized graph, only when there is a new arrival for that link. Therefore, if a link is not included in the self-randomized matching after its cell arrival, it should wait for the next cell arrival to get another chance. The extensive delay for low utilizations incurs more dramatically for log-diagonal and uniform load distribution, since the arrival rate for some of the links are very low under these distributions. This behavior is not clear in [19] results at the first glance, since authors have reported and plotted average queue length, rather than the average delay in their simulation results. Even though delay and queue length are directly related by Little's theorem, delay is more tangible, common and relevant for practical applications.

The same set of simulations are repeated for B2 in Figures 4.5, 4.6, 4.7, for diagonal, log-diagonal, and uniform load distributions respectively. For the diagonal traffic the results are very similar to the B1 algorithm. In both cases, increasing the matching memory size results in considerable improvement of delay performance. For the log-diagonal distribution, B2 outperforms B1 delay performance for low utilizations. The average delay for B1 is around 40 cell times, where as it is around 20 for B2.

There is still a minimum in the delay curve of B2, but its intensity is decreased. Recall that in B2 a link with new arrival is continuously introduced to the self-randomized block, until it is scheduled. Therefore, new arrivals scheduling delay is in general less than B1. This, specially improves the delay performance of the very low rate links.

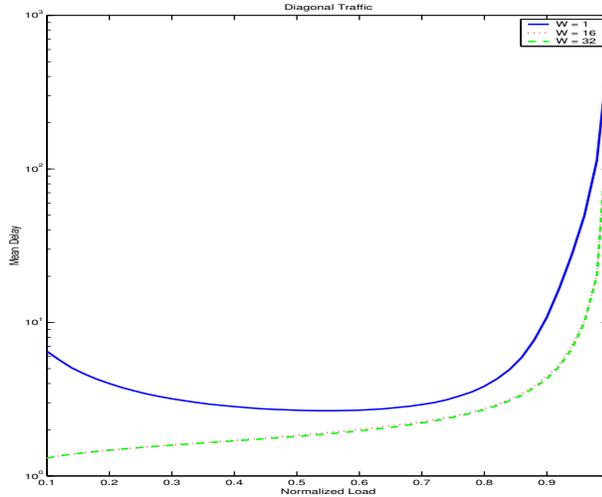


Figure 4.2: Mean delay for B1 under diagonal traffic.

For the uniform traffic, advantage of employing memory for the matching is considerable, and delay performance is evidently enhanced by increasing the memory size from 1 to 16 and 16 to 32.

In summary, we can conclude that B2 outperforms B1 in cases where traffic load is distributed among a larger number of ports, and specially when we have some links with very low rates. The advantage of introducing memory for previous matching is clear and it improves the performance, since we have more alternative matchings to select from.

## 4.5 Max-min fair scheduling weight function

In this section, we introduce a new class of self-randomized scheduling algorithms that use a set of token values for the link weight function. Token based weight function enables us to do bandwidth allocation among different flows based on their reserved rates. In the context of prioritized max-min fair scheduling algorithms

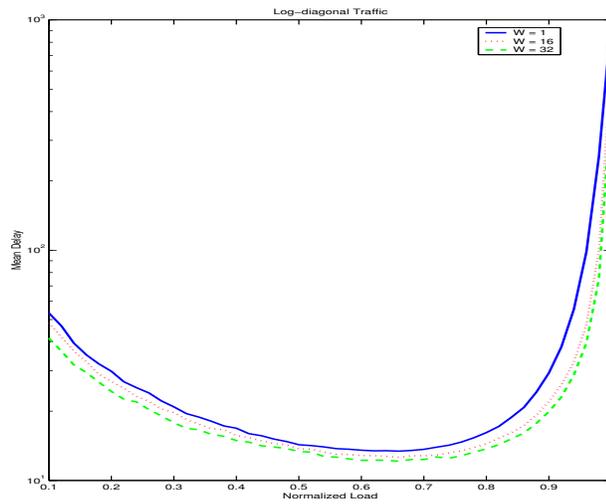


Figure 4.3: Mean delay for B1 under log-diagonal traffic.

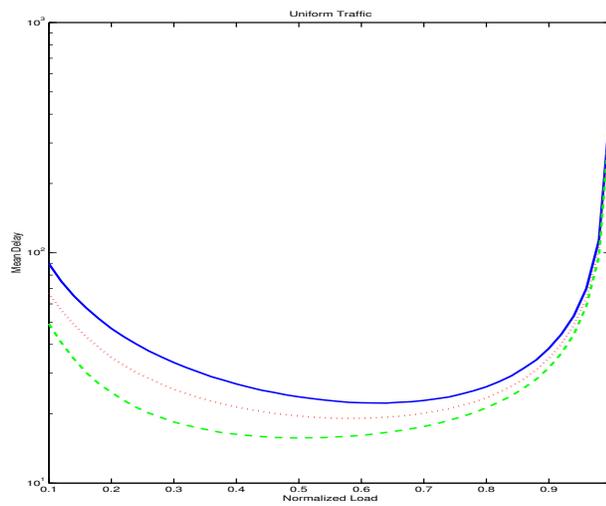


Figure 4.4: Mean delay for B1 under uniform traffic.

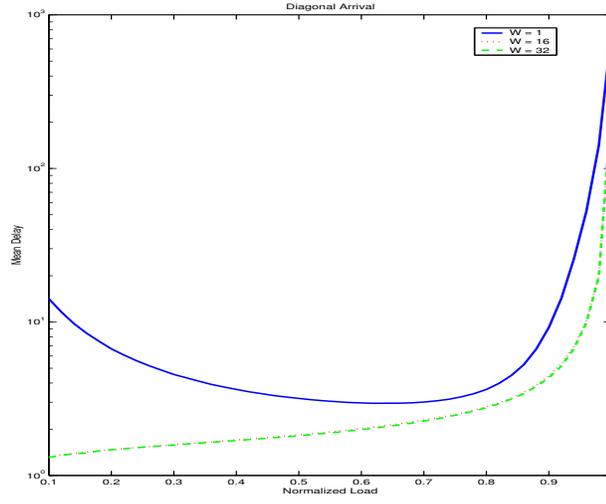


Figure 4.5: Mean delay for B2 under diagonal traffic.

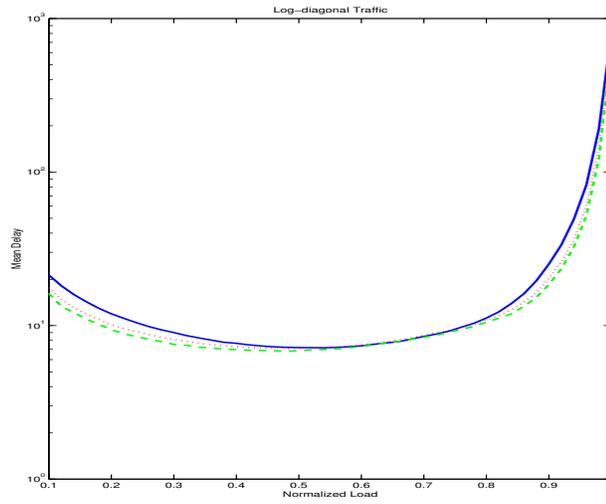


Figure 4.6: Mean delay for B2 under log-diagonal traffic.

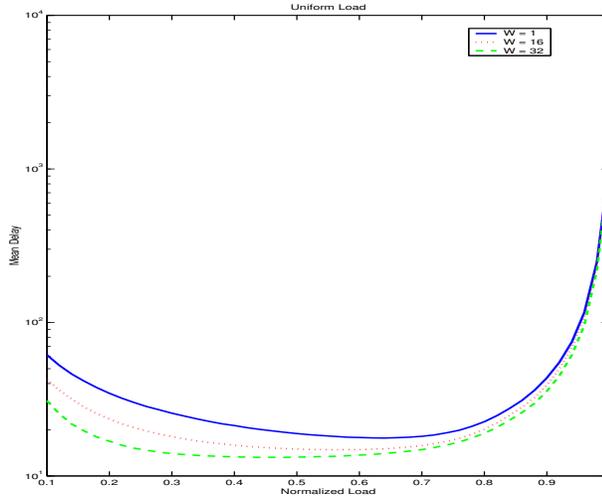


Figure 4.7: Mean delay for B2 under uniform traffic.

each flow  $i$  has a normalized reserved rate or priority,  $w_i$ . Greater the reserved rate of a flow, better the service it should receive, subject to bandwidth availability. A bandwidth allocation is max-min fair if it is not possible to increase the bandwidth of any flow  $i$ , without hurting another flow  $j$  for which  $g_j(n)/w_j \leq g_i(n)/w_i$ ,  $g_i(n)$  is the bandwidth allocated to flow  $i$  at time  $n$ .

Tassiulas and Sarkar [38] have proposed a token based deterministic max-min fair scheduling algorithm that works based on MWM algorithm. Weight of every flow depends on an estimate of the max-min fair bandwidth of the flow, previous service received by the flow, and traffic demand of the flow. The proposed algorithm achieves max-min fair objective for any arbitrary set of flows and graphs. More specifically, their result is general and is valid for an arbitrary network graph topology and a set of flows traversing arbitrary paths in the network. We will use and extend the same concepts in the context of self-randomized scheduling algorithms, and introduce a new class of randomized scheduling algorithms that have rate provisioning capabilities for input buffered switches.

Rate provisioning is one of the important requirements for switch fabrics scheduling algorithms that has not been considered in the context of randomized scheduling algorithms. Token based weight functions enable us to divide the available bandwidth among backlogged queues according to the reserved rates of the links in a max-min fair fashion. In this way, we can provision the serving rate of the links and guarantee their quality of service.

The first question that should be answered is feasibility of a set of normalized reserved rates  $w_i$ . For input buffered switches, a necessary condition for feasibility is that sum of normalized reserved rates of all flows sharing a node should not exceed 1. Paper [38] has shown that this is also the sufficient condition for feasibility when the network graph is bipartite, which is the case for input-buffered switches.

The original algorithm proposed in [38] distributes tokens between flows according to the following procedure,

**Algorithm 4.5.1** (*Deterministic Max-min Fair*):

1. *Each node allocates service tokens to the links connected to it, in a weighted round-robin (WRR) fashion. The service token allocation mechanism is described later.*
2. *For each link the total number of tokens that are allocated to it, in each of its two end nodes, are maintained. The minimum of the two token buckets equals link's weight that is used in matching if that link is backlogged. For non-backlogged links weight is zero.*
3. *Whenever a link is scheduled and one cell is served, one token is deducted from each of its two token buckets.*

For the deterministic max-min fair scheduling algorithms it is proved [38] that in any interval of length  $t$ , the total number of served cells for a link  $(i, j)$  differs from  $g_{ij}t$  by at most a constant  $\kappa_{ij}$ . Note that these results are based on the assumption that the

$$g_{ij} \leq \lambda_{ij}. \quad (4.16)$$

For the randomized max-min scheduling algorithms we assume that 4.16 holds, and we consider the following definition for the randomized max-min fair scheduling algorithm.

**Definition 4.5.2** *A randomized scheduling algorithm is max-min fair if with probability one*

$$\lim_{n \rightarrow \infty} \frac{D_{ij}(n)}{n} = g_{ij} \quad i, j = 1, \dots, N \quad (4.17)$$

for any arrival process that satisfies (4.16).

We will introduce three different self-randomized max-min fair scheduling algorithms that work based on three different token allocation methodologies. The proposed token allocation mechanisms are slightly different from the original scheme proposed in [38].

The same general architecture that is given in Fig. 4.1 is employed here too. Link weights are tokens and are determined based on the procedure given above. Different algorithms are characterized through their token allocation mechanism, and the self-randomized matching generator scheme that is used in the second stage of the general architecture.

### 4.5.1 A1 : Self-Randomized Based on Instantaneous Token and Cell Arrivals

The service allocation mechanism in [38] uses a WRR scheduler to select one of the eligible flows to get a token. A link between node  $i$  and  $j$  is eligible to receive a token in node  $i$ , if it meets the following eligibility conditions.

1. **Queue condition:** Number of queued cells for the link is more than the number of tokens.
2. **Token condition:** Tokens of link  $(i, j)$  in  $i$  does not exceed its tokens in  $j$  by more than a constant  $W$ .

In A1, we only check the token condition for eligibility. Basically, for A1 once a flow becomes backlogged it is eligible to acquire token in node  $i$  if the token condition is met. However, when all cells of a link are scheduled, its tokens are reset to zero.

In the original scheme, due to queue condition, tokens always lag number of backlogged cells. This can result in starvation or excessive delay for some of the cells. For instance, consider a link with a high reserved rate. Intuitively, we expect that the cells belonging to that link experience low delay. However, regardless of the flow's reserved rate, if there is only one cell in the queue of that link its weight does not exceed 1, and it can experience a very high delay. In contrast, in A1 this link accumulates tokens with its max-min fair rate, even if there is only one buffered cell for that flow. Hence, weight of backlogged links increases and ultimately they will be scheduled.

For full characterization of A1 we need to specify the self-randomized generator mechanism too. Self-randomized scheme that is used in A1 is based on instantana-

neous cell and token arrivals. It is similar to B1, but it works based on the arrival of both cells and tokens. By adding the token arrival, every link gets more chances to be included in the self-randomized matching, and the higher the rate of the link, the higher is its chance. This enhances delay performance of the scheduling algorithm, as it is illustrated in the simulation results later.

**A1 characterization:**

- **Weight function:** Number of token for each link with token condition for eligibility test.
- **Self-Randomized Generator:** Works based on the recent token and cell arrival graph.

To prove that A1 is max-min fair, we first prove that A1 allocates enough tokens to links. The following lemma proves that number of tokens for a link differs from its max-min fair share by a constant.

**Lemma 4.5.3** *If the threshold  $W$  is sufficiently large, then the number of tokens generated at each end of a link  $(i, j)$  in any interval of length  $t$  differs from  $g_{ij}t$  by at most a constant  $\nu_{ij}$ , where  $g_{ij}$  is the max-min fair rate of  $(i, j)$ , and  $\nu_{ij}$  is a constant which does not depend on the length of the interval.*

**Proof of lemma 4.5.3:** We will prove the theorem under the assumption that all queues have packet for transmission all the time. The generalization for the case when all queues may not have packet all the time is straightforward. Recall that the token allocation scheme is deterministic and very similar to the original algorithm 4.5.1. The same result is proved in [38] for the original algorithm when both token and queue conditions are tested for eligibility. However, in the A1 only the token condition is used.

Note that if the number of backlogged cells in each queue is arbitrary large then for the original algorithm the queue condition does not have any impact on the token generation process. Therefore, A1 can be considered as a special case of the original algorithm 4.5.1 when for all queues number of backlogged cells is larger than the number of tokens. Hence, this lemma can be considered as a special case of the more general result proved in Theorem 3 of [38].

■

**Theorem 4.5.4** *A1 is a randomized (self-randomized) max-min fair scheduling algorithm.*

**Proof of theorem 4.5.4:** In lemma 4.5.3, we proved that tokens are generated with max-min fair rate. Moreover, whenever a link that has token is scheduled, at least one of its tokens is consumed. Therefore, we can view the tokens as virtual cells that are arriving with max-min rate, and are served (consumed), whenever a link is scheduled. If we view tokens as virtual cells, A1 functions as a self-randomized scheduling algorithm with virtual cell (token) backlogged weight function. Since A1 serves at least one virtual cell whenever a link is scheduled, token serving rate of a link under A1 is at least as large as its scheduling rate. This is exactly similar to B1, with the exception that B1 always serves at most one cell when it is scheduled. Since B1 is rate stable (corollary 4.4.1), we can conclude that A1, with respect to token arrivals, is rate stable. Hence,

$$\lim_{n \rightarrow \infty} \frac{D_{ij}(n)}{n} \geq g_{ij} \quad i, j = 1, \dots, N \quad (4.18)$$

So far we have proved that serving rate of every flow is at least equal to its max-min rate. We still need to prove that it is exactly equal to the max-min rate. Consider that for a link  $(i, j)$ , serving rate is higher than its max-min rate. By

definition, every link  $(i, j)$  has at least one bottleneck node [38]. Node  $i(j)$  is a bottle neck for link  $(i, j)$ , if sum of the max-min rate of all links connecting to that node is 1, and link  $(i, j)$  normalized share  $g_{ij}/w_{ij}$  is maximum amongst all links connected to  $i(j)$ . Without loss of generality, assume that  $i$  is the bottleneck node for link  $(i, j)$ . If  $(i, j)$  is served at a higher rate than  $g_{ij}$  then by definition of bottleneck node, there is another link  $(i, j')$  that should be served at a lower rate than its max-min rate, and this contradicts (4.18) for  $(i, j')$ . Therefore every link is exactly served with its max-min rate. ■

#### 4.5.2 A2 : Self-Randomized Based on Total Token and Cell Arrivals

Similar to A1, A2 uses number of tokens as the weight function for links. Tokens are generated and distributed similar to A1. However, for self-randomized generator block, we use the same idea that is used in B2. In B2, we introduced the concept of total cell arrival graph. The idea was to persistently present new arrivals to the self-randomized block, until they are scheduled. In general, this expedites the process of scheduling a new arrival, specially at low rates. We employ the same idea in A2, by maintaining total cell, token arrival graph in the self randomized matching computer block. The self randomized matching generator, extracts a matching from the total cell, token graph at every time slot. Similar to the total arrival graph introduced in B2, total cell, token arrival graph evolves by insertion and ejection rules,

- **Insertion** : For new cell and token arrivals, if the link is not in the total cell token graph, it is inserted to the graph.

- **Ejection** : All scheduled link are removed from the total cell, token arrival graph.

A2 outperforms A1 when the utilization is low, because it expedites scheduling of low rate links. However, for high utilization, both algorithms perform poorly. There are two intuitive reasons for low performance of these algorithms in high rates.

The first reason is related to the total cell and token arrival graph. The main objective of the total graph is to provide a small set of candidate links for the self-randomized block. At high utilizations the probability of a link being backlogged is very high, and consequently as soon as a link acquires a token, it is inserted into the Total graph. This results in a very populated total graph, that does not serve the purpose of providing a good, small set of candidate links for scheduling.

The second reason is related to the weight function. Weight function sets priority for scheduling of the links. At high rates it is vital to give high priority to the congested links that has a large number of backlogged cells. Weight function of A1 and A2 is merely based on the max-min rate which does not necessarily reflects the actual number of backlogged cells. These problems are addressed in A4.

### **4.5.3 A4 : Self-Randomized Based on Total Link Weight and Cell Arrivals**

A4 is introduced to compensate for some of the problems that are observed in A1 and A2. The weight function in A4 is different from A1 and A2. In A4, weight of a link does not exceed its queue length by more than a constant  $B$ . Number of tokens still grow, regardless of the queue length, but it is not reflected directly into the link weight. However, as soon as a new cell arrives, weight of the link increases

if there are enough tokens in both node buckets. Note that this is slightly different from the algorithm proposed in [38], where number of tokens generated in a node for a connected link was limited to the number of backlogged cells.

Similar to A1, we prove that A4 allocates tokens to the links according to their max-min rate. The following lemma proves that number of tokens for a link differs from its max-min fair share by a constant.

**Lemma 4.5.5** *Under A4, for an arbitrary threshold  $B$ , and a sufficiently large threshold  $W$ , the number of tokens allocated to a link  $(i, j)$ ,  $F_{ij}(n)$ , satisfies the following asymptotic relation,*

$$\lim_{n \rightarrow \infty} \frac{F_{ij}(n)}{n} = g_{ij} \quad i, j = 1, \dots, N \quad (4.19)$$

**Proof of lemma 4.5.5:** A4 uses the same token generation mechanism that was used in A1, and A2. Hence, based on lemma 4.5.3, we know that in every time interval  $n$ , total number of tokens generated at each end node for link  $(i, j)$ , differs from  $g_{ij}n$ , by a constant  $\nu_{ij}$ . However for A4,  $F_{ij}(n)$ , total number of assigned tokens to the weight of a link is less than the total number of arrived cells plus  $B$ ,

$$F_{ij}(n) \leq A_{ij}(n) + B \quad i, j = 1, \dots, N. \quad (4.20)$$

Hence,

$$\lim_{n \rightarrow \infty} \frac{F_{ij}(n)}{n} \leq \lambda_{ij}(n) \quad i, j = 1, \dots, N. \quad (4.21)$$

Based on (4.16), number of generated tokens at end nodes of a link for a flow, is asymptotically less than or equal to number of arrived cells. Therefore, as  $n$  tends to infinity weight of a link is limited by the number of generated tokens and not the arrival rate, i.e.,

$$\lim_{n \rightarrow \infty} \frac{F_{ij}(n)}{n} = g_{ij} \quad i, j = 1, \dots, N \quad (4.22)$$

Besides the weight function, A4 also uses a slightly different mechanism to generate the self-randomized matching. Recall that A2 uses the total cell, token arrival graph as the basis for derivation of the self-randomized matching. In the previous section, we explained that at high rates this graph becomes so dense that it would not provide a good set of candidate links for the self-randomized graph. In A4, instead of working with total cell and token arrival graphs, we consider the total cell arrival and weight increase graph. Basically, a link is inserted to the graph if there is a new arrival or there is an increase in weight of that link. Consequently, even at very high rates, links are not inserted into the self-randomized computing block excessively.

**A4 characterization:**

- **Weight function:** At each ending node tokens are generated for a link with token condition for eligibility set. A link weight is minimum of it tokens and number of backlogged cells plus a constant  $B$ .
- **Self-Randomized Generator:** Works based on the total cell arrival, link weight increase graph.

Before ending this section, we state that A4 algorithm is a max-min randomized (self-randomized) scheduling algorithm.

**Theorem 4.5.6** *A4 is a randomized (self-randomized) max-min fair scheduling algorithm.*

**Proof of theorem 4.5.6:** Proof is very similar to theorem 4.5.4, so we skip the details. The key point is in lemma 4.5.5, where we proved that link weight evolves with max-min fair rate. On the other hand, whenever a link that has

token is scheduled at least one of its tokens is consumed and weight of the link will be decremented by one. Hence again, link weight can be considered as virtual cell arrival process, and the same reasoning given for A1 in theorem 4.5.4 can be repeated here. ■

#### 4.5.4 Simulation Results

In this section, we present two sets of simulations. First set presents the delay performance of the three token based self-randomized schedulers introduced here. In the second set, we demonstrate that A4 serves links according to their max-min fair rate, and furthermore it adapts to the changes in max-min rate very fast.

##### Delay Performance

We start with A1 scheduler. Experiments setup is similar to what we used for backlogged weight function schedulers. Results are given in Figures 4.8, 4.9, 4.10. At low to moderate utilizations, delay performance for diagonal and uniform load distributions are better than B1. Specially for low utilizations A1 performs better. The undesirable phenomena of having high delays at very low rates is also eliminated for uniform and diagonal traffic. Improvement at low rates is due to the fact that links are inserted to the candidate set for the self-randomized matching based on the token and cell arrival. Hence, low rate links are considered more often.

Recall that the weight function of A1 is customized to provide max-min fairness and not good delay performance, however good delay performance is achieved as a side product.

In A2 the input graph to the self-randomized computer is persistent, and therefore we expect a better delay performance, specially for low utilization factors.

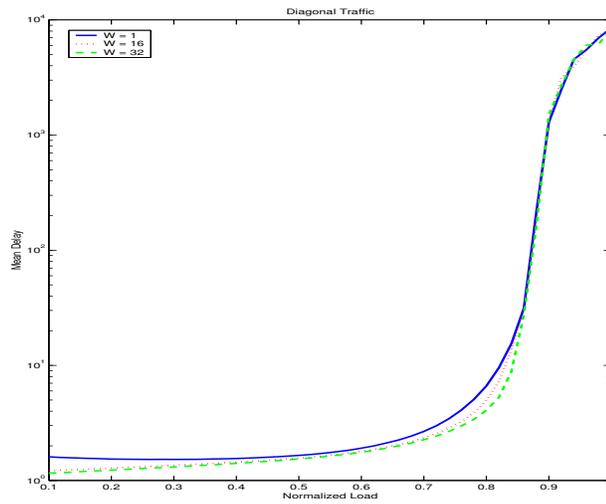


Figure 4.8: Mean delay for A1 under diagonal traffic.

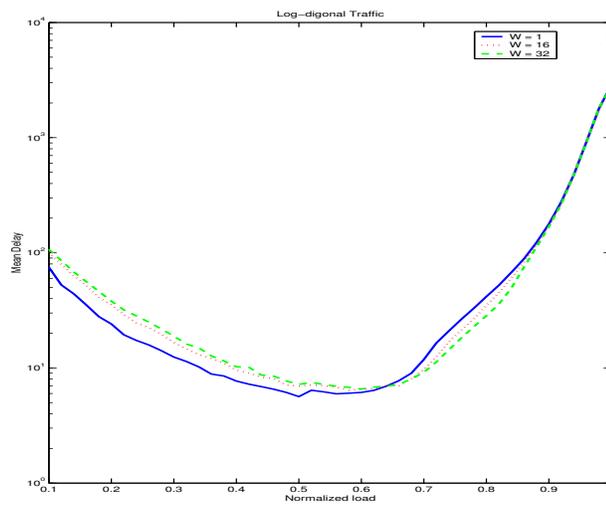


Figure 4.9: Mean delay for A1 under log-diagonal traffic.

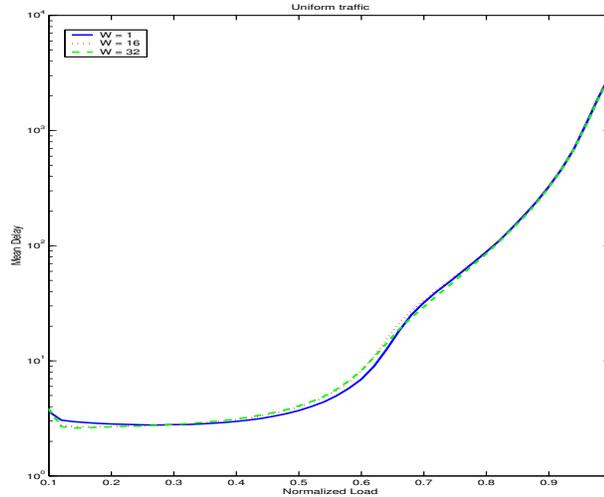


Figure 4.10: Mean delay for A1 under uniform traffic.

Simulation results are given in figures 4.11, 4.12, 4.13. In all cases we see a better overall performance specially for low rates. For the first time, even for log-diagonal traffic, the delay-throughput curve is uniformly increasing, which is the expected, desirable shape. The draw back of A2 is in higher utilization rates, where B1 and B2 outperform it.

In A4, link weight exceeds the number of backlogged cells by not more than a constant  $B$ . In this way, we intended to capture desirable features of both schemes, and have a good delay performance for the whole utilization range, and at the same time, maintain the max-min fairness property. A4 has the best overall delay performance for all examined load distributions, and for the total range of utilization.

The difference is better illustrated for log-diagonal traffic in low utilization. All other algorithms presented in this paper, have a poor delay performance for low utilizations, and up to some point, average delay keeps decreasing, until it reaches its minimum. However, throughput-delay curve for A4 is always increasing.

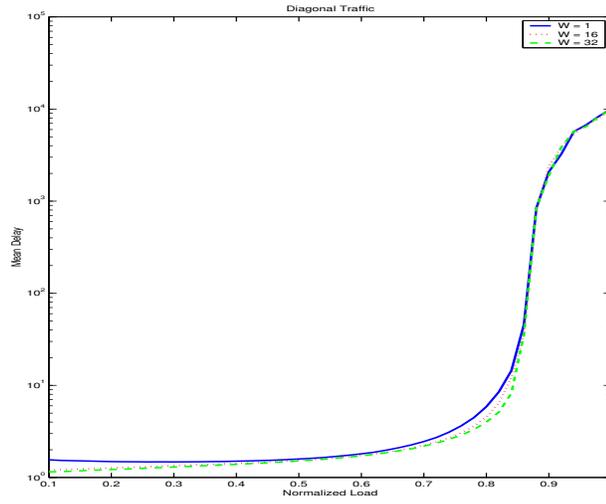


Figure 4.11: Mean delay for A2 under diagonal traffic.

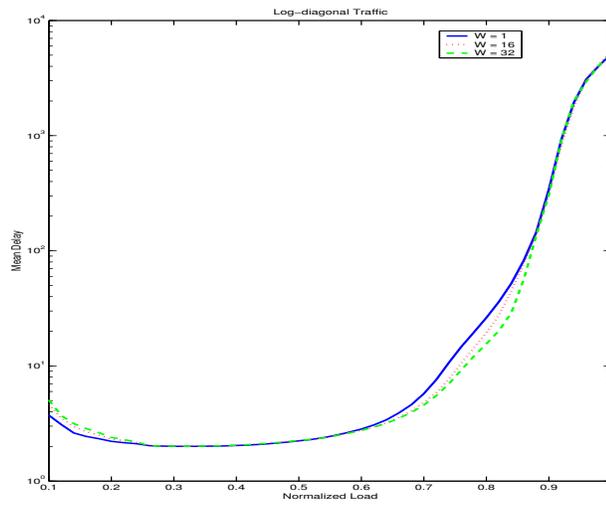


Figure 4.12: Mean delay for A2 under log-diagonal traffic.

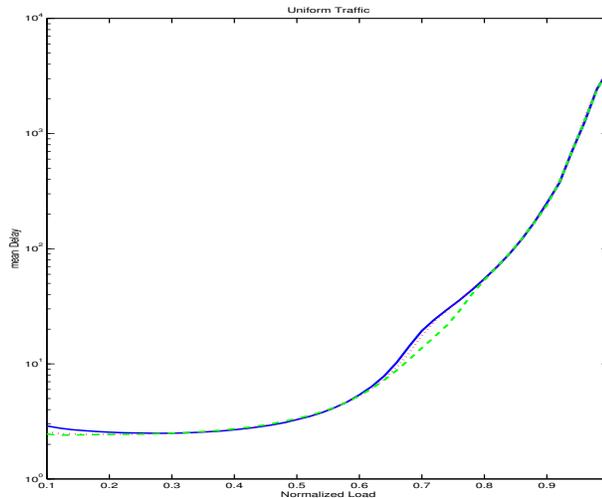


Figure 4.13: Mean delay for A2 under uniform traffic.

The adopted weight function has a positive effect on the performance. In token based schemes, links' weights increases at a rate which is proportional to the max-min rate of the links. Therefore, weight function operates proactively, prior to the cell accumulation in the buffers. Intuitively, links that have higher reserved rates are served faster, even if they are not the most congested buffers. This means that we have allowed flows to accumulate tokens, even before the cell arrivals, and consequently experiences lower delays, and buffer length.

However, for stability purposes, we need to maintain a balance and take into account the current buffer lengths. This becomes more crucial at higher utilization, as the number of congested links increases. In A4 the link weight can not exceed the number of buffered cells by a constant. In this way, A4 makes a balance between the two weight function; at low rates links weight is effectively equals number of tokens, where as at higher rates, it reflects number of backlogged cells.

The weight function enables A4 to have a dual behavior, it works based on the reserved rates, and provides max-min fairness, and at the same time takes the

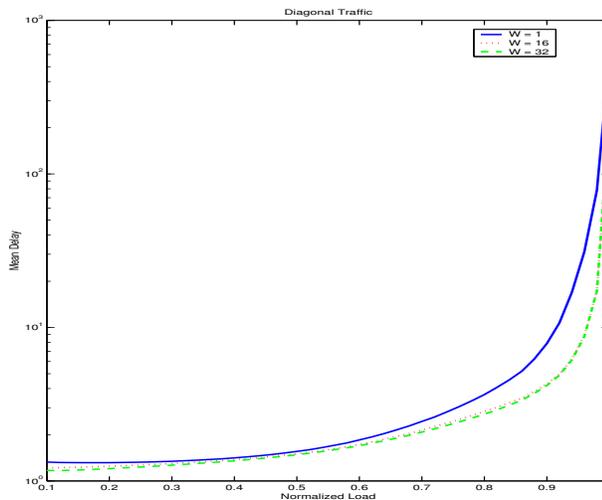


Figure 4.14: Mean delay for A4 under diagonal traffic.

buffers length into account. It is interesting that the dual behavior of A4 is also reflected in delay performance curves. For instance for the uniform traffic the delay curve slope changes considerably and gets lower around 0.7 utilization, as if the delay curve is combination of two different curves, one for lower rates based on the token weight function, and the other for higher rates based on the backlogged cells weight function.

The delay performance of all the proposed schemes are compared in figures 4.17, 4.18, 4.19. Generally, B1 and B2 performs better for high utilization, and A1 and A2 in low utilization. A4, due to its dual behavior, follows the curve of A1 and A2 for low utilization, and B1 and B2 in high utilization. In fact the performance of A4 is even better than A1 and A2 in low utilization. The difference is greater for log-diagonal traffic, where A4 is the only algorithm that does not have an intermediate minimum in the delay curve. Recall that log-diagonal load distribution is in a sense the most non-uniform load distribution among the examined distributions. On some links it has the lowest non-zero arrival rate, and at the same time has the

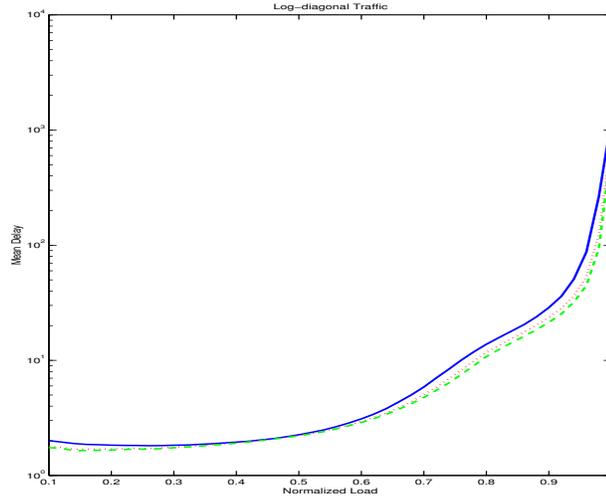


Figure 4.15: Mean Delay for A4 under log-diagonal traffic.

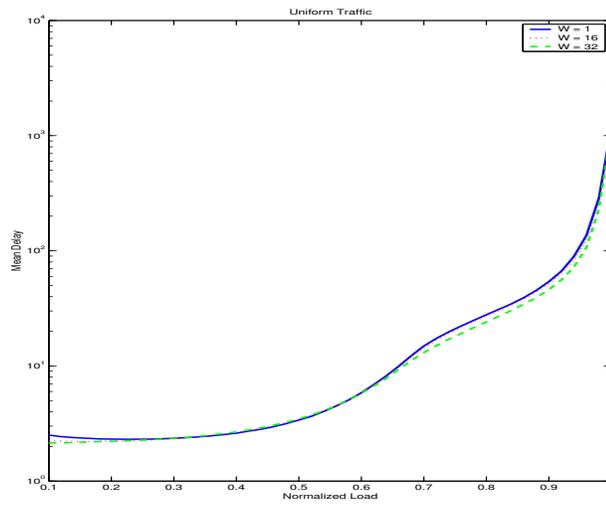


Figure 4.16: Mean Delay for A4 under uniform traffic.

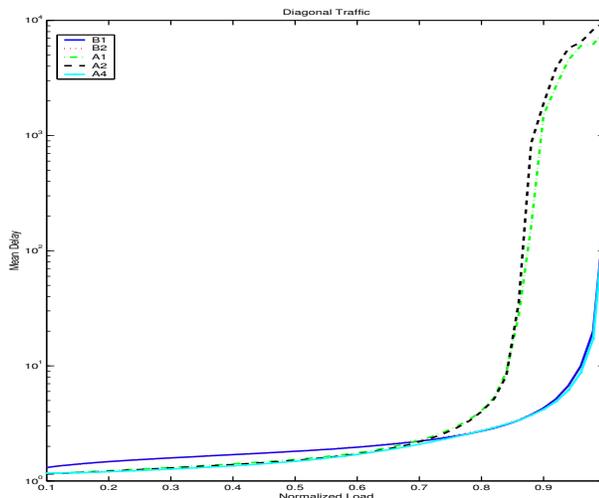


Figure 4.17: Mean delay for different schemes under diagonal traffic.

highest rate on some other links. The dual behavior of A4, enables it to perform well both for low and high rate links.

### Max-min fairness simulation for A4

In this section, we study the max-min fairness behavior of the proposed algorithm. We consider a 4x4 input-buffered switch fabric, with log-diagonal weight matrix. The average serving rate for all links are calculated for every 1000 time slots, and plotted.

We study how well the algorithm tracks the max-min fair rate, and how fast it adapts the serving rate to the changes in arrival pattern. We start the experiment with all queues backlogged, then in the middle of simulation we stop arrivals for links (1, 1), and (2, 2). The max-min fair rate, before and after the shut down are given below in rate matrices  $G_1$  and  $G_2$  respectively,

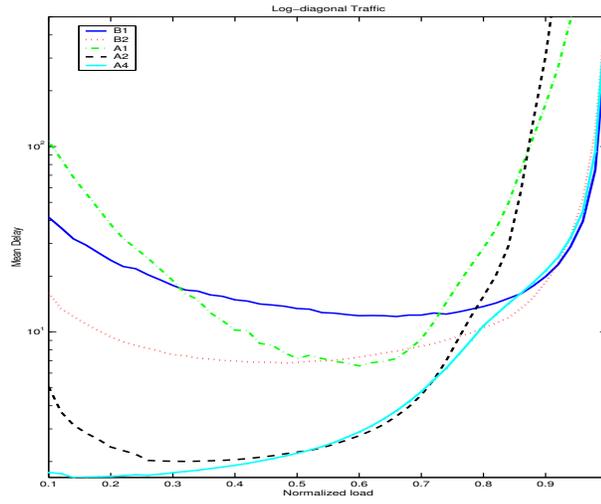


Figure 4.18: Mean delay for different schemes under log-diagonal traffic.

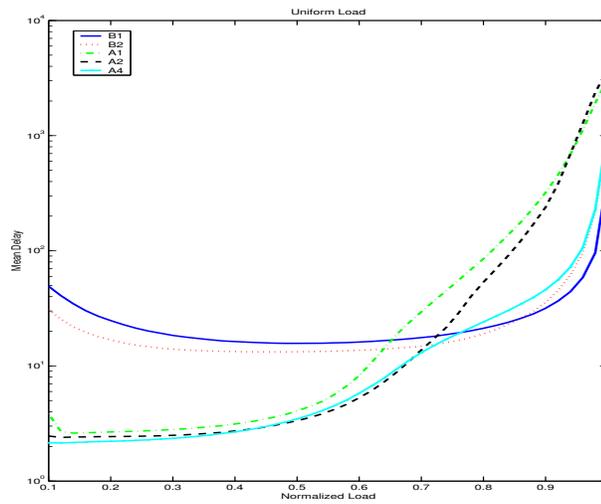


Figure 4.19: Mean delay for different schemes under uniform traffic.

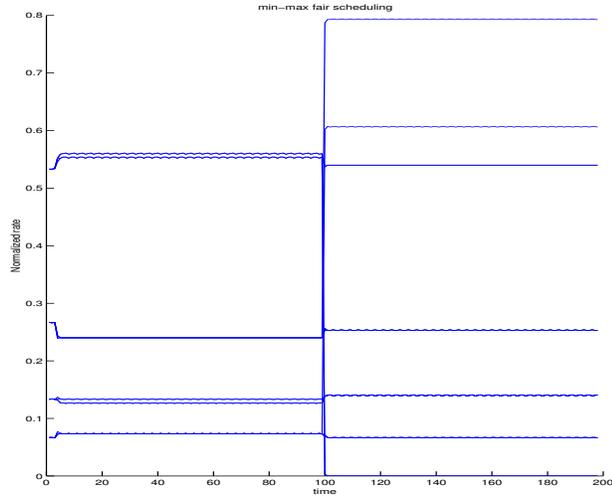


Figure 4.20: Serving rate adaptation to arrival pattern changes.

$$\begin{matrix}
 \begin{pmatrix} 0.533 & 0.267 & 0.133 & 0.067 \\ 0.067 & 0.533 & 0.267 & 0.133 \\ 0.133 & 0.067 & 0.533 & 0.267 \\ 0.267 & 0.133 & 0.067 & 0.533 \end{pmatrix} & 
 \begin{pmatrix} 0. & 0.8 & 0.133 & 0.067 \\ 0.6 & 0. & 0.267 & 0.133 \\ 0.133 & 0.067 & 0.533 & 0.267 \\ 0.267 & 0.133 & 0.067 & 0.533 \end{pmatrix} \\
 G_1 & G_2
 \end{matrix}$$

Simulation results are given in Figure 4.20. The arrival pattern changes at time frame 100 in the plot. Each time frame consists of 1000 cell times. The serving rate adapts to the new arrival pattern immediately. This experiment illustrates that serving rate of A4 scheme is very close to the max-min rates. Moreover, algorithm adapts to the changes of arrival pattern very rapidly.

# Chapter 5

## Summary and concluding remarks

In this dissertation we investigated the problem of scheduling in an input buffered crossbar based switch fabric. The switch fabric consists of  $N$  input and  $N$  output ports. At each input port, cells are buffered in separate virtual output queues (VOQ) corresponding to different output ports. The scheduling problem is equivalent to finding a matching between the input and output ports. The objective is to develop scheduling algorithms that have low complexity and high performance.

The performance metrics that we considered are latency, throughput and rate provisioning capability. The latency and throughput are common performance metrics for switch fabrics. We consider the rate provisioning as another vital requirement for next generation switch fabrics that in most of the proposed solutions is overlooked.

The rate based schedulers such as weighted round robin (WRR) and weighted fair queueing (WFQ) are the dominant solutions for line card schedulers. There are several analytical results regarding the delay performance and memory requirements for data flows when a rate based scheduler is used. In line cards, scheduling is a many to one problem since there are multiple flows that are sharing a sin-

gle outgoing link. In a crossbar switch fabric, scheduling is inherently a many to many problem and line card solutions are not applicable anymore. Nevertheless, it is desirable to have a rate based scheduler with predictable performance.

In chapter 2, we start from fluid policies as the origin of rate based schedulers and tried to develop packetized tracking policies for fluid policies. Under a packetized tracking policy departure time for every cell is at most one cell time behind the fluid policy. For the special case of  $2 \times 2$  it is proved that tracking policies always exist and a packetized tracking policy is provided. Furthermore, a counter example for a  $3 \times 3$  switch fabric is given and it is concluded that in general tracking policies do not exist. However, a heuristic tracking policy is proposed that performs reasonably well. Under the heuristic scheduling algorithm all cells are scheduled with bounded delay with respect to the fluid policy. The heuristic policy measures how many cells are served for every link node and compare it with the number of cells that should have been served under the fluid policy. The difference is considered as the credits of a node and the heuristic algorithm always select the matching with the highest credit. The credits can also be considered as number of virtual backlogged cells.

The scheduler is capable of rate provisioning if accumulated credits remain finite. If we envision credits as the virtual backlogged cells, rate provisioning is possible if the scheduler is stable. This fact motivates us to focus on the heuristic algorithm and try to prove that it is stable. In chapter 3, we were able to go beyond that and introduce a class of scheduling algorithms that we call them the maximum node containing matching (MNCM) algorithms and show that they are rate stable. We use fluid model techniques to prove that these algorithms achieve 100% throughput with no speedup. Fluid model techniques are very powerful

tools for the analysis of the queueing systems. The only assumption on the arrival pattern is that it satisfies strong law of large numbers. Note that this property was essential for us since most of the classical proofs of stability are for Bernoulli i.i.d. arrivals. While the Bernoulli arrival model may seem reasonable for cell arrivals it is definitely not a valid assumption for the rate based systems when we are dealing with virtual cell (credit) arrivals. We also introduce a new weighted matching algorithm, the maximum first matching (MFM) with complexity  $O(N^{2.5})$  that belongs to MNCM. The MFM algorithm, to the best of our knowledge, is the lowest complexity deterministic scheduling algorithm that delivers 100% throughput.

We also introduced maximal sorted matching (MSM) algorithm that for all practical purposes perform similar to MFM. MSM complexity is  $O(N^2)$ , which is less than MFM. However, it does not belong to the MNCM class. One interesting path for future research is to prove that the MSM is also rate stable or to provide lower complexity algorithms that are rate stable.

For the rate provisioning applications, we introduced MSUIM. The MSUIM tries to find a matching that contains maximum number of nodes that their weight is at most one unit less than the maximum weight. We were able to find a deterministic bound (worst case performance bound) for the MSUIM.

In chapter 4, we focused on self-randomized scheduling algorithms. Due to their low complexity, the randomized algorithms have recently created a lot of interest in research community. Contrary to the deterministic scheduling algorithms, the randomized scheduling algorithms do not calculate a new matching every time, but they select a matching from a predetermined set of matchings. The selection criterion is usually weight of the matching and the matching with the highest weight is always selected. Note that calculating weight of a set of matchings and

selecting the one with highest weight is much simpler than constructing a matching with the maximum weight. The matching set is usually updated by insertion of a randomized generated matching to the set. In the self-randomized algorithms, source of the randomization process is usually the arrival pattern of the cells.

We used fluid model techniques to show that the self-randomized scheduling algorithms deliver 100% throughput. We provided a general architecture for the design of the self-randomized algorithms, and introduce two algorithms that consider number of the backlogged cells as the weight function. To do rate-provisioning, we introduced the concept of the max-min fair self-randomized scheduling algorithms. The idea here is to introduce self-randomized algorithms that can provide QoS by sharing the switch bandwidth proportional to the assigned weights. We introduced three self-randomized scheduling algorithms, and prove that they are max-min fair. In order to study and compare the performance of the proposed scheduling algorithms, several simulations were carried out and their results were provided and discussed.

We illustrated that previously proposed self-randomized scheduling algorithms do not have acceptable delay performance. In particular, we showed that for very low throughput values, cells can experience high delays. The delay performance gets better as the throughput increases to a certain level. After that as we expect, increasing the throughput results in the delay increase. Note that the self-randomized scheduling algorithms rely on the arrival pattern as the source of randomization, and when the arrival rate is very low they do not have sufficient sample points to generate good randomized matchings. We have addressed this problem in our proposed solutions and to some extent were able to resolve this problem. However, we believe that this field is far from maturity and it is still pos-

sible to come up with new randomized scheduling algorithms that perform well, have low complexity, and are capable of rate provisioning.

## BIBLIOGRAPHY

- [1] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High-speed switch scheduling for local area networks. *ACM Transactions on Computer Systems*, November 1993.
- [2] J.C.R. Bennett and H. Zhang. WF<sup>2</sup>Q: Worst-case Fair Weighted Fair Queueing. *IEEE INFOCOM '96*, 1996.
- [3] Jon C. R. Bennett and Hui Zhang. Hierarchical packet fair queueing algorithms. *IEEE/ACM Transactions on Networking*, 5(5):675–689, 1997.
- [4] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1992.
- [5] M.A. Bonuccelli and M.C. Clo. EDD algorithm performance guarantee for periodic hard-real-time scheduling in distributed systems. *IPPS/SPDP*, April 1999.
- [6] C.S. Chang, W.J. Chen, and H.Y. Huang. Birkhoff-von neumann input-buffered crossbar switches. *IEEE INFOCOM'00*, 2000.
- [7] N. Chrysos and M. Katevenis. Transient behavior of a buffered crossbar converging to weighted max-min fairness. *Inst. of Computer Science, FORTH, Heraklio, Crete, Greece*, 2002.

- [8] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queuing with a combined input output queued switch. *INFOCOM '99*, 1999.
- [9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. Mc Graw Hill, 1990.
- [10] R. L. Cruz. A Calculus for Network Delay, Part I, Network Elements in Isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991.
- [11] R. L. Cruz. A Calculus for Network Delay, Part II, Network Analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, 1991.
- [12] J.G. Dai. Stability of fluid and stochastic processing networks. *Miscellanea Publication, No. 9, Center for Mathematical Physics and Stochastics, Denmark (<http://www.maphysto.dk/>)*, January 1999.
- [13] J.G. Dai and B. Prabhakar. The throughput of data switches with and without speedup. *INFOCOM '00*, pages 556–564, 2000.
- [14] A. Demers, S. Keshav, and S. Shenkar. Analysis and simulation of a fair queueing algorithm. *Proceedings of SIGCOM '89*, pages 1–12, 1989.
- [15] J. Edmonds and R.M. Karp. Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM*, 19, 1972.
- [16] G. L. Frazier and Y. Tamir. The design and implementation of a multi-queue buffer for VLSI communication switches. *International Conference on Computer Design, Cambridge, Massachusetts*, pages 466–471, 1989.

- [17] Leonidas Georgiadis, Roch Guérin, and Abhay K. Parekh. Optimal multiplexing on single link: Delay and buffer requirements. *IEEE/Transactions on Information Theory*, 43(5):1518–1535, 1997.
- [18] Leonidas Georgiadis, Roch Guérin, Vinod Peris, and Kumar N. Sivarajan. Efficient network QoS provisioning based on per node traffic shaping. *IEEE/ACM Transactions on Networking*, 4(4):482–501, 1996.
- [19] P. Giaccone, B. Prabhakar, and D. Shah. Towards simple, high-performance schedulers for high-aggregate bandwidth switches. *INFOCOM '02*, 2002.
- [20] J. Giles and B. Hajek. Scheduling multirate periodic traffic in a packet switch shaping. *Conference on Information Science and Systems at John Hopkins University*, 1997.
- [21] S. Golestani. A self-clocked fair queueing scheme for broadband applications. *Proceedings of INFOCOM '94*, pages 636–646, April 1994.
- [22] E.L. Hahne. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on selected areas in communications*, 9(7), 1991.
- [23] J.E. Hopcroft and R.M. Karp. An  $n^{5/2}$  algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.
- [24] A. Hung, G. Kesidis, and N. Mckeown. Atm input-buffered switches with guaranteed rate property. *IEEE ISCC'98, Athens*, 1998.
- [25] T. Inukai. An efficient SS/TDMA time slot assignment algorithm. *IEEE Transactions on Communications*, 27, 1979.

- [26] M. Karol, M. Hluchyj, and S.P. Morgan. Input versus output queueing on a space division packet switch. *IEEE Transactions on Communications*, 35(12):1347–1356, 1987.
- [27] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
- [28] N. McKeown. iSLIP: A scheduling algorithm for input-queued switches. *IEEE Transaction on Networking*, 7(2):188–201, April 1999.
- [29] N. McKeown, V. Anantharam, and J. Warland. Achieving 100% Throughput in an Input-Queued Switch. *INFOCOM '96*, pages 296–302, 1996.
- [30] A. Mekkittikul and N. McKeown. *Scheduling VOQ switches under non-uniform traffic*. Technical Report CSL-TR-97-747, Stanford University, 1997.
- [31] A. Mekkittikul and N. McKeown. A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches. *INFOCOM '98*, pages 792–799, 1998.
- [32] A.K. Parekh and R.G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.
- [33] A.K. Parekh and R.G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case. *IEEE/ACM Transactions on Networking*, 2(2):137–150, 1994.
- [34] I.R. Philp. *Scheduling real-time messages in packet-switched networks*. PhD thesis, Department of Computer Science University of Illinois at Urbana-Champaign, 1997.

- [35] V. Tabatabaee, L. Georgiadis, and L. Tassiulas. QoS Provisioning and Tracking Fluid Policies in Input Queueing Switches. *IEEE Transaction on Networking*, 9(5), 2001.
- [36] L. Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. *INFOCOM'98*, 2:533–539, 1998.
- [37] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transaction on Automatic Control*, 37(12):1936–1948, Dec. 1992.
- [38] L. Tassiulas and S. Sarkar. Maxmin fair scheduling in wireless networks. *INFOCOM'02*, 2002.