

WEIGHTED ROUND ROBIN SCHEDULING IN  
INPUT-QUEUED PACKET SWITCHES SUBJECT TO  
DEADLINE CONSTRAINTS

A THESIS  
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND  
ELECTRONICS ENGINEERING  
AND THE INSTITUTE OF ENGINEERING AND SCIENCE  
OF BILKENT UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
MASTER OF SCIENCE

By

Idris A. Rai

July 2000

THESIS  
TS  
157.5  
-R35  
2000

WEIGHTED ROUND ROBIN SCHEDULING IN  
INPUT-QUEUED PACKET SWITCHES SUBJECT TO  
DEADLINE CONSTRAINTS

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND  
ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES  
OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF  
MASTER OF SCIENCE

By

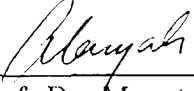
Idris A. Rai

July 2000

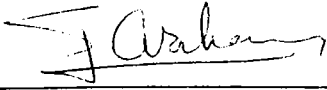
TS  
157.5  
· R35  
2000

B053005

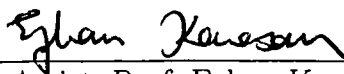
I certify that I have read this thesis and that in my opinion it is fully adequate,  
in scope and in quality, as a thesis for the degree of Master of Science.

  
Assist. Prof. Dr. Murat Alanyah(Supervisor)

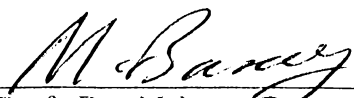
I certify that I have read this thesis and that in my opinion it is fully adequate,  
in scope and in quality, as a thesis for the degree of Master of Science.

  
Prof. Dr. Erdal Arıkan

I certify that I have read this thesis and that in my opinion it is fully adequate,  
in scope and in quality, as a thesis for the degree of Master of Science.

  
Assist. Prof. Ezhan Karařan

Approved for the Institute of Engineering and Sciences:

  
Prof. Dr. Mehmet Baray  
Director of Institute of Engineering and Sciences

## ABSTRACT

# WEIGHTED ROUND ROBIN SCHEDULING IN INPUT-QUEUED PACKET SWITCHES SUBJECT TO DEADLINE CONSTRAINTS

Idris A. Rai

M.S. in Electrical and Electronics Engineering

Supervisor: Assist. Prof. Dr. Murat Alanyalı

July 2000

In this thesis work, the problem of scheduling deadline constrained traffic is studied. The problem is explored in terms of Weighted Round Robin (WRR) service discipline in input queued packet switches. Application of the problem may arise in packet switching networks and Satellite-Switched Time Division Multiple Access (SS/TDMA) systems. A new formulation of the problem is presented. The main contribution of the thesis is a "backward extraction" technique to schedule packet forwarding through the switch fabric. A number of heuristic algorithms, each based on backward extraction, are proposed, and their performances are studied via simulation. Numerical results show that the algorithms perform significantly better than earlier proposed algorithms. The experimental results strongly assert Philp and Liu conjecture.

*Keywords:* input-queued packet switches, weighted round robin (WRR), scheduling algorithms, maximum matching, Quality of Service (QoS).

## ÖZET

### GİRİŞ-KUYRUKLU PAKET ANAHTARLARINDA SON-GÜN KISITLI TRAFİK İÇİN AĞIRLIKLI-DAİRESEL-SIRALI ZAMAN ÇİZELGELEMESİ

Idris A. Rai

Elektrik ve Elektronik Mühendisliği Bölümü Yüksek Lisans

Tez Yöneticisi: Yrd. Doç. Dr. Murat Alanyalı

Temmuz 2000

Bu tezde son-gün kısıtlı paket trafiğinin giriş-kuyruklu paket anahtarlarındaki zaman çizelgeleme problemi ele alınmaktadır. Problemin çözümü için Ağırlıklı-Dairesel-Sıralı servis disiplini öngörülmüştür. Ele alınan durum paket anahtarlama ağırlarında ve uydu anahtarlama zaman bölütlemeli sistemlerde ortaya çıkmaktadır. Tezde problemin yeni bir formülasyonu verilmiştir. Tezin ana özgün katkısı anahtardan paket geçişini çizelgelemek için kullanılan bir “geriye doğru çıkarma” tekniğidir. Herbiri bu tekniğe dayanan buluşsal çizelgeleme algoritmaları önerilmiş ve bu algoritmaların başarımları benzetimlerle örneklenerek çalışılmıştır. Elde edilen sayısal sonuçlar algoritmaların başarımlarının daha önceden kullanılan algoritmalara göre çok daha iyi olduğunu göstermektedir. Deneysel sonuçlar ayrıca Philp ve Liu tarafından ileri sürülen bir sanıtı desteklemektedir.

*Anahtar Kelimeler:* Giriş-kuyruklu paket anahtarları, ağırlıklı-dairesel-sıra, zaman çizelgeleme algoritmaları, en fazla eşleme, servis kalitesi.

## ACKNOWLEDGMENTS

I would like to express my deep gratitude to my supervisor Assist. Prof. Dr. Murat Alanyalı for his guidance, suggestions and invaluable encouragement throughout the development of this thesis.

I would like to thank Prof. Dr. Erdal Arıkan and Assist. Prof. Ezhan Karaşan for reading and commenting this thesis.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>INTRODUCTION</b>   | <b>1</b>  |
| <b>2</b> | <b>BACKGROUND INFORMATION</b>                                   | <b>4</b>  |
| 2.1      | Switching Background . . . . .                                  | 4         |
| 2.2      | Survey of Previous Work . . . . .                               | 9         |
| 2.2.1    | Scheduling Algorithms for Pure FIFO switches . . . . .          | 9         |
| 2.2.2    | Scheduling Algorithms for non-FIFO switches . . . . .           | 10        |
| 2.2.3    | Scheduling Traffic with Deadline Constraints . . . . .          | 13        |
| 2.2.4    | Fluid Tracking Policies in Packet Switches . . . . .            | 15        |
| <b>3</b> | <b>PROBLEM FORMULATION</b>                                      | <b>17</b> |
| 3.1      | Weighted Round Robin Scheduling                                 | 18        |
| 3.2      | Weighted Round Robin Scheduling in Input Queued Packet Switches | 19        |
| 3.3      | Analytical Problem Formulation . . . . .                        | 21        |
| 3.4      | Backward Extraction and Philp and Liu Conjecture . . . . .      | 25        |



|          |   |           |
|----------|---|-----------|
| 3.5      | Delay Bounds for Periodic Traffic . . . . . | 29        |
| 3.6      | Feasible Schedule for 2×2 Switch            | 30        |
| 3.7      | Per-Port vs Per-VC Scheduling               | 31        |
| <b>4</b> | <b>HEURISTIC ALGORITHMS</b>                 | <b>33</b> |
| 4.1      | Basic Algorithm . . . . .                   | 33        |
| 4.2      | Variations of Basic Algorithm . . . . .     | 37        |
| 4.2.1    | Oldest Deadline First Approach . . . . .    | 37        |
| 4.2.2    | Balancing Service Ratios Approach . . . . . | 41        |
| 4.2.3    | Deadline Relaxations . . . . .              | 43        |
| <b>5</b> | <b>NUMERICAL RESULT</b>                     | <b>44</b> |
| 5.1      | Traffic Matrix Generation                   | 44        |
| 5.2      | Discussion of Numerical Results . . . . .   | 45        |
| <b>6</b> | <b>SUMMARY</b>                              | <b>56</b> |

# List of Figures

|     |  |    |
|-----|--|----|
| 2.1 | A Generic Switch. . . . .  | 5  |
| 2.2 | Output Queued Switch.  | 6  |
| 2.3 | Input Queued Switch.   | 7  |
| 2.4 | Virtual Output Queuing. . . . .  | 8  |
| 2.5 | Combined Input Output Queued switch. . . . .   | 8  |
| 3.1 | Input Queued Weighted Round Robin (IQ-WRR). . . . .  | 20 |
| 3.2 | Per-VC vs Per-Port Scheduling. . . . .   | 31 |
| 4.1 | The Basic Algorithm. . . . .   | 34 |
| 4.2 | Oldest Deadline First Approach. . . . .  | 38 |
| 4.3 | Balancing Service Ratios Approach. . . . .   | 42 |
| 4.4 | Deadline Relaxation by 1 time slot. . . . .  | 43 |
| 5.1 | Success rates of Basic Algorithm, Oldest Deadline First Approach<br>and Balancing Service Ratios Approach without deadline relax-<br>ation for $N = 4$ and 10000 trials per schedule length. | 47 |

|     |  |    |
|-----|--|----|
| 5.2 | Success rates of Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach without deadline relaxation for $N = 8$ and 10000 trials per schedule length.   | 48 |
| 5.3 | Success rates of Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach without deadline relaxation for $N = 16$ and 5000 trials per schedule length.   | 49 |
| 5.4 | Percentage of packets missing deadline by 1 time slot for Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach with one time slot deadline relaxation for $N = 4$ and 10000 trials per schedule length. | 50 |
| 5.5 | Percentage of packets which miss their deadlines for Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach with one time slot deadline relaxation for $N = 8$ and 10000 trials per schedule length.      | 51 |
| 5.6 | Percentage of packets which miss their deadlines for Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach with one time slot deadline relaxation for $N = 16$ and 5000 trials per schedule length.      | 52 |

To My Parents ...

# Chapter 1

## INTRODUCTION

Rapid advances in optical communications, which has made the available transmission bandwidth to the tune of many gigabits per second seem to solve bandwidth problem. Advancements in optical technology however, also resulted in the emergence of new applications such as real-time services. Broadband Integrated Services Digital Networks (B-ISDN) coupled with packet switching is one of the techniques proposed to make network supports all existing and emerging services in a unified fashion.

The advancement in transmission speed leaves switching as the barrier to building high-performance networks. Switching is a task that basically involves two separate tasks: 1) *scheduling*; choosing the eligible packet to be sent to the output port if there are more than one in the same input port, and 2) *data forwarding*; delivering the packet to its addressed output port.

Output queued switches have been widely used for packets scheduling. When a packet arrives at an output-queued switch, it is immediately placed in a queue that is dedicated to its outgoing line, where it waits until it departs from the switch. This approach is known to provide 100% throughput [1]. Many queue

management policies are shown to provide various Quality of Service (QoS) features such as delay, bandwidth and fairness guarantees [2–6]. For pure output queuing scheme to work, the speed of switching fabric and output buffer memory is required to be  $N$  times the input line speed (where  $N$  is the switch size). As the line speed increases to gigabits per second range and as the switch size increases due to exponentially increasing demand, the required speed becomes infeasible.

To overcome these problems, switches which employ input queuing are being extensively considered [7, 8, 8–15]. In this scheme an incoming packet is first stored in queues at the input side and a slower fabric would transfer some of packets to the output line immediately. A scheduling algorithm decides when packets are transferred across the fabric. Input queued switches however, limit the switch throughput to at most 56.8% due to head of line (HOL) blocking [1]. Two techniques proposed to improve the throughput of input queued switch are Virtual Output Queuing (VOQ) and increasing the speedup of the switch fabric. In VOQ, an input queue maintains a separate queue for all packets to be departed to the same output queue while increasing speedup of switch fabric enables the switch fabric to transfer more than one packet from the same input queue to output ports in one time unit.

In this thesis work, the problem of scheduling deadline constrained traffic in input-buffered packet switches using weighted round robin (WRR) servers was studied. In the case of deadline constrained traffic, each packet is characterized by its network life time before which it should be delivered to its destination. This problem was earlier considered in single link (output queued switch) case where in [16], an algorithm that always satisfies all deadlines is proposed. Multiple link (input queued) case was studied in [17–19] where an efficient algorithm was defined to be the one that minimized the number switching matrices (sets of conflict free connections) and switching duration. The problem which considers deadline guarantees was first presented in [20, 21]. A schedule which satisfies all

deadlines is called *feasible schedule*. In [21], Philp and Liu conjectured that there exists a feasible schedule to any periodic traffic if link utilization is no more than unity.

In [22], Giles and Hajek showed that a feasible schedule always exists if each period is evenly divides all longer periods and if utilization at each link is no larger than unity. Giles and Hajek proposed an algorithm which schedules an arbitrary multi-periodic traffic if utilization at each link is less than  $\frac{1}{4}$ . In [23], scheduling deadline constrained traffic was shown to be a tracking problem in which scheduling algorithm tracks the deadlines of all packets so that they are serviced before their deadlines. In [23], it is shown that a tracking policy always exists for a  $2 \times 2$  switch. Heuristic algorithm for a general switch size is proposed.

In this work, a new formulation of obtaining a set of connections to receive service at each time slot is presented. This formulation is based on a technique known as backward extraction. The formulation provides much insight to the understanding the problem. In particular, it leads to analytical general interpretation of Philp and Liu conjecture. A number of heuristic algorithms are proposed and simulation results which show their success rates in finding a feasible schedule are presented.

By guaranteeing deadlines, a switch actually satisfies various QoS measures such as worst delay bound, and provides bandwidth and fairness guarantees. In addition, by taking the advantages of simplicity of WRR servers hardware implementation of algorithms is possible.

Thesis organization is as follows; in Chapter 2 switching background and survey of previous works are presented. Next, scheduling problem is formulated in Chapter 3. Heuristic algorithms are proposed in Chapter 4. In Chapter 5, numerical results are presented and discussed and finally the thesis is summarized in Chapter 6.

## Chapter 2

# BACKGROUND INFORMATION

In this chapter, background information related to this thesis work is discussed. Packet switching background is discussed in Section 2.1 and in Section 2.2 a survey of scheduling algorithms for input queued packet switches is presented. The chapter is finished by a discussion of fluid tracking policies.

### 2.1 Switching Background

A generic packet switch consists of input ports, a switch fabric and output ports. The number of input ports and output ports determine the switch size. In general, the number of input ports is equal to the number of output ports and they are connected to links of equal bandwidth. Figure 2.1 elaborates the architecture of a generic switch. In this thesis a switch which transmits information in fixed size entities called packets (cells) is assumed. Switch fabric is used to transmit packets from input ports to output ports in the switch. Crossbar, for example, is a very popular fabric used in building networks with input queued switches



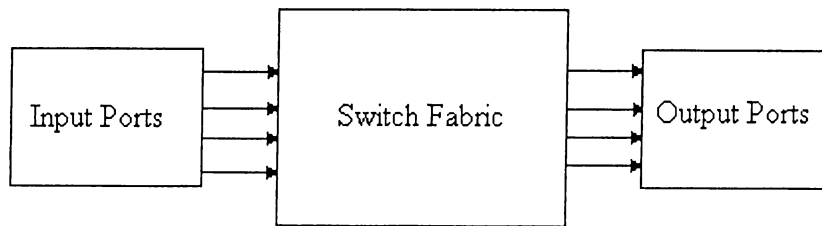


Figure 2.1: A Generic Switch.

because of its low cost, good scalability and non-blocking property [9]. A Switch fabric is non-blocking if it can transmit packets from different inputs to different outputs simultaneously in a single time slot; otherwise, the fabric is termed as blocking. Time slot is the time between packet arrivals at input ports.

A Switch fabric is characterized by its *speedup* which is defined as the ratio of switch fabric speed to the line speed. Speedup, sometimes called *switching capacity*, determines the number of packets that a switch can transmit to output ports from the same input port in one time slot. If the speedup is equal to the switch size,  $N$ , buffers/queues are required at the output ports and the switch is called an output-queued switch. If the speedup is 1, buffers are required at the input ports and the switch is called an input-queued switch. A switch is called a combined input/output queued (CIOQ) switch if its switch fabric has a speedup between 1 and  $N$ . The terms buffer and queue are used interchangeably meaning memory device used to store packets before or after they are scheduled. In the next few subsections, the basic mechanisms of these buffered switches are discussed.

### Output Queued Switches

In output queued switches, all buffers are maintained at the output ports as can be seen in Figure 2.2. When a packet arrives at input port of output queued (OQ) switch, it is immediately placed in a queue that is dedicated to outgoing line, where it waits until departing from the switch. This approach is known to

maximize the switch throughput, it completely eliminates output blocking [1]. So long as no input or output is oversubscribed, i.e. link utilization of each link is less than unity, the switch is able to support the traffic and the occupancies of queues remain bounded. Furthermore, by controlling departure times of packets belonging to different sessions, a switch can control latency of packets and hence provides quality of service (QoS) guarantees such as bandwidth guarantee, delay guarantees, fairness etc. Sophisticated but practical scheduling algorithms which provide QoS in output queued switches are proposed [2–6, 24, 25], etc. Output

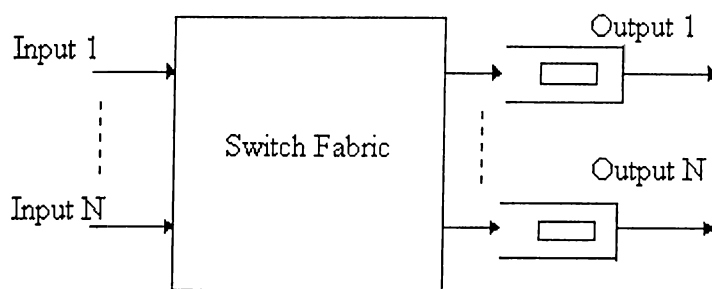


Figure 2.2: Output Queued Switch.

queuing is impractical for switches with high line rates and/or with large number of ports. A switch fabric and output memory of switch of size  $N$  must run  $N$  times as fast as the line rate. At high line rates, the switch memory and fabric running at high speed are not available or very expensive. This major limitation to output queued switches has diverted the attention in switching research to input queued switches.

### Input Queued Switches

In input queued switches, buffers are maintained at each input port of the switch as seen in Figure 2.3. When a packet arrives at an input port, it is queued in the buffer and waits for its time to be scheduled for departure to output port. When First In First Out (FIFO) technique is used, a packet reaches the head of line of the queue if all cells which arrived before it are scheduled. In contrast to output

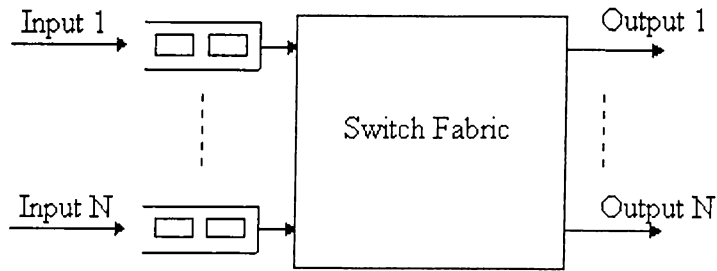


Figure 2.3: Input Queued Switch.

queued switch, the fabric and the memory of the input queued switch needs only to run as fast as the line rate. This makes input queuing very appealing for switches with fast line rates or with large number of ports, hence, input queued switches are increasingly being pursued by the research community.

The longstanding view has been that input-queued switches are impractical because of their poor performance. If FIFO queues are used to queue packets at each input, only the first cell in each queue is eligible to be forwarded. As a result, FIFO input queues suffer from head of line (HOL) blocking. If a packet at the front of the queue is blocked due to another packet contending the same output port, it also blocks all the packets behind it in the queue even though those packets may be destined for an output that is currently idle. HOL blocking limits switch throughput to approximately 58.6% when FIFO service technique is used under uniform traffic<sup>1</sup> [1]. To make the input queued switches practical, there are proposed strategies which completely eliminate HOL blocking [26,27].

One of the techniques used to improve the throughput of input queued switches is *Virtual Output Queuing* (VOQ) [27]. In VOQ, each input maintains a separate queue for each output. With a suitable centralized scheduling algorithm, HOL blocking is completely eliminated and the throughput is increased to 100% [12,13]. Figure 2.4 elaborates VOQ architecture. The other proposed method to improve the input-queued switch performance is increasing speedup

---

<sup>1</sup>Traffic is uniform if all arrival processes have the same arrival rate, and if the destination of packets are uniformly distributed over all outputs

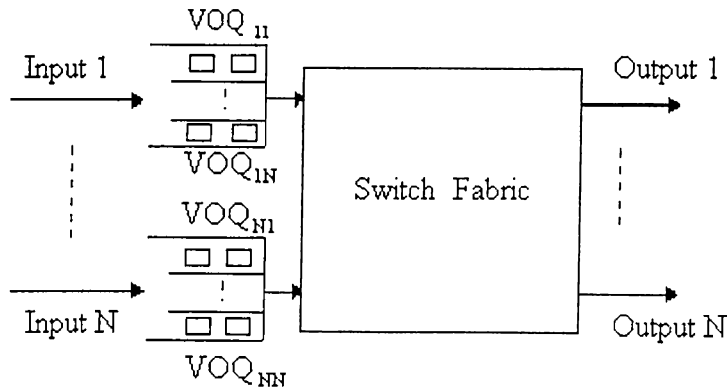


Figure 2.4: Virtual Output Queuing.

of the switch fabric. This results in a combined input/output queued switch, Figure 2.5. This switch has buffers at input and output ports, this is because only one packet departs from an output port in each time slot and all packets arriving at an input port may not be transferred to their addressed output ports immediately after their arrivals. In this technique throughput is improved by scheduling a number of packets from the same input queue at the same time slot. A CIOQ switch is said to emulate output queued switch, if each packet

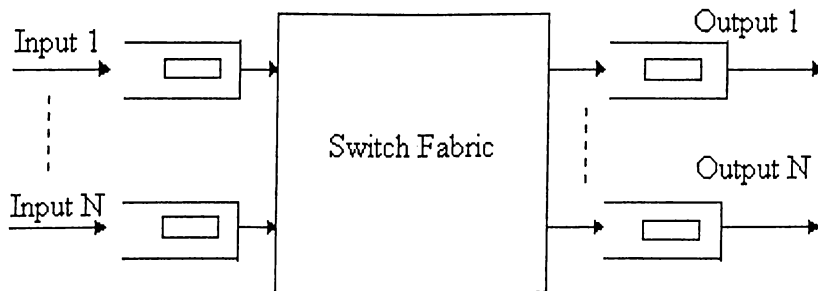


Figure 2.5: Combined Input Output Queued switch.

departs from CIOQ switch at exactly the same time as it would depart from an output queued switch, had the traffic been applied to the output queued switch. In [9, 28] it is shown that for a CIOQ switch with appropriate scheduling algorithm, speedup of 2 is sufficient to guarantee the emulation of output queued switch. Input-queued switches with improved throughput need very fast scheduling algorithms so that the switch can operate in very high-speed networks. A

brief survey for scheduling algorithms for input queued switches is presented in Section 2.2.

## 2.2 Survey of Previous Work

Scheduling algorithms are sometimes called selection policies, contention resolution mechanisms or head of line (HOL) arbitration mechanisms. The main task of a scheduling algorithm is to select a set of contention free packets (cells) from a set of input queues to be transmitted through the switch fabric. Contention free set of packets is a set which has all packets from different inputs to be departed to different outputs. Desirable properties of a scheduling algorithm are *efficiency*; it should provide high throughput, *fairness*; it should not leave other input queues starving, *implementational simplicity*; the algorithm should be simple to implement in hardware, *high speed*; it should be fast, *stability*; the expected occupancy of every input queue should remain finite for every admissible traffic pattern<sup>2</sup> Scheduling algorithms for input queued switches can be classified as those using pure FIFO queuing strategies and those which do not.

### 2.2.1 Scheduling Algorithms for Pure FIFO switches

Maintaining single FIFO queue for each input is the simplest approach to managing queues and scheduling cells. An arriving cell is stored in an input queued switch if the cell arrived before which is not scheduled. Scheduling is done by examining the cells at the head of each FIFO queue. A problem arises when more than one packet of different input ports contend for the same output port. In this case, the scheduler has to select only one among the contending cells to transmit. The problem of FIFO scheduling algorithms then is to determine

---

<sup>2</sup>A traffic pattern is said to be admissible when no input or output is oversubscribed.

which of the contending head of line packets are to be selected first. A number of approaches are proposed in the literature, such as, maintaining fixed priority, randomly selecting a cell among the contending inputs, rotating priorities, etc. A classification of the proposed selection policies are as follows:

- Cyclic (Round Robin) selection and variants [29],
- Global FIFO selection [29],
- HOL FIFO selection [29–31],
- LIFO (Last-in First-out selection) [31],
- Longest Queue selection [29],
- Random selection [30, 31].

The throughput of all above strategies is generally small, but other performance measures such as queuing delay, cell loss probability differ from one algorithm to another. In [29–31], the throughput is shown to be limited to 58.6% . In some cases however, it is possible to increase the throughput to 63.2% [29,30], for example in longest queue selection and random queue selection. Next, more complicated contention resolution algorithms aimed at removing the 58.6% barrier of maximum achievable throughput are presented.

### **2.2.2 Scheduling Algorithms for non-FIFO switches**

Most scheduling algorithms for non-FIFO switches make use of one or both techniques for improving input-queue performance discussed in Section 2.1 and matching algorithms. In this section, a brief survey of previous works of scheduling algorithms for non-FIFO switches is discussed.

## Iterative Matching Algorithms [7, 8, 11]

These algorithms attempt to achieve maximal matching by iteratively adding connections to fill the missing connections in the previous iteration in matching.

Examples of these algorithms are;

1. Parallel Iterative Matching (PIM) [8, 11],
2. Iterative Round Robin Matching (IRRM) [11],
3. Iterative Round Robin Matching with SLIP(SLIP-IRRM) [7],
4. Least Recently Used (LRU) [8, 11].

In these algorithms, it is important to decide how many iterations to be performed in one time slot (i.e. the speedup). These algorithms are very similar, except for the ordering of the entries in the schedule. In [8], simulation was conducted and the performance of PIM, SLIP and LRU was compared for single iteration in a time slot (speedup of 1). It is shown that, PIM, SLIP and RLU achieve maximum offered load of 63%, 100% and 64% respectively. Simulation results in [8] show that, SLIP can deliver throughput asymptotic to 100% of each link and it is simple to implement in hardware, but it has much higher output delay. It is also shown that, when four iterations are used, the above algorithms are indistinguishable. IRRM is a simplified version of PIM [11], it is said to be simpler to implement and its maximum achievable throughput is asymptotic to 100% in two iterations (speedup of 2). SLIP is identical to IRRM with a few modifications [11].

## Maximum Matching Algorithms for Input Queued Switches

Maximum matching algorithms find a match with maximum possible size. Maximum matching achieves the highest possible throughput in each slot for an

input-queued switch but can result in starvation of an input-output connection under certain traffic pattern [8]. The most efficient algorithm solves maximum matching problem in  $O(N^{2.5})$  time. Examples of scheduling algorithms which use maximum matching algorithm is neural networks algorithms [27] and Longest Port First(LPF) [8].

### Weighted Matching Algorithms for Input Queued Switches

In maximum weight matching algorithm edges are characterized by their weights. The algorithm then finds a match which maximizes total weight. As with maximum size matching, bipartite maximum weight matching can be found by solving an equivalent network flow problem. The most efficient known algorithm for solving this problem converges in  $O(N^3 \log N)$  [12]. In the context of cell scheduling, weights are considered to be occupancies of queues and ports, and waiting times of packets. Examples of weighted matching used in packet scheduling in input-queued switches are;

1. Matrix Units Cell Scheduler (MUCS) [32],
2. Longest Queue First (LQF) [13, 14],
3. Longest Port First (LPF) [12–14],
4. Oldest Cell First (OCF) [9],
5. Largest Output Occupancy First Algorithm(LOOFA) [9],
6. Gale-Shapley Oldest Cell First (GS-OCF), [26],
7. Gale-Shapley Longest Queue First (GS-LQF) [26].

MUCS resolves output contention by computing a traffic matrix which summarizes the status of queued cells in the inputs. It gives priorities to head of



queue cell which has least contending candidates from the same input to the same output port. This weighting technique is claimed to offer maximum opportunities to the remaining head of line cells in each iteration of MUCS [32]. The algorithm offers near-optimal throughput, approximately 100% [32]. Other examples, as their names imply, are obtained by using different weights for the underlined weighted matching algorithm. Some of the above algorithms are used in conjunction with each other, for example LOOFA-OCF is used in to emulate the performance of output-queued switch by input-queued switch with speedup of 2 [28].

In [12–14], LQF and LPF are used to make input queued switch achieve throughput of 100%. The difference between these algorithms is that, LPF combines the benefits of maximum matching algorithm with those of maximum weight algorithm while lending itself to simple implementation in hardware. LPF effectively finds a set of maximum matches and chooses the one match from the set with largest total weight. LPF achieves 100% throughput, fast with complexity improved to  $O(N^{2.5})$  and is hardware implementable [14]. Similarly, in [12], OCF algorithm was proposed to overcome the permanent starvation problem that can result in LQF algorithm. It is also shown to achieve 100% switch throughput. GS-LQF and GS-LPF are examples of stable matching problem [26] which is used for scheduling packets in input queued switches [28, 33]. The interest in using stable matching techniques in packet switch has increased recently. This is due to its linear complexity of  $O(N^2)$ , (i.e., linear in the number of edges) as opposed to other weighted matching techniques which have higher complexity.

### 2.2.3 Scheduling Traffic with Deadline Constraints

Each packet of deadline constrained traffic is characterized by life time (deadline) in the network. If a packet is not delivered to its destination by its deadline it needs extra care from a network such as buffering or discarding. In IP networks,

for example, each packet has its deadline, and a message is sent to the source when the packet is dropped for failing to reach its destination by that deadline [34]. Other examples of deadline constrained traffic are constant rate traffic and multi-periodic traffic. An example of deadline constrained traffic mostly considered is periodic traffic.

Scheduling periodic messages has extensively been studied in the context of Satellite Switched Time Division Multiple Access SS/TDMA. Originally, the problem was solved for output queued case [16]. It is known that Earliest Deadline First (EDF) and Maximum Laxity First (MLF) algorithms provide optimum results whenever the link utilization is less than or equal to 1.0 [16]. For input queued case, scheduling multi-periodic traffic is generally known as Time Slot Assignment (TSA) problem. In [17–19] for example, the optimal algorithm was defined to minimize the number of switching modes and transmission time. In [17], Inukai identified an optimal scheduling algorithm for  $N \times N$  switch when periods of all messages are equal and utilization of each input and output links is less than or equal to 1.0.

Previous works for input queued case, did not consider the concept of deadline of packets. This consideration was first introduced by Philp and Liu in [21] and further explored in [20, 22]. In these works, the problem was to find a conflict free schedule which satisfies all deadlines of packets. This schedule is called a *conflict free feasible schedule*. A number of heuristic time slot assignment (TSA) algorithms are proposed in [21]. These algorithms are based on EDF and MLF, and are named as Earliest Deadline First Row by Row (EDF-RR) and Maximum Laxity First Row by Row (MLF-RR) respectively. Other algorithms are based on System of Distinct Representatives (SDR); EDF-SDR and MLF-SDR. These algorithms degrades as switch size increases.

In [21], simulations were performed to determine how often their algorithms provide a feasible schedule for traffic with link utilization less than or equal to

1.0. The simulation results led Philp and Liu to conjecture the existence of a feasible schedule as introduced in Chapter 1. In [22], Multiple Period Time Slot Assignment (MP-TSA) problem is explored and a number of heuristic algorithms are proposed. Nested Period Scheduling (NPS) algorithm finds a feasible schedule for evenly divided periods if switch utilization is less than or equal to 1.0. NPS is shown to schedule traffic with arbitrary periods if the utilization is not larger than  $1/4$ . Other proposed algorithms in [22] are Slot-by-Slot, Earliest Deadline First, Earliest Arrival First (SS-EDF-EAF). If utilization is less than or equal to  $1/14$ , SS-EDF-EAF is shown to provide a feasible schedule.

In this work, a similar problem of scheduling deadline constrained traffic in input queued packet switches as considered by these previous works was studied. This work differs from the previous works in its more general formulation the which contributes in giving more insight to the problem. The generality of formulation as will be presented in the next chapter is the fact that previous works assume that all packets have equal relative deadlines which are equal to the period of their connection.

## 2.2.4 Fluid Tracking Policies in Packet Switches

Fluid Policy is an idealized scheduling policy governed by a server which does not transmit information in the form of packets. It assumes that the server can serve all backlogged sessions simultaneously and the traffic is infinitely divisible. This policy is defined for analysis purposes only, it can not be implemented in real network. Generalized Processor Sharing (GPS) or Fluid-Flow Fair Queuing (FFQ) is a fluid policy proposed for output queued switch in [2, 3].

A realistic packet system that tracks departure process of fluid policy is called a *tracking policy*. In a tracking policy, only one session can receive service at a time and the entire packet must be served before another packet can be served.

A number of tracking policies which approximately mimic fluid policy are proposed in literature for output queued case. For example, Packetized Fair Queuing (PFQ), Weighted Fair Queuing (WFQ), Self-Clocked Fair Queuing (SCFQ) schemes are considered [2, 3, 5, 6]. Hardware implementations of SCFQ and Distributed Packet Fair Queuing (D-PFQ) are presented in [4] and [24] respectively.

Designing an optimal tracking policy for input-queued switches is still an open problem. The first reference to deal with input queued switch tracking policies noted is [23]. In this thesis work, a fluid model under constant-rate traffic is considered instead of nonanticipative<sup>3</sup> traffic as defined in [23]. It is easy to note that if a scheduling algorithm provides a feasible schedule, then it is a tracking policy of the defined fluid policy. In [23], a tracking policy is shown to exist for a  $2 \times 2$  input-queued switch and it is claimed that it also exists for a general switch size.

---

<sup>3</sup>It is a traffic in which future link rate depends on future arrivals.

## Chapter 3

# PROBLEM FORMULATION

In this section, the problem of scheduling deadline constrained traffic is formulated. A traffic stream obtained by superposition of periodic streams, each with possibly different period is called *multi-periodic traffic*. In this work, multi-periodic traffic and an  $N \times N$  switch with link utilizations of exactly 1.0 are considered. Periodic traffic is defined to be the traffic which transmits the same number of packets in one period and each packet arrives at the deadline of the previous transmitted packet by the same connection. Packets of a connection may have different deadlines less than the period of the connection. A packet has its deadline equal to the period of a connection if and only if it is the only packet to be transmitted by that connection in one period. The objective is to devise a scheduling algorithm which guarantees that each packet is serviced from its input queue before its deadline. Real-time messages fit the periodic traffic model because they are generated by periodic processes [21]. This formulation is also suitable to SS/TDMA systems because it is not usually possible to output-queue packets in satellite switches [23]. This is due to the fact that the line speed in satellite switches is very high.

By employing WRR service technique, the problem becomes guaranteeing a specific number of services to each connection in specified duration under the constraint that all packets are serviced before or at their deadlines. The problem is analytically formulated by introducing WRR scheduling technique in the next section.

### 3.1 Weighted Round Robin Scheduling

Round robin is a service discipline which services sessions sharing the same link in a cyclic manner. It is the simplest technique to approximate GPS; an idealized fluid model for output queued packet switches. Weighted round robin (WRR) is a round robin service discipline which services sessions according to their defined weights [20, 35, 36]. In every round of service, the number of packets serviced from a queue is proportional to its associated weight. WRR service discipline has low complexity, schedules at constant rate and guarantees various QoS such as minimum bandwidth and fairness. WRR is well studied in output queued case where for a queue shared by  $N$  sessions each with weight  $m_{ji}$ , WRR of queue  $j$  is represented as:

$$WRR_j = (m_{j1}, m_{j2}, \dots, m_{jN}). \quad (3.1)$$

WRR schedule of queue  $j$  is the schedule which guarantees that every connection  $i$  in the queue  $j$  receives  $m_{ji}$  services in every  $\sum_{i=1}^N m_{ji}$  time slots. In the next section, an input queuing mechanism that uses WRR servers is presented.

## 3.2 Weighted Round Robin Scheduling in Input Queued Packet Switches

In this work, an input buffered switch with Virtual Output Queuing (VOQ) strategy discussed in Section 2.1 is considered. WRR servers at each input queue, select a set of conflict free head of line cells of virtual output queues to be transferred to output ports of the switch. Input-queued WRR (IQ-WRR) service technique is similar to output queued WRR, each input queue has its WRR server to service sessions sharing the same queue. In contrast to output queued WRR servers however, input queued WRR servers need to provide services to each session in a defined regular mechanism. In this discipline, each server should select a packet from input queue to be transferred to a different output port. Moreover, WRR servers should satisfy deadlines of all packets which is guaranteed by a scheduling algorithm. In any time slot, exactly  $N$  packets are transmitted from input queues to output ports such that their deadlines are all satisfied.

VOQ switch architecture has  $N^2$  input queues whose status can be efficiently represented by an  $N \times N$  matrix. In this thesis, the status of this matrix represents the number of packets each VOQ needs to transfer to output port in one *schedule length* and the matrix is called *traffic matrix* defined as follows;

**Definition 1.** *Traffic matrix,  $M = [m_{ij}]_{N \times N}$ , is a non-negative integer matrix such that for some integer  $P$ ,*

$$\sum_{i=1}^N m_{ij} = P \quad \forall j = 1, 2, \dots, N,$$

$$\sum_{j=1}^N m_{ij} = P \quad \forall i = 1, 2, \dots, N.$$

where  $P$  is called the *schedule length*.

In general, each row(column) of a traffic matrix represents an independent output-queued WRR scheduling discipline by definition in Equation 3.1. WRR

of input queues  $i$  is then represented as;

$$WRR_i = (m_{i1}, m_{i2}, \dots, m_{iN}), \quad i = 1, 2, \dots, N. \quad (3.2)$$

It is easy to see that Equation 3.1 is the same as Equation 3.2 which means that output-queued WRR is the same as input-queued WRR. The only difference between output-queued WRR and input-queued service disciplines is that the service order in the former can be arbitrary while it is regulated among all servers by scheduling algorithm in the latter discipline. In this work, weights of input-queued WRR service discipline,  $m_{ij}$ , are considered to be number of packets to be transferred to output port  $j$  from input queue  $i$  in one schedule length, i.e.  $P$  time slots. Figure 3.1 illustrates the input-queued WRR. The mechanism is

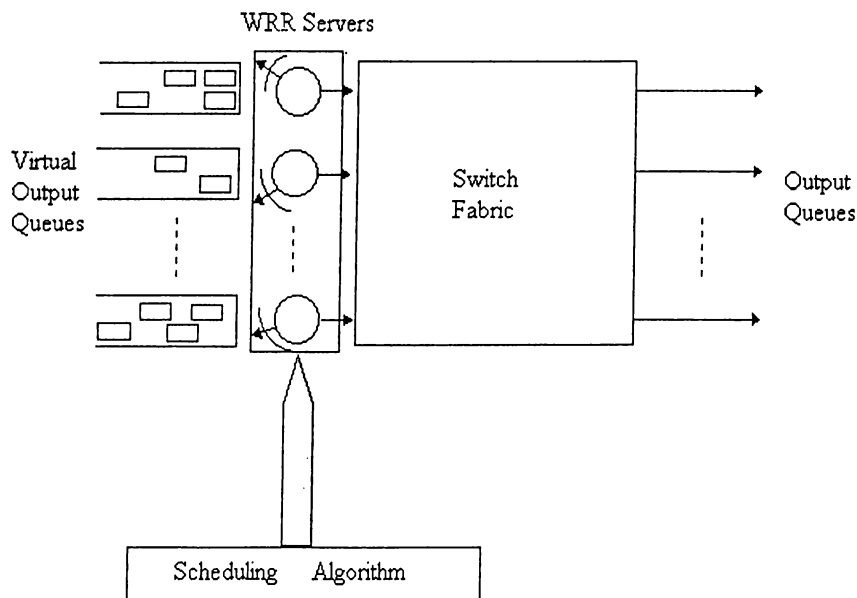


Figure 3.1: Input Queued Weighted Round Robin (IQ-WRR).

made up of  $N$  input queues, one server for each input queue, a non-blocking crossbar, and  $N$  output ports. Virtual output queuing is assumed to each input queue. In this mechanism, work conserving WRR servers are assumed, i.e. they are never idle if there are packets to service. The synchronization of servers and the choice of sessions to be transmitted at each time slot is commanded by scheduling algorithms as seen in Figure 3.1. Analytical formulation of the problem is presented in the next section.



### 3.3 Analytical Problem Formulation

In this section, definitions and some combinatorial background required in problem formulation are presented. Scheduling algorithms take as input a traffic matrix defined in the previous section. The ratio  $\frac{m_{ij}}{P}$  represents the rate of service required between input  $i$  and output  $j$ , and it is denoted by  $\rho_{ij}$ . First, WRR schedule is defined.

**Definition 2.** *WRR schedule is a schedule such that the  $k^{\text{th}}$  service to connection  $(i,j)$  is provided by the time-slot  $\lceil k \frac{P}{m_{ij}} \rceil$  for any  $k = 1, 2, 3, \dots, m_{ij}$ , and for each connection  $(i,j)$ , where  $\lceil x \rceil$  is defined as the minimum integer greater than or equal to  $x$ .*

WRR schedule is a feasible schedule because it satisfies all deadlines. By definition, deadline of the  $k^{\text{th}}$  service is  $\lceil k \frac{P}{m_{ij}} \rceil$ . Next, the notion of permutation matrix is presented.

**Definition 3.** *Permutation matrix is a binary matrix with exactly one nonzero element in each row and each column.*

This matrix shall repeatedly be used in the next sections such that nonzero entries represent  $N$  conflict free set of connections to be simultaneously serviced in any time slot. A proposition based on permutation matrices which is of use in this work is presented. The proof of this proposition can be found in several books in combinatorial analysis. Particularly see Theorem 2.2.6 in [37].

**Proposition 1.** *Any non-negative integer matrix  $M$  with row and column sums equal to  $P > 0$  can be represented as a sum of  $P$  permutations matrices  $\bar{\Pi}_1, \bar{\Pi}_2, \dots, \bar{\Pi}_P$ :*

$$M = \bar{\Pi}_1 + \bar{\Pi}_2 + \dots + \bar{\Pi}_P.$$

|             |   |   |   |   |   |
|-------------|---|---|---|---|---|
| $l$         | 1 | 2 | 3 | 4 | 5 |
| $d_l(i, j)$ | 0 | 1 | 0 | 1 | 1 |

Table 3.1: Computation of deadline sequence vector for  $\rho_{i,j} = 3/5$ .

Note that the above proposition implies that a traffic matrix can be represented as a sum of  $P$  permutation matrices. Next, a binary deadline sequence is defined;

**Definition 4.** *Deadline sequence for connection  $(i, j)$  is a one-sided binary sequence  $d_1(i, j), d_2(i, j), \dots$  such that for any  $k \geq 1$ ,  $\sum_{l=1}^k d_l(i, j)$  is the minimum number of services that should be received by the connection  $(i, j)$  by time slot  $k$ .*

Note that, the entries  $d_l$  are obtained as follows;

$$d_l(i, j) = \begin{cases} 1, & \text{if } l = \lceil k \frac{P}{m_{ij}} \rceil, k \geq 1 \\ 0, & \text{else.} \end{cases}$$

Table 3.1 illustrates computation of deadline sequence vector  $d(i, j) = (d_1(i, j), d_2(i, j), \dots, d_P(i, j))$  for a connection with service rate  $\rho_{ij} = 3/5$ .

1s in deadline sequence represent deadlines of services in a connection. In addition, since  $d_l(i, j) = d_{l+P}(i, j)$ , the vector  $(d_1(i, j), \dots, d_P(i, j))$  suffices to represent the deadline sequence.

WRR schedule for a traffic matrix  $M$  exists if and only if there exists a sequence of  $P$  permutation matrices  $\Pi_1, \Pi_2, \dots, \Pi_P$  such that  $\forall k = 1, 2, \dots, P$  and for each connection  $(i, j)$ ,

$$\sum_{l=1}^k \Pi_l(i, j) \geq \sum_{l=1}^k d_l(i, j). \quad (3.3)$$

To see this, let  $\Pi_1(i, j), \Pi_2(i, j), \dots, \Pi_k(i, j)$  indicate services received by a connection  $(i, j)$  by any time  $k$ . Equation 3.3 then states that, the number of services received by connection  $(i, j)$  in WRR schedule is greater than or equal to the number of packets of the connection that reached their deadlines by that

time. This is a necessary and sufficient condition for satisfying deadlines and so the existence of WRR schedule. Since Equation 3.3 is satisfied by equality if  $k = 1$ , the following equation follows;

$$\sum_{l=k}^P \Pi_l(i, j) \leq \sum_{l=k}^P d_l(i, j). \quad (3.4)$$

For each  $l = 1, 2, \dots, P$ , *deadline matrix* is a binary matrix defined as follows;

$$D_l \triangleq [d_l(i, j)]_{N \times N}.$$

Equation 3.4 is then equivalent to

$$\sum_{l=k}^P D_l - \sum_{l=k}^P \Pi_l \geq 0 \quad \forall k = 1, 2, \dots, P. \quad (3.5)$$

where the matrix inequality is componentwise. From the above definitions and basic concepts, the problem of guaranteeing specified services to any connection of deadline constrained traffic represented in a traffic matrix,  $M$ , can be expressed as finding  $P$  permutation matrices  $(\Pi_1, \Pi_2, \dots, \Pi_P)$  such that Equation 3.5 is satisfied. Proposition 1 guarantees that Equation 3.5 is satisfied by equality if lower bound of summation is  $k = 1$ . To provide WRR schedule the main task of the problem is then to devise a scheduling algorithm that will satisfy Equation 3.5 for all  $k$ .

A permutation matrix is said to be extractable from an integer matrix if the matrix obtained by subtracting the permutation matrix from the given matrix results in a positive matrix. The extraction process is performed by maximum matching algorithm which results to the extraction of full permutation matrices. Next, a submatrix is defined as follows.

**Definition 5.** An  $n \times m$  submatrix  $S = [s_{ij}]_{n \times m}$  of  $Q = [q_{ij}]_{N \times N}$  is identified by a row set  $\{r_1, \dots, r_n\}$  and a column set  $\{c_1, \dots, c_m\}$  such that;

$$s_{ij} = q_{r_i, c_j}, \quad i = 1, \dots, n \text{ and } j = 1, \dots, m.$$

Illustrating the notion of submatrix, consider an integer matrix  $Q$  as follows;

$$Q = \begin{bmatrix} 3 & 4 & 2 \\ 0 & 2 & 9 \\ 2 & 8 & 2 \end{bmatrix}.$$

An example of  $2 \times 1$  submatrix,  $S$ , of  $Q$  is then provided in Equation 3.6 which is identified by  $\{1, 2\}$  and  $\{1\}$ ; row and column sets of the matrix  $Q$  respectively.

$$S = \begin{bmatrix} 3 \\ 0 \end{bmatrix}. \quad (3.6)$$

The necessary and sufficient conditions for extractability of a permutation matrix is then presented by the following proposition.

**Proposition 2.** *A permutation matrix can be extracted from a nonnegative,  $N \times N$  integer matrix  $Q$  if and only if  $Q$  has no zero submatrix of size  $n \times m$  where  $m + n > N$ .*

The proof of the above proposition can be found in many combinatorics books, particularly see Theorem 2.4.3 in [37]. Noting that a switch with line utilization equal to 1 is considered, exactly  $N$  conflict free packets should be transmitted in each time slot. This is achieved by employing a full permutation matrix whose ones represent connections to receive service at that time slot. Therefore, for a scheduling algorithm to provide a WRR schedule it should guarantee extraction of the right order of full permutation matrices. By Proposition 1,  $P$  permutation matrices can always be extracted from a traffic matrix. The consistence of the problem formulated and Philp and Liu conjecture is analytically discussed in the next section.

### 3.4 Backward Extraction and Philp and Liu Conjecture

Philp and Liu conjecture states that a feasible schedule for periodic traffic always exists whenever links have utilizations less than or equal to 1.0. This conjecture was a result of exhaustive numerical experiments for searching feasible schedules of periodic messages whose results did not provide a counter example for small switch size [21]. Philp and Liu defined periodic traffic as a packet stream in which the number of time slots between packet arrivals (i.e. time to deadline) of a connection is constant. So, Philp and Liu formulation is a special case of the formulation in this work because this work considers services with possibly different deadlines less than or equal to periods of their connections. In particular, the situation considered by Philp and Liu arises in the current formulation if each connection  $(i, j)$  has service rate  $\rho_{ij} = \frac{1}{p_{ij}}$  for some integer  $p_{ij} \leq P$ . In this section, it is shown that the results of the formulated problem in this thesis work is consistent with Philp and Liu conjecture. This is illustrated by proving the following lemma and corollaries.

**Lemma 1.** *For any connection  $(i, j)$  and  $k = 1, 2, \dots, P$ ,*

$$\sum_{l=k}^P D_l(i, j) \geq \lceil (P - k + 1) \frac{m_{ij}}{P} \rceil. \quad (3.7)$$

*Proof.* Note that,

$$\sum_{l=1}^k D_l(i, j) = q \quad \text{if} \quad \lceil q \frac{P}{m_{ij}} \rceil \leq k < \lceil (q + 1) \frac{P}{m_{ij}} \rceil. \quad (3.8)$$

Using the left inequality in Equation 3.8 and the relation that  $\lceil x \rceil \geq x$ , Equation 3.8 implies that

$$\sum_{l=1}^k D_l(i, j) \leq k \frac{m_{ij}}{P}. \quad (3.9)$$

But

$$\sum_{l=1}^P D_l(i, j) = m_{ij},$$

which is equivalent to,

$$\sum_{l=1}^k D_l(i, j) + \sum_{l=k+1}^P D_l(i, j) = m_{ij}. \quad (3.10)$$

Putting Equation 3.9 in Equation 3.10, the following relation directly follows;

$$\sum_{l=k}^P D_l(i, j) \geq (P - k + 1) \frac{m_{ij}}{P}. \quad (3.11)$$

Since  $\sum_{l=k}^P D_l(i, j)$  is an integer then the following relation also true;

$$\sum_{l=k}^P D_l(i, j) \geq \lceil (P - k + 1) \frac{m_{ij}}{P} \rceil, \quad (3.12)$$

which completes the proof of the lemma.  $\square$

The lemma above provides a lower bound to elements of a matrix obtained by the backward summation of any  $k$  deadline matrices. The following corollary directly follows from Lemma 1.

**Corollary 1.**

$$\begin{aligned} \sum_{i=1}^N \sum_{l=k}^P D_l(i, j) &\geq P - k + 1, & j = 1, 2, \dots, N, \\ \sum_{j=1}^N \sum_{l=k}^P D_l(i, j) &\geq P - k + 1, & i = 1, 2, \dots, N. \end{aligned}$$

*Proof.* Summing Equation 3.12 over a column ( $j = 1, 2, \dots, N$ ) and using the relation  $\lceil x \rceil \geq x$  and the fact  $\sum_{i=1}^N m_{ij} = P$  imply that,

$$\sum_{i=1}^N \sum_{l=k}^P D_l(i, j) \geq P - k + 1.$$

Proceeding with similar procedures by considering sum over any row ( $i = 1, 2, \dots, N$ ), Equation 3.12 yields,

$$\sum_{j=1}^N \sum_{l=k}^P D_l(i, j) \geq P - k + 1.$$

which completes the proof.  $\square$

Extraction of permutation matrices is said to be *backward oriented* if the first permutation matrix,  $\Pi_P$ , is extracted from deadline matrix  $D_P$  and the  $k^{\text{th}}$  permutation matrix is extracted from  $\sum_{l=k}^P D_l - \sum_{l=k+1}^P \Pi_l$  for any  $k = 1, 2, \dots, P - 1$ . Backward oriented extraction is the backbone approach used in the proposed algorithms in Chapter 4. Corollary 1 reveals that the row sums and column sums of any  $k$  deadline matrices is at least  $P - k + 1$  if the summation is done in backward oriented manner. In general, if permutation matrices are extracted in backward oriented manner, at any time  $k$ , WRR schedule needs exactly  $P - k + 1$  permutation matrices. Hence, Corollary 1 provides some insight on the existence of permutation matrices by any time  $k$  if backward extraction is adopted. However, the corollary does not guarantee the existence of at least  $P - k + 1$  permutation matrices at any time  $k$ . For example, the following matrix,  $Q$ , satisfies the corollary but a permutation matrix can not be extracted from it.

$$Q = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \quad (3.13)$$

However, it can be shown that the above form of matrix is never encountered in the backward extraction process. This is illustrated by the following corollary.

**Corollary 2.** For any  $n \times m$  submatrix,  $S$ , of  $\sum_{l=k}^P D_l$  such that  $n + m > N$  and any  $k = 1, 2, \dots, P$ ,

$$\sum_{(i,j) \in S} \sum_{l=k}^P D_l(i,j) \geq (P - k + 1)(m + n - N).$$

*Proof.* Summing Equation 3.12 over all connections  $(i, j) \in S$  and using the relation  $\lceil x \rceil \geq x$  imply that,

$$\sum_{(i,j) \in S} \sum_{l=k}^P D_l(i,j) \geq \frac{(P - k + 1)}{P} \sum_{(i,j) \in S} m_{ij}. \quad (3.14)$$

Defining

$$M(S) \triangleq \sum_{(i,j) \in S} m_{ij},$$

and complementary submatrix,  $\hat{S}$ , of  $\sum_{l=k}^P D_l$  to submatrix  $S$  as a submatrix identified by rows and columns complement to the rows and columns of the submatrix  $S$ , i.e.,

$$\sum_{l=k}^P D_l = \left[ \begin{array}{c|c} \hat{S} & \\ \hline & S \end{array} \right],$$

then it follows that;

$$M(S) = M(\hat{S}) + (n + m - N)P, \quad \text{where } M \text{ is the traffic matrix.}$$

But  $M(\hat{S}) \geq 0$ , therefore

$$M(S) \geq (n + m - N)P. \quad (3.15)$$

Equation 3.14 then becomes,

$$\sum_{(i,j) \in S} \sum_{l=k}^P D_l(i, j) \geq \frac{(P - k + 1)}{P} M(S). \quad (3.16)$$

Putting Equation 3.15 in Equation 3.16 results to,

$$\sum_{(i,j) \in S} \sum_{l=k}^P D_l(i, j) \geq (P - k + 1)(n + m - N), \quad (3.17)$$

this completes the proof.  $\square$

Corollary 2 states that the sum of elements in any large submatrix  $S$  of a matrix obtained by backward summation of any  $k$  deadline matrices has a defined lower bound greater than zero. This rules out the possibility of having the a matrix of the form of Equation 3.13. It is interesting to note that if all rows and columns satisfy Corollary 1 by equality at any  $k$  then required permutation matrices always exists and can be extracted but it is not obvious if the corollary is satisfied by inequality as seen in Equation 3.13. However, Corollary 2 provides further insight in the possible existence of WRR schedule. This is the main reason of proposing heuristic algorithms as will be seen in Chapter 4. This argument is



also consistent with Philp and Liu conjecture as far as the existence of feasible schedule in this thesis work is concerned. Numerical analysis of the proposed algorithms discussed in Chapter 5 show that they are very close to satisfying the conjecture.

### 3.5 Delay Bounds for Periodic Traffic

It was mentioned in Section 3.3 that WRR schedule satisfies deadlines of all packets in a deadline constrained traffic. However, service times in WRR are out of phase with the arrival times of packets. This incurs some delay to a packet in a network. Satisfying deadline of a packet is equivalent to satisfying its worst delay bound in the network. In this section, the worst delay bound guaranteed by any WRR schedule under periodic traffic is derived.

**Claim 1.** *The delay of any packet in WRR schedule of periodic traffic is bounded by  $\lceil \frac{P}{m_{i,j}} \rceil$  time slots.*

*Proof.* Proving the above claim, is equivalent to proving the claim that there is at least one service between any two deadlines of a connection in a feasible schedule, or equivalently;

$$\sum_{l=\lceil \frac{k}{\rho_{ij}} \rceil}^{\lceil \frac{k+1}{\rho_{ij}} \rceil} \Pi_l(i, j) \geq \begin{cases} 1, & \text{if } m_{ij} > 0 \\ 0, & \text{else.} \end{cases}$$

This is true if and only if any packet in a connection is serviced between its deadline and the deadline of the previous service (inclusive) which is the case for any WRR schedule. But, any two deadlines are separated by at most  $\lceil \frac{1}{\rho_{ij}} \rceil$  time slots hence, any packet can be delayed by at most this amount of time in WRR schedule. This argument completes the proof of the claim.  $\square$

Deadline relaxation by  $n$  time slots is the term used when a packet can be further delayed by  $n$  time slots after its deadline is reached. If deadline of any

service is relaxed by  $n$  time slots, then the delay bound becomes  $\lceil \frac{1}{\rho_{ij}} \rceil + n$ . Delay bounds guarantees also results to fairness among the connections, as no connection can make other connections starving in any WRR schedule.

### 3.6 Feasible Schedule for $2 \times 2$ Switch

The notion of tracking policy was introduced in Section 2.2.4 where it was mentioned that tracking policy is a packetized policy designed to track generalized processor sharing (GPS) scheme [2, 3]. A tracking policy services each packet by the time it finishes its service under fluid policy. The number of packets transmitted under tracking policy should be at least the integer amount of information transmitted under fluid policy by that time and it should not exceed the information transmitted under fluid policy by more than one packet. This is the necessary and sufficient condition for the existence of tracking policy [23]. As a result, any scheduling algorithm which satisfies delay bounds of all packets of a given traffic (feasible schedule) is a tracking policy. In this section, the existence of a feasible schedule  $2 \times 2$  switch is proven. The same results as provided in this section has been recently available in terms of tracking policy [23]. This result however, is immediate by using the formulated problem in this chapter. This is elaborated by proving the claim below.

**Claim 2.** *A feasible schedule for any  $2 \times 2$  switch always exists.*

*Proof.* Proposition 2 reveals that a permutation matrix can not be extracted from any  $2 \times 2$  matrix if it contains either a  $2 \times 1$ ,  $1 \times 2$  zero submatrices or if  $2 \times 2$  matrix is a zero matrix. From Corollary 1, there exists enough elements in deadline matrices to extract the required number of permutation matrices at any time slot if the extraction is backward oriented. That is, there is at least one nonzero element in each row and column at any time slot. This argument rules out any possibility of having zero row(column) and so the whole matrix at any

time slot if a traffic matrix is  $2 \times 2$ . Therefore, permutation matrix can always be extracted.  $\square$

The existence of tracking policy (feasible schedule) for a general switch size is still an open problem. However, in [23], it is claimed that tracking policy for a general switch size always exists. This is strongly supported by Corollary 1 and Corollary 2.

### 3.7 Per-Port vs Per-VC Scheduling

So far, the problem formulated and its analysis assumed that there is only one connection or virtual circuit (VC) in each VOQ. This scheduling mechanism is called *per-port scheduling* or *transmission scheduling*. In more realistic scenarios, a single VOQ can accommodate more than one VC.

This work considers VCs scheduling technique as proposed in [38] in which the scheduling mechanism is separated into two stages, *VC scheduling* and *Per-Port scheduling*. The VC scheduling schedules packets from VCs to transmission queues of a switch; while port scheduling is responsible for resolving the potential conflict among packets from different VOQs. This mechanism is said to achieve scalability and flexibility of the scheduling process. The flexibility rises from the fact that scheduling algorithms from each stage can be designed independently; they can be the same or different. More important to note, VC scheduling is like output queue scheduling and so more sophisticated scheduling algorithms which provide advanced QoS requirements can be used if required. In this work,

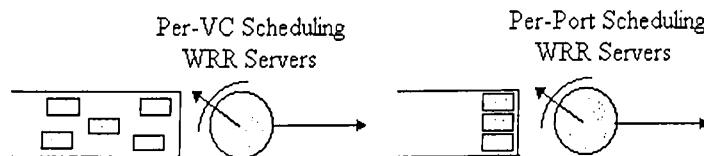


Figure 3.2: Per-VC vs Per-Port Scheduling.

a switch mechanism which uses simple WRR servers for both, VC and Port scheduling mechanisms as seen in Figure 3.2 is considered. There is no need of sophisticated scheduling algorithms for VC scheduling because constant rate traffic is assumed which can be efficiently scheduled by WRR servers. Once a packet is scheduled by a WRR server at VC buffer, it is transmitted to its VOQ where it waits to be scheduled. When VOQ is shared by more than one session, the elements of a traffic matrix represent the sum of number of packets of all sessions in that queue. Therefore, problem formulation and analysis as performed for per-port case only is the same when more than one session exists in the same VOQ. For example, the  $k^{th}$  deadline of the  $n^{th}$  session with  $m$  packets to be transferred from input  $i$  to output  $j$  is at time  $\lceil k \frac{P}{m_{n,i,j}} \rceil$ . In particular, a similar claim as was proved for per-port only case in section 3.5 is also presented for general scheduling (per-VC and per-port) case. The proof of the claim easily follows from the proof of per-port only scheduling version of the claim presented in Section 3.5.

**Claim 3.** *The delay of any packet of VC,  $(n, i, j)$ , in WRR schedule of periodic traffic is bounded by  $\lceil \frac{m_{n,i,j}}{P} \rceil$  time slots, where,  $m_{n,i,j}$  is the number of packets of the  $n^{th}$  VC to be destined to output port  $j$  from input port  $i$  in one schedule length.*

In the next chapter, scheduling algorithms for solving the formulated problem in this section are proposed.

# Chapter 4

## HEURISTIC ALGORITHMS

In this chapter, basic scheduling algorithm for scheduling deadline constrained traffic is proposed. The basic algorithm is incorporated with some heuristics to increase its performance. These heuristics are based on Oldest Deadline First (ODF) and Balancing Service Ratios (BSR) approaches to be defined later in this chapter. Deadline Relaxation (DR) heuristic is also employed to basic algorithm and its variants for further performance improvement. These algorithms are all backward oriented as presented in Section 3.4. The backward orientation of the problem is easier to analyze and provides more insight to the problem as noted in Chapter 3. This chapter is organized as follows; basic algorithm is presented in Section 4.1, and variants of the basic algorithm are discussed in Section 4.2.

### 4.1 Basic Algorithm

Given a traffic matrix,  $M$ , deadline matrix,  $D$ , is first computed. The algorithm then selects eligible sets of connections to be scheduled at each time slot from the deadline matrix in backward oriented technique introduced in Section 3.4. This is realized by fixing an  $N \times N$  empty matrix and shifting the deadline matrices

from  $D_P$  to  $D_1$ , one submatrix at each time slot, into the window matrix. The fixed matrix is called a *window matrix* denoted by  $W_k$ . The state of the window matrix at any time slot  $k$  is an  $N \times N$  matrix whose elements are the sum of ones representing the deadlines of unserved connections of deadline matrices already entered the window and  $D_k$ ; binary deadline matrix which entered the window matrix at time  $k$ . A permutation matrix is arbitrarily extracted if no priority measure is associated to existing permutation matrices during extraction. At any time slot  $k$ , the basic algorithm arbitrarily extracts a permutation matrix from a window matrix, by using *maximum matching* techniques. Maximum matching is necessary in all algorithms proposed in this work because only successful extraction of full permutation matrix in each time slot  $k$  from window matrix provides WRR schedule. When  $P$  permutation matrices are successfully extracted the order of permutation matrices as extracted by scheduling algorithm is reversed to get the service order. The basic algorithm is summarized in Figure 4.1.

---

**Initialization:**

Given a traffic matrix,  $M$ , compute the deadline matrices,  $D_l$ , for each  $l = 1, 2, \dots, P$ .

**Main Loop :**

**for**  $k = P : 1$  **do**

    Compute the current window's state:

$$W_k = \begin{cases} D_P, & \text{if } k = P \\ \sum_{l=k}^P D_l - \sum_{l=k+1}^P \Pi_l, & \text{else.} \end{cases}$$

    Extract a full permutation matrix,  $\Pi_k$ , by applying *maximum matching* algorithm to the window matrix,  $W_k$ .

**if** *no such permutation matrix exists* **then**

        The algorithm fails.

**end if**

**end for**

Use service order of permutation matrices as:  $\Pi_1, \Pi_2, \dots, \Pi_P$ .

---

Figure 4.1: The Basic Algorithm.

The basic algorithm results in a WRR schedule if at any time slot  $k$  in a schedule length,  $P$ , it is possible to extract a full permutation matrix from a

window matrix  $W_k$ . If a WRR schedule is obtained, then the same service order of connections sets is repeated during service in the next schedule lengths. Numerical results show that small percentage of trials fail when it is possible to extract full permutation matrices over all schedule length.

Arbitrary extraction of permutation matrices in basic algorithm however, does not always provide a feasible schedule. In all observed failures of basic algorithm, failures occur at time slot equal to half of the schedule length where the condition stated by Proposition 2 is satisfied, i.e. an  $n \times m$  zero submatrix in a window matrix exists such that  $n + m > N$ . A critical set exists if some connections sharing the same port have equal deadlines, i.e. when  $n$  connections at time  $k$  each with deadline equal to  $n + k$ . Critical set can be detected in traffic matrix or it can form at any time. Once it forms, a critical set needs to receive service in each time slot until all connections of the set are serviced. One or more critical sets were observed in all failures of the basic algorithm and in general, it is believed to be the cause of most failures. Detecting all critical sets in advance however, is a difficult task and involves much computational complexity. Heuristic algorithm which detects critical sets in advance is presented in [23]. Basic algorithm without critical set detection however, offers very promising performance as will be seen in Chapter 5. An example of failure of basic algorithm is illustrated by Table 4.1.

By observing the states of window matrix and the sum of previous service matrices  $(\Pi_P, \Pi_{P-1}, \dots, \Pi_{k-1})$  if the basic algorithm fails at time  $k$ , the following observations are noted:

1. Critical sets (e.g. rows and columns whose all elements are 4 in the example above) are detected in the traffic matrix or at the failure.
2. There are enough number of positive elements in window matrix to form a permutation matrix but fail to make a permutation pattern,

| $l$ | $D_l$  | $W_l$  | $\Pi_l$  |
|-----|--|--|--|
|     | $M = \begin{bmatrix} 5 & 4 & 4 & 3 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 3 & 4 & 4 & 5 \end{bmatrix} \quad \sum_{l=9}^{16} D_l = \begin{bmatrix} 3 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 3 \end{bmatrix} \quad \sum_{l=9}^{16} \Pi_l = \begin{bmatrix} 3 & 1 & 1 & 2 \\ 2 & 2 & 2 & 1 \\ 2 & 2 & 2 & 1 \\ 0 & 2 & 2 & 3 \end{bmatrix}$ |  |  |
| 16  | $\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$   | $\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |
| 15  | $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$   | $\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ | $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ |
| 14  | $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$   | $\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$ | $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$ |
| 13  | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$   | $\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |
| 12  | $\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$   | $\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$ |
| 11  | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$   | $\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 \end{bmatrix}$ | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ |
| 10  | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$   | $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 2 & 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |
| 9   | $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$   | $\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 \end{bmatrix}$ | <i>ERROR</i>   |

Table 4.1: Example of Failure of Basic Algorithm.



3. Next deadline matrix of  $D_{k-1}$  to enter into window matrix has many ones waiting,
5. There is at least one connection not serviced,
6. Inspection indicates that the algorithm might yield a feasible schedule by exercising different options for matrix extraction.

These are only the observed cases in all failures of basic algorithm, they may not be general. It is also important to note that the observed failures in basic algorithm are not complete failures in the sense that, there exists a set of permutations that can provide a feasible schedule. This is supported by the observation number 6 presented above. The driving intuition of this work is that, more sophisticated permutation selection measures may always provide a feasible schedule. Numerical results of the proposed algorithms are to be discussed in the next chapter.

## 4.2 Variations of Basic Algorithm

Heuristic algorithms proposed to improve the performance basic algorithm are motivated by observing failures of the basic algorithm. These heuristics basically induce priority measures to a permutation matrix to be selected or relaxing deadlines of services of some connections only when this is necessary to avoid failure.

### 4.2.1 Oldest Deadline First Approach

This algorithm is motivated by the fact that there exists at least one unserviced active connection whenever the basic algorithm fails. The algorithm imposes priorities to permutation matrices to be selected from a window matrix in case

more than one choice exists by favoring the one with maximum number of old deadline connections possible. And so the name *Oldest Deadline First Approach* implies. This heuristic algorithm is the same as the basic algorithm except that this algorithm selects all possible permutation matrices from the current window matrix before new deadline matrix enters. If connections with new deadlines exist in the window matrix the algorithm tries to select a permutation matrix with the most number of connections possible whose deadlines are due the soonest. This algorithm is summarized in Figure 4.2.

---

**Initialization:**

Given a traffic matrix,  $M$ , compute the deadline matrix,  $D_l$ , for each  $l = 1, 2, \dots, P$ .

Set  $k_1 = P$ .

Set  $k_2 = P + 1$ .

**Main Loop :**

**while** it is possible to extract  $\Pi_{k_2-1}$  from  $\sum_{l=k_1}^P D_l - \sum_{l=k_2}^P \Pi_l$  **do**

    Extract  $\Pi_{k_2-1}$ ,

$k_2 \leftarrow k_2 - 1$ ,

**end while**

$k_1 \leftarrow k_1 - 1$

**if**  $k_1 < k_2 - 1$  **then**

    The algorithm fails.

**else**

    Go to the while loop.

**end if**

Use service order of permutation matrices as:  $\Pi_1, \Pi_2, \dots, \Pi_P$ .

---

Figure 4.2: Oldest Deadline First Approach.

By imposing oldest first priority measure, the algorithm actually adopts *Earliest Deadline First* (EDF) approach, since the service order of permutation matrices is the reverse of their extracted order by the algorithm. Many failures under basic algorithm are resolved by this algorithm (e.g. example illustrated in Table 4.1) and as will be discussed in the Chapter 5, the performance is improved. The improvement in this algorithm asserts that if more sophisticated priority measures in selecting permutation matrix are applied to basic algorithm, a feasible

schedule may always be obtained. The algorithm however, may not always maximize the number of oldest deadline connections at each time slot, some of them may not be serviced in order to extract a full permutation matrix. Observation 6 as mentioned to basic algorithms failures also applies to ODF algorithm. An example of failure in oldest deadline first approach is presented in Table 4.2;

Observing the above example, one can see that the failure occurs at  $k = \frac{P}{2}$  as observed for basic algorithms. In contrast to the observed failures for basic algorithm however, this is not always the case for the failures observed for the oldest deadline first approach. At the failure of the above example, it can be easily verified that the service opportunities are not fairly distributed among connections. Connections with 11, 8 and 7 packets to be serviced in a schedule length have equal number of packets successfully scheduled before the failure. Similarly, a connection with 4 packets has successfully scheduled 1 packet the same as number of packets serviced by a connection with only 1 packet to be transmitted in a schedule length. This reveals that, although it improves the performance of basic algorithm, oldest deadline first approach does not provide services to connections relative to their service rates.

This observation motivates that, some service measures relative to service rates of each connection could be incorporated to basic algorithm and possibly results in better performance. That is, service opportunities could reflect service rates of connections. This observation leads to proposing another heuristic algorithm based on balancing service ratios of connections at any time slot which is discussed in the next section.

| $l$ | $D_l$   | $W_l$  | $\Pi_l$   |
|-----|---|--|---|
|     | $M = \begin{bmatrix} 11 & 1 & 2 & 2 \\ 1 & 7 & 4 & 4 \\ 2 & 4 & 8 & 2 \\ 2 & 4 & 2 & 8 \end{bmatrix}$ | $\sum_{l=9}^{16} D_l = \begin{bmatrix} 6 & 1 & 1 & 1 \\ 1 & 4 & 2 & 2 \\ 1 & 2 & 4 & 1 \\ 1 & 2 & 1 & 4 \end{bmatrix}$ | $\sum_{l=10}^{16} \Pi_l = \begin{bmatrix} 4 & 1 & 1 & 1 \\ 1 & 4 & 1 & 1 \\ 1 & 1 & 4 & 1 \\ 1 & 1 & 1 & 4 \end{bmatrix}$ |
| 16  | $\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$      | $\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$                       | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$                          |
| 15  | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$      | $\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$                       | $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$                          |
| 14  | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$      | $\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$                       | $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$                          |
| 13  | $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$      | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$                       | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$                          |
| 12  | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$      | $\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$                       | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$                          |
| 11  | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$      | $\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$                       | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$                          |
| 10  | $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$      | $\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$                       | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$                          |
| 9   | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$      | $\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$                       | <i>ERROR</i>  |

Table 4.2: Example of Failure of Oldest Deadline First Algorithm.

### 4.2.2 Balancing Service Ratios Approach

In this section another heuristic algorithm based on *balancing service ratios* of connections is presented. Service ratio of a connection  $(i, j)$  at time slot  $k$  is defined as the ratio of number of service received by connection  $(i, j)$  by time  $k$  to total services required by the connection in a schedule length. i.e.,

$$\text{Service Ratio} = \frac{s_k(i, j)}{m_{ij}}, \quad k = 1, 2, \dots, P, \quad (4.1)$$

where  $s_k(i, j)$  is the number of services received by connection  $(i, j)$  by the  $k^{\text{th}}$  time slot. Service ratios of all connections are zeros at the beginning of switch operation, they are all ones at the end of schedule length for any feasible schedule.

A traffic matrix whose all elements are equal, for example, results to a WRR schedule if and only if a certain connection receives service again after each connection in the same input(output) port has received exactly one service opportunity. The number of services received by any connection never exceeds the other by more than 1. This elucidates the importance of keeping services received by each connection corresponds to its service rate. Another obvious example is a connection with service rate equal to 1.0, i.e.  $\rho = 1.0$ . This connection is the only active connection of its input and output ports (by definition of traffic matrix, it is the only nonzero element in its row and column). It therefore needs to be serviced at each time slot. Doing so, the Equation 4.1 is related to  $\frac{k}{P}$  by equality. This example is assumed to be the ideal case in which services are directly related to the service rate of a connection in all time slots. Balancing serving ratios approach tries to make service ratios of all other connections mimic this ideal service ratio and so the ratio  $\frac{k}{P}$  at any time slot  $k$ .

When elements of a traffic matrix in the same row(column) are not equal, balancing their service ratios is not obvious and can not be maintained at a constant bound over all schedule length. That is, Equation 4.1 and ratio  $\frac{k}{P}$  is not always related by equality. Since service ratios are equal before scheduling

and at the end of schedule length of any feasible schedule, intuitively, making all connections emulate the ideal service ratio would balance service ratios in some sense. This is achieved by servicing as many connections possible with smallest service ratios in their corresponding input ports. The algorithm is summarized in Figure 4.3.

---

**Initialization:**

Given a traffic matrix,  $M$ , compute the deadline matrix,  $D_l$ , for each  $l = 1, 2, \dots, P$ .

**Main Loop :**

**for**  $k = P : 1$  **do**

    Compute the current window's state:

$$W_k = \begin{cases} D_P, & \text{if } k = P \\ \sum_{l=k}^P D_l - \sum_{l=k+1}^P \Pi_l, & \text{else.} \end{cases}$$

**for**  $j = 1 : k$  **do**

        By applying *maximum matching* algorithm to window matrix  $W_k$ , extract the first full permutation matrix,  $\Pi_k$ , which satisfies;

$$\frac{s_{k-1}(i, j)}{m_{ij}} \leq \frac{j}{P}. \quad (4.2)$$

**end for**

**if** *no such permutation matrix exists* **then**

        The algorithm fails.

**end if**

**end for**

Use service order of permutation matrices as:  $\Pi_1, \Pi_2, \dots, \Pi_P$ .

---

Figure 4.3: Balancing Service Ratios Approach.

Equation 4.2 is called eligibility condition. This algorithm tries to improve service fairness among backlogged connections. Just like previously proposed algorithms in this thesis, this algorithm also fails to provide WRR schedule for some traffic matrices. However, it successfully schedules many failures of basic algorithm (e.g. example in Table 4.1) and some failures of oldest deadline first approach (e.g. example in Table 4.2). The performance of this algorithm is better than the performance of the basic algorithm but closely the same as the oldest

deadline first approach. Its improvement over the basic algorithm elaborates that inducing extra fairness measures to the basic algorithm results in performance increase. The quantitative comparison of these algorithms is presented in Chapter 5.

### 4.2.3 Deadline Relaxations

When analyzing failures of basic algorithm and its variants, it is noted that there are always some packets just to enter into the window matrix whenever algorithms fail. This motivates the proposal of other heuristic algorithms which are based on relaxing deadlines. In contrast to the previous work in [21] whereby deadline relaxation means relaxing deadlines of all packets in a traffic, in this heuristic, deadlines are relaxed by one time slot only if this is necessary to avoid failure of the algorithm. This heuristic is applied to basic algorithm, oldest first approach and balancing service ratios approach. This heuristic results to nearly perfect performance for all algorithms. As will be seen in the discussion of simulation results in the next chapter, in almost all cases this heuristic provides WRR schedule. Deadline relaxation however, results in an increase in buffer size. Deadline Relaxation heuristic is adopted to the basic algorithm and its variants by adding the condition presented in Figure 4.4 whenever they fail.

---

**Run the algorithm; BA, ODF, or BDR:**

**if** *the algorithm fails at time k* **then**

    Modify the window matrix as;

$$W_k \leftarrow W_k + D_{k-1}.$$

    Extract  $\Pi_k$  and  $\Pi_{k+1}$  from  $W_k$

**if** *either  $\Pi_k$  or  $\Pi_{k+1}$  does not exist* **then**

        Deadline relaxation by 1 time slot fails.

**end if**

**end if**

---

Figure 4.4: Deadline Relaxation by 1 time slot.

# Chapter 5

## NUMERICAL RESULT

### 5.1 Traffic Matrix Generation

Given the required switch size  $N$  and schedule length  $P$ , traffic matrix generation starts by generating  $P$ ,  $N \times N$  random permutation matrices,  $\hat{\Pi}_1, \hat{\Pi}_2, \dots, \hat{\Pi}_P$ . By Proposition 1, the sum of these permutation matrices is a random traffic matrix. Let  $M_0$  is the initial random traffic matrix generated among an infinite number of possible  $N \times N$  matrices with row(column) sums  $P$ , i.e.,

$$M_0 = \hat{\Pi}_1 + \hat{\Pi}_2 + \hat{\Pi}_3 + \dots + \hat{\Pi}_P.$$

It is assumed that  $M_0$  is the initial state of an infinite states Markov Chain (MC) with states  $M_0, M_1, M_2, \dots$ , which are  $N \times N$  integer matrices with row(column) sums  $P$ . At any step, two random elements whose rows and columns enclose a square submatrix in the current traffic matrix are selected. Another traffic matrix is obtained by subtracting 1 from the selected elements and adding one to two other elements which are at the intersection of columns and rows of two randomly selected elements. Equation 5.1 elaborates this where  $A$  and  $B$  in the first matrix are the randomly selected elements. It is assumed that rows and columns of  $A$  and  $B$  enclose a square matrix. Unity is subtracted from  $A$  and



| $N$ | $P$             | Number of trials per $(N, P)$ |
|-----|-----------------|-------------------------------|
| 4   | 8,16, ..., 160  | 10,000                        |
| 8   | 16,32, ..., 320 | 10,000                        |
| 16  | 32,64, ..., 640 | 5,000                         |

Table 5.1: Matrix parameters used in simulations.

$B$  and added to  $C$  and  $D$  to obtain another random matrix. This is equivalent to moving from one state of the underlined MC to another. In equilibrium,  $M$  is uniformly distributed over all traffic matrices with same  $N$  and  $P$  parameters and is chosen as;

$$M = \lim_{k \rightarrow \infty} M_k.$$

There exist finitely many traffic matrices with the same  $N$  and  $P$  parameters. A very small set is actually considered in the simulations of this work. The randomly looking traffic matrix is considered so as to provide a good sample of traffic matrices to be used in simulations.

$$M_k = \begin{bmatrix} \vdots & & & \vdots \\ \vdots & C & \dots & B & \vdots \\ \vdots & & & & \vdots \\ \vdots & A & \dots & D & \vdots \\ \vdots & & & & \vdots \end{bmatrix}, \quad M_{k+1} = \begin{bmatrix} \vdots & & & \vdots \\ \vdots & C+1 & \dots & B-1 & \vdots \\ \vdots & & & & \vdots \\ \vdots & A-1 & \dots & D+1 & \vdots \\ \vdots & & & & \vdots \end{bmatrix}. \quad (5.1)$$

The parameters used in simulations for all algorithms are tabulated in Table 5.1.

## 5.2 Discussion of Numerical Results

The simulations were done using *MATLAB* programming in which a builtin function which solves a maximum matching problem `dmperm()` is used. The same set of traffic matrices is used to all algorithms for any  $(N, P)$  parameter

set. Success rate determines how often a scheduling algorithm provides WRR schedule. The simulations involve three experiments.

The first experiment was to determine success rates of algorithms without deadline relaxation. By observing the tabulated results in Tables 5.2, 5.3, and 5.4 and Figures 5.1, 5.2 and 5.3, one can easily deduce that success rates of all algorithms are above 95%. As expected, heuristic algorithms perform better than basic algorithm. The performance seem to increase as switch size and schedule length increase, the possible reason behind this phenomenon is to be discussed later in this section. Oldest deadline first approach and balancing service ratios have about the same performance.

The second experiment was performed to determine the success rates of proposed algorithms when deadline relaxation based heuristic is used. Simulation results can be seen in Tables 5.2, 5.3, and 5.4. They are all ones for  $N = 4$  for all algorithms. For  $N = 8, 16$  the success rates of basic algorithms are above 99.9% for all values of schedule lengths. Success rates are all ones for oldest deadline first and balancing service ratios approaches with the exception of very few cases for  $N = 8$  where the algorithms result success rates of above 99.9%.

The third experiment was to determine the percentage of number of packets that miss their deadlines when deadline relaxation is used. Figures 5.4, 5.5 and 5.6 and Tables 5.5, 5.6, and 5.7 show the numerical results. In all cases the percentage of packets that miss their deadlines is less than 0.5%, the performance is noted to increase as schedule length and switch size increase. In general, percentage of packets which miss their deadlines seems to be reasonably small for all algorithms. This percentage is much smaller for oldest deadline first and balancing service ratios, and it decreases as switch sizes and schedule length increase.

| $P$ | $BA$   | $ODF$  | $BSR$  | $BA(DR)$ | $ODF(DR)$ | $BSR(DR)$ |
|-----|--------|--------|--------|----------|-----------|-----------|
| 8   | 0.9857 | 0.9870 | 0.9872 | 1.0000   | 1.0000    | 1.0000    |
| 16  | 0.9772 | 0.9783 | 0.9800 | 1.0000   | 1.0000    | 1.0000    |
| 24  | 0.9666 | 0.9669 | 0.9675 | 1.0000   | 1.0000    | 1.0000    |
| 32  | 0.9744 | 0.9764 | 0.9763 | 1.0000   | 1.0000    | 1.0000    |
| 40  | 0.9676 | 0.9730 | 0.9710 | 1.0000   | 1.0000    | 1.0000    |
| 48  | 0.9671 | 0.9695 | 0.9709 | 1.0000   | 1.0000    | 1.0000    |
| 56  | 0.9712 | 0.9760 | 0.9763 | 1.0000   | 1.0000    | 1.0000    |
| 64  | 0.9732 | 0.9746 | 0.9764 | 1.0000   | 1.0000    | 1.0000    |
| 72  | 0.9742 | 0.9787 | 0.9765 | 1.0000   | 1.0000    | 1.0000    |
| 80  | 0.9755 | 0.9792 | 0.9797 | 1.0000   | 1.0000    | 1.0000    |
| 88  | 0.9775 | 0.9799 | 0.9805 | 1.0000   | 1.0000    | 1.0000    |
| 96  | 0.9723 | 0.9748 | 0.9753 | 1.0000   | 1.0000    | 1.0000    |
| 104 | 0.9796 | 0.9838 | 0.9834 | 1.0000   | 1.0000    | 1.0000    |
| 112 | 0.9798 | 0.9866 | 0.9860 | 1.0000   | 1.0000    | 1.0000    |
| 120 | 0.9776 | 0.9830 | 0.9844 | 1.0000   | 1.0000    | 1.0000    |
| 128 | 0.9790 | 0.9867 | 0.9875 | 1.0000   | 1.0000    | 1.0000    |
| 136 | 0.9765 | 0.9904 | 0.9898 | 1.0000   | 1.0000    | 1.0000    |
| 144 | 0.9800 | 0.9860 | 0.9864 | 1.0000   | 1.0000    | 1.0000    |
| 152 | 0.9870 | 0.9902 | 0.9906 | 1.0000   | 1.0000    | 1.0000    |
| 160 | 0.9862 | 0.9923 | 0.9925 | 1.0000   | 1.0000    | 1.0000    |

Table 5.2: Success rates for  $N = 4$  and 10000 trials per  $P$ : BA = Basic algorithm, ODF = Oldest deadline first approach, BSR= Balancing service ratios approach, DR = Deadline relaxation.

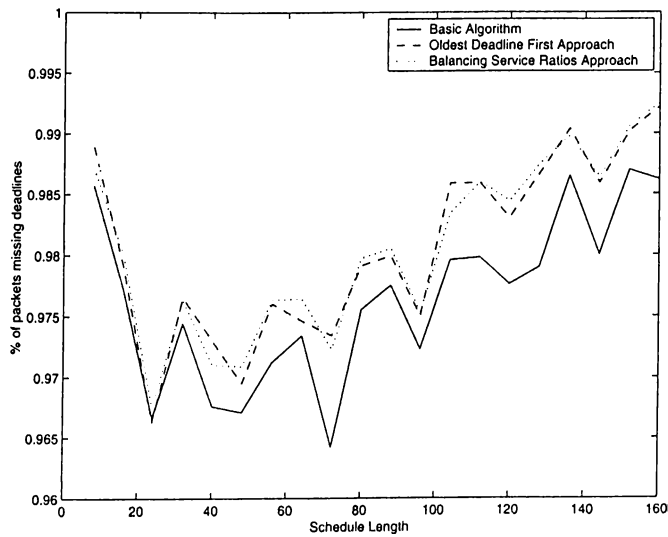


Figure 5.1: Success rates of Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach without deadline relaxation for  $N = 4$  and 10000 trials per schedule length.

| $P$ | $BA$   | $ODF$  | $BSR$  | $BA(DR)$ | $ODF(DR)$ | $BSR(DR)$ |
|-----|--------|--------|--------|----------|-----------|-----------|
| 16  | 0.9610 | 0.9712 | 0.9677 | 1.0000   | 0.9998    | 0.9997    |
| 32  | 0.9510 | 0.9794 | 0.9795 | 0.9996   | 1.0000    | 0.9999    |
| 48  | 0.9509 | 0.9834 | 0.9834 | 0.9998   | 0.9999    | 0.9999    |
| 64  | 0.9623 | 0.9865 | 0.9872 | 0.9998   | 1.0000    | 1.0000    |
| 80  | 0.9676 | 0.9912 | 0.9914 | 0.9999   | 1.0000    | 1.0000    |
| 96  | 0.9633 | 0.9912 | 0.9934 | 0.9998   | 1.0000    | 1.0000    |
| 112 | 0.9722 | 0.9946 | 0.9953 | 0.9998   | 0.9999    | 0.9999    |
| 128 | 0.9787 | 0.9948 | 0.9965 | 0.9998   | 1.0000    | 1.0000    |
| 144 | 0.9756 | 0.9968 | 0.9958 | 0.9998   | 1.0000    | 1.0000    |
| 160 | 0.9787 | 0.9961 | 0.9968 | 0.9998   | 1.0000    | 1.0000    |
| 176 | 0.9825 | 0.9968 | 0.9978 | 0.9999   | 1.0000    | 1.0000    |
| 192 | 0.9766 | 0.9961 | 0.9970 | 1.0000   | 1.0000    | 1.0000    |
| 208 | 0.9819 | 0.9978 | 0.9971 | 1.0000   | 1.0000    | 1.0000    |
| 224 | 0.9813 | 0.9982 | 0.9982 | 1.0000   | 1.0000    | 1.0000    |
| 240 | 0.9822 | 0.9983 | 0.9980 | 0.9999   | 1.0000    | 1.0000    |
| 256 | 0.9846 | 0.9985 | 0.9986 | 1.0000   | 1.0000    | 1.0000    |
| 272 | 0.9847 | 0.9982 | 0.9989 | 0.9999   | 1.0000    | 1.0000    |
| 288 | 0.9843 | 0.9984 | 0.9986 | 1.0000   | 1.0000    | 1.0000    |
| 304 | 0.9849 | 0.9987 | 0.9990 | 0.9999   | 1.0000    | 1.0000    |
| 320 | 0.9849 | 0.9986 | 0.9988 | 1.0000   | 1.0000    | 1.0000    |

Table 5.3: Success rates,  $N = 8$  and 10000 trials per  $P$  :  $BA$  = Basic algorithm,  $ODF$  = Oldest deadline first approach,  $BSR$  = Balancing service ratios approach,  $DR$  = Deadline relaxation.

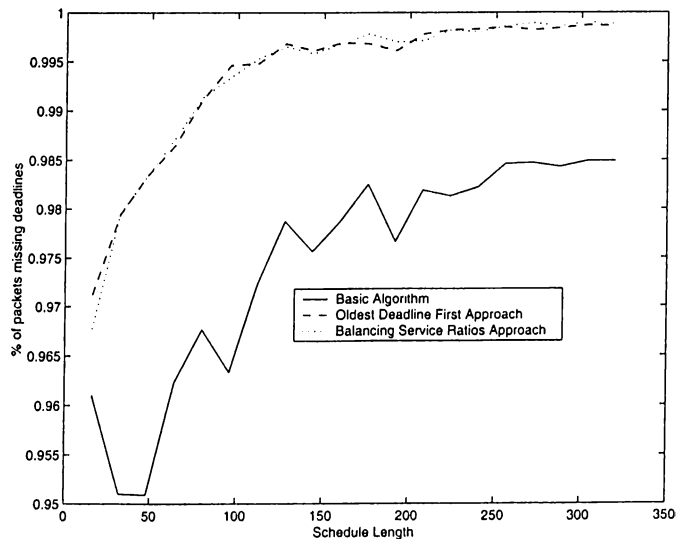


Figure 5.2: Success rates of Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach without deadline relaxation for  $N = 8$  and 10000 trials per schedule length.

| $P$ | $BA$   | $ODF$  | $BSR$  | $BA(DR)$ | $ODF(DR)$ | $BSR(DR)$ |
|-----|--------|--------|--------|----------|-----------|-----------|
| 32  | 0.9638 | 0.9816 | 0.9814 | 1.0000   | 1.0000    | 1.0000    |
| 64  | 0.9690 | 0.9942 | 0.9948 | 0.9998   | 1.0000    | 1.0000    |
| 96  | 0.9854 | 0.9996 | 0.9994 | 1.0000   | 1.0000    | 1.0000    |
| 128 | 0.9918 | 0.9996 | 0.9998 | 1.0000   | 1.0000    | 1.0000    |
| 160 | 0.9928 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 192 | 0.9942 | 1.0000 | 1.0000 | 0.9998   | 1.0000    | 1.0000    |
| 224 | 0.9966 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 256 | 0.9980 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 288 | 0.9966 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 320 | 0.9988 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 352 | 0.9984 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 384 | 0.9978 | 1.0000 | 1.0000 | 0.9998   | 1.0000    | 1.0000    |
| 416 | 0.9976 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 448 | 0.9992 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 480 | 0.9988 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 512 | 0.9986 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 544 | 0.9986 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 576 | 0.9984 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 608 | 0.9992 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |
| 640 | 0.9990 | 1.0000 | 1.0000 | 1.0000   | 1.0000    | 1.0000    |

Table 5.4: Success rates,  $N = 16$  and 5000 trials per  $P$ : BA = Basic algorithm, ODF = Oldest deadline first approach, BSR= Balancing service ratios approach, DR = Deadline relaxation.

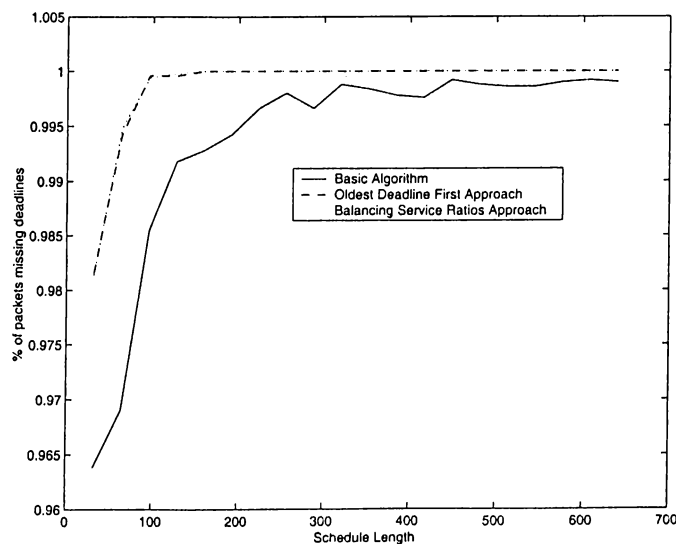


Figure 5.3: Success rates of Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach without deadline relaxation for  $N = 16$  and 5000 trials per schedule length.

| $P$ | $BA(DR)$ | $ODF(DR)$ | $BSR(DR)$ |
|-----|----------|-----------|-----------|
| 8   | 0.0591   | 0.0513    | 0.0513    |
| 16  | 0.0492   | 0.0469    | 0.0419    |
| 24  | 0.0470   | 0.0464    | 0.0447    |
| 32  | 0.0295   | 0.0267    | 0.0266    |
| 40  | 0.0289   | 0.0225    | 0.0229    |
| 48  | 0.0248   | 0.0218    | 0.0196    |
| 56  | 0.0187   | 0.0148    | 0.0144    |
| 64  | 0.0158   | 0.0145    | 0.0129    |
| 72  | 0.0183   | 0.0128    | 0.0134    |
| 80  | 0.0113   | 0.0089    | 0.0088    |
| 88  | 0.0094   | 0.0080    | 0.0077    |
| 96  | 0.0107   | 0.0093    | 0.0091    |
| 104 | 0.0076   | 0.0055    | 0.0059    |
| 112 | 0.0069   | 0.0042    | 0.0045    |
| 120 | 0.0071   | 0.0052    | 0.0044    |
| 128 | 0.0073   | 0.0037    | 0.0034    |
| 136 | 0.0038   | 0.0026    | 0.0027    |
| 144 | 0.0055   | 0.0032    | 0.0032    |
| 152 | 0.0033   | 0.0023    | 0.0023    |
| 160 | 0.0035   | 0.0019    | 0.0018    |

Table 5.5: Percentage of packets missing their deadlines by 1 time slot,  $N = 4$  and 10000 trials per  $P$ : BA = Basic algorithm, ODF = Oldest deadline first approach, BSR= Balancing service ratios approach, DR = Deadline Relaxation.

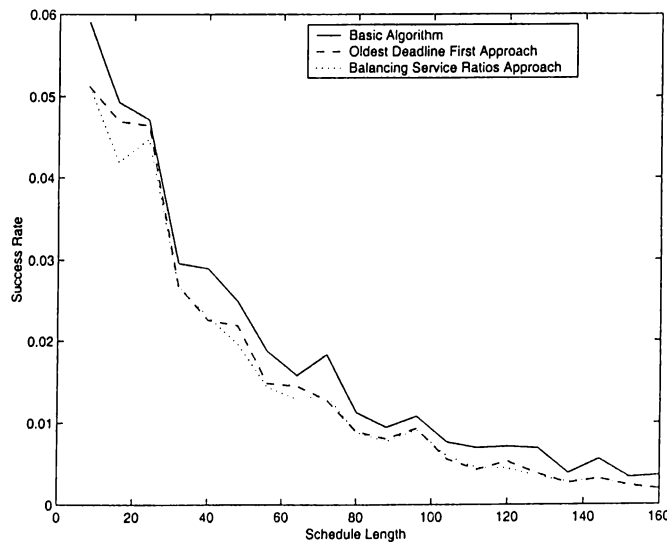


Figure 5.4: Percentage of packets missing deadline by 1 time slot for Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach with one time slot deadline relaxation for  $N = 4$  and 10000 trials per schedule length.

| $P$ | $BA(DR)$ | $ODF(DR)$ | $BSR(DR)$ |
|-----|----------|-----------|-----------|
| 16  | 0.1188   | 0.0795    | 0.0909    |
| 32  | 0.0850   | 0.0325    | 0.0323    |
| 48  | 0.0567   | 0.0172    | 0.0178    |
| 64  | 0.0348   | 0.0114    | 0.0103    |
| 80  | 0.0249   | 0.0060    | 0.0059    |
| 96  | 0.0232   | 0.0031    | 0.0037    |
| 112 | 0.0163   | 0.0025    | 0.0021    |
| 128 | 0.0111   | 0.0014    | 0.0017    |
| 144 | 0.0107   | 0.0015    | 0.0016    |
| 160 | 0.0086   | 0.0010    | 0.0011    |
| 176 | 0.0065   | 0.0011    | 0.0007    |
| 192 | 0.0080   | 0.0012    | 0.0009    |
| 208 | 0.0060   | 0.0006    | 0.0008    |
| 224 | 0.0056   | 0.0004    | 0.0005    |
| 240 | 0.0050   | 0.0004    | 0.0005    |
| 256 | 0.0041   | 0.0004    | 0.0005    |
| 272 | 0.0039   | 0.0004    | 0.0002    |
| 288 | 0.0038   | 0.0004    | 0.0004    |
| 304 | 0.0033   | 0.0003    | 0.0002    |
| 320 | 0.0033   | 0.0002    | 0.0003    |

Table 5.6: Percentage of packets which miss their deadlines for  $N = 8$  and 10000 trials per schedule length: BA = Basic algorithm, ODF = Oldest deadline first approach, BSR= Balancing service ratios approach.

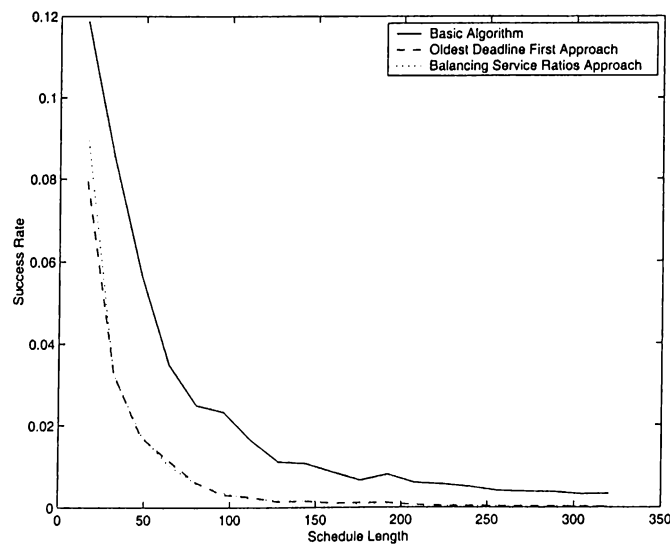


Figure 5.5: Percentage of packets which miss their deadlines for Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach with one time slot deadline relaxation for  $N = 8$  and 10000 trials per schedule length.

| $P$ | $BA(DR)$ | $ODF(DR)$ | $BSR(DR)$ |
|-----|----------|-----------|-----------|
| 32  | 0.0473   | 0.0236    | 0.0238    |
| 64  | 0.0285   | 0.0045    | 0.0042    |
| 96  | 0.0100   | 0.0002    | 0.0003    |
| 128 | 0.0048   | 0.0002    | 0.0001    |
| 160 | 0.0033   | 0.0000    | 0.0000    |
| 192 | 0.0021   | 0.0000    | 0.0000    |
| 224 | 0.0012   | 0.0000    | 0.0000    |
| 256 | 0.0006   | 0.0000    | 0.0000    |
| 288 | 0.0008   | 0.0000    | 0.0000    |
| 320 | 0.0003   | 0.0000    | 0.0000    |
| 352 | 0.0004   | 0.0000    | 0.0000    |
| 384 | 0.0004   | 0.0000    | 0.0000    |
| 416 | 0.0004   | 0.0000    | 0.0000    |
| 448 | 0.0002   | 0.0000    | 0.0000    |
| 480 | 0.0002   | 0.0000    | 0.0000    |
| 512 | 0.0002   | 0.0000    | 0.0000    |
| 544 | 0.0002   | 0.0000    | 0.0000    |
| 576 | 0.0001   | 0.0000    | 0.0000    |
| 608 | 0.0001   | 0.0000    | 0.0000    |
| 640 | 0.0001   | 0.0000    | 0.0000    |

Table 5.7: Percentage of packets which miss their deadlines for  $N = 16$  and 5000 trials per schedule length: BA = Basic algorithm, ODF = Oldest deadline first approach, BSR= Balancing service ratios approach.

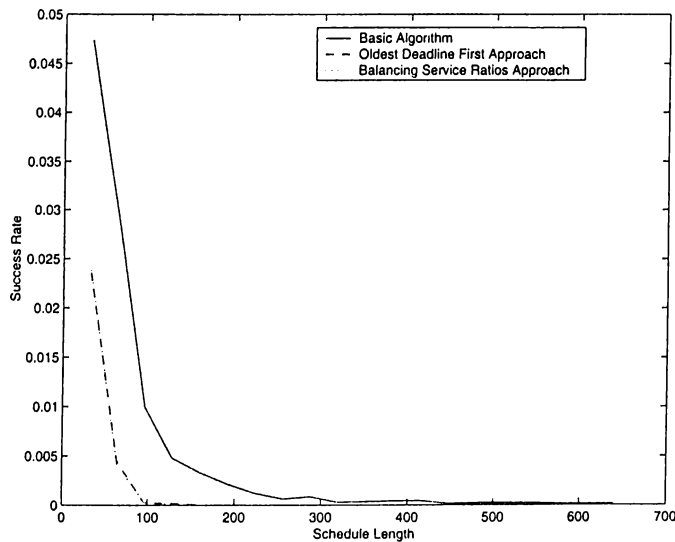


Figure 5.6: Percentage of packets which miss their deadlines for Basic Algorithm, Oldest Deadline First Approach and Balancing Service Ratios Approach with one time slot deadline relaxation for  $N = 16$  and 5000 trials per schedule length.



| P   | N = 4  |         |         | N = 8  |         |         | N = 16 |         |         |
|-----|--------|---------|---------|--------|---------|---------|--------|---------|---------|
|     | BA(DR) | ODF(DR) | BSR(DR) | BA(DR) | ODF(DR) | BSR(DR) | BA(DR) | ODF(DR) | BSR(DR) |
| 2N  | 0      | 0       | 0       | 0      | 6       | 10      | 0      | 12      | 0       |
| 4N  | 0      | 0       | 0       | 17     | 0       | 3       | 8      | 8       | 0       |
| 6N  | 0      | 0       | 0       | 10     | 2       | 2       | 0      | 0       | 0       |
| 8N  | 0      | 0       | 0       | 7      | 0       | 0       | 0      | 0       | 0       |
| 10N | 0      | 0       | 0       | 4      | 0       | 0       | 0      | 0       | 0       |
| 12N | 0      | 0       | 0       | 8      | 0       | 0       | 11     | 0       | 0       |
| 14N | 0      | 0       | 0       | 8      | 3       | 3       | 0      | 0       | 0       |
| 16N | 0      | 0       | 0       | 11     | 0       | 0       | 0      | 0       | 0       |
| 18N | 0      | 0       | 0       | 9      | 0       | 0       | 0      | 0       | 0       |
| 20N | 0      | 0       | 0       | 11     | 0       | 0       | 0      | 0       | 0       |
| 22N | 0      | 0       | 0       | 2      | 0       | 0       | 0      | 0       | 0       |
| 24N | 0      | 0       | 0       | 0      | 0       | 0       | 14     | 0       | 0       |
| 26N | 0      | 0       | 0       | 0      | 0       | 0       | 0      | 0       | 0       |
| 28N | 0      | 0       | 0       | 0      | 0       | 0       | 0      | 0       | 0       |
| 30N | 0      | 0       | 0       | 5      | 0       | 0       | 0      | 0       | 0       |
| 32N | 0      | 0       | 0       | 0      | 0       | 0       | 0      | 0       | 0       |
| 34N | 0      | 0       | 0       | 3      | 0       | 0       | 0      | 0       | 0       |
| 36N | 0      | 0       | 0       | 0      | 0       | 0       | 0      | 0       | 0       |
| 38N | 0      | 0       | 0       | 5      | 0       | 0       | 0      | 0       | 0       |
| 40N | 0      | 0       | 0       | 0      | 0       | 0       | 0      | 0       | 0       |

Table 5.8: Number of packets which miss deadline by 2 time slots when deadline relaxation is used.

Finally, the number of packets which miss their deadlines by two time slots was determined. In most cases no packet misses deadline by more than 2 time slots even for basic algorithms, see Table 5.8. Tabulated results in Table 5.8 also show that no packet misses its deadline by more than 1 time slot for  $N = 4$ . In few experimented cases for  $N = 8$  and  $N = 16$  small number of packets miss their deadlines by two time slots. The number of packets which miss their deadlines by one time slots is higher for basic algorithm than oldest deadline first approach and balancing service ratios approach. No packet misses deadline by more than one time slot when deadline relaxation is used.

It was noted before that the performance of algorithms increases as schedule length and switch size increase. By Proposition 2, it is can be verified that

sparseness of a matrix can reduce the likelihood of extracting a permutation matrix. This is very difficult to quantify since the most sparse possible traffic matrix is the matrix with all nonzero elements in a single permutation pattern and all equal to  $P$ . All proposed algorithms provide a feasible schedule under this traffic matrix. For small schedule lengths, the possibility that a traffic matrix is more sparse is higher than when the schedule length is larger according to traffic matrix generation procedure discussed in Section 5.1. This is a possible reason behind the increase in performance as schedule length increases. Similarly, number of possible permutation patterns for a switch with size  $N$  is  $N!$ . Obviously it is a huge number for switches with large switch sizes. The increase in the number of permutation patterns as switch size increases is also a possible reason for the increase in performance for large  $N$ .

It is difficult to compare oldest first approach to balancing service ratios approach. Numerical results show that success rates and percentage of number of packets which miss deadlines in these algorithms are almost equal. In most cases balancing service ratio approach shows slightly better performance. Their major difference however lies on the fact that balancing service ratios distributes service opportunities to connections more fairly.

Simulation results of this work are much more promising than the results of the previous works in [20,21,23]. This thesis work considers switch working at full capacity in contrast to all noted previous works. In [20,21], link utilization of 0.85 for all simulations was used. Simulation results in [20] show peak success rate of 0.6 for  $N = 4$ , it is worse for larger  $N$  values. Generally, algorithms proposed in [20,21] perform inversely proportional to the schedule length, switch size and link utility. Heuristic algorithms proposed in [20] seem to give better success rates but the performance deteriorates as  $P, N$  and link utilization increase. Finally, these algorithms and their heuristics are based on the fact that the switch utility

used is strictly less than 1.0, if link utilization is exactly 1.0, some heuristics are not applicable.

Similarly, the simulation results in [23] used links utility around 0.92. The percentage of number of packets missing their deadlines by the proposed tracking policy is larger compared to the proposed algorithms in this thesis. Moreover, the computational complexity is very high because the algorithm involves detecting critical sets in each time slot.

# Chapter 6

## SUMMARY

In this thesis work, the problem of scheduling packets in input queued packet switches is explored. In particular, scheduling multi-periodic traffic using input queued weighted round robin (WRR) servers is considered. Basic algorithm and its variants are proposed. These heuristics are balancing connections service ratios and servicing oldest deadline connection first. The other heuristic is deadline relaxation which can be adopted to any of the above algorithms whenever a failure is about to occur.

Proposed algorithms can be offline which are commonly used to exploit the flexibility and simplicity while maintaining reliability requirement. In these algorithms, complexity of operation is eliminated and so they can suit high speed networks better. The algorithms can also be online, whereby extraction of permutation matrices is desired to be *forward oriented* and storage units to store permutation matrices are unnecessary. Online algorithms however, must have low complexity so that they can suit high speed networks. In general, the proposed algorithms are simple and efficient because they do not involve much computation such as detecting critical sets.

Numerical experiments of this work are not exhaustive, limited subsets of infinitely many possible traffic matrices are considered. However, by randomly generating each traffic matrix, it is believed that healthy sample of traffic matrices is used. Simulations show that success rates of all algorithms are improved (close to 1.0) compared to previous works. Moreover, the performance of proposed algorithms is not negatively affected by the increase of switch dimension and schedule length and it is independent of link utilization. The performance increases as switch size and schedule length increase for all proposed algorithms. The percentage of number of packets which miss their deadlines by 1 time slot is small without deadline relaxations heuristic. In most cases the percentage of packets which miss their deadlines by 1 time slot is zero if deadline relaxation is employed. This can be an acceptable performance in some applications.

The result of exhaustive searches of feasible schedules presented in [20, 21] led Philp and Liu to conjecture that there always exists a feasible schedule for periodic traffic if switch utilization is less than or equal to 1.0. The analysis of the formulated problem and simulation results in this thesis are consistent with Philp and Liu conjecture and support the existence tracking policy for a general switch size as claimed in [23]. In contrast to this thesis work, despite assuming links with utilizations strictly less than 1.0, previously proposed algorithms provide low success rates. The performance worsens as switch dimension, schedule length and/or link utilization increase.

The hardware implementation of algorithms is possible and simple by making use of simplicity of WRR servers, no speedup is required and buffer size can be minimized to the size of one packet if no deadline relaxation is employed. Moreover, this work contributes to providing a better insight of the problem by presenting a new formulation. Simulation results and analyses of the problem are consistent with Philp and Liu conjecture and assert the existence tracking policy for a general switch size as claimed in [23].

Future works are in the directions of determining if Philp and Liu conjecture is true and if tracking policy for a general switch size exists. Particularly, the approach of devising better scheduling algorithms will be followed.

# Bibliography

- [1] J. Y. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*. Boston/Dordrech/London: Kluwer Academic Publishers, 1990.
- [2] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: Single node case," *IEEE/ACM Trans. on Networking*, vol. 1, pp. 344–357, June 1993.
- [3] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: Multiple node case," *IEEE/ACM Trans. on Networking*, vol. 2, pp. 137–150, Apr. 1994.
- [4] J. L. Rexford, A. G. Greenberg, and F. G. Bonomi, "Hardware-efficient fair queueing architectures for high-speed networks," *Proc. IEEE INFOCOM*, pp. 638–646, 1996.
- [5] J. C. R. Bennet and H. Zhang, " $WF^2Q$ : Worst-case Weighted Fair Queueing," *Proc. IEEE INFOCOM*, pp. 120–128, Mar. 1997.
- [6] J. C. R. Bennet and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Trans. on Networking*, pp. 675–689, Oct. 1997.
- [7] N. McKeown, "iSLIP: A scheduling algorithm for input-queued switches," *IEEE/ACM Trans. on Networking*, vol. 7, Apr. 1999.

- [8] N. McKeown and T. E. Anderson, "A quantitative comparison of scheduling algorithms for input-queued switches," *Computer Networks and ISDN Systems*, vol. 30, pp. 2309–2326, Dec. 1998.
- [9] P. Krishna, N. S. Patel, A. Charny, and R. J. Simcoe, "On the speedup required for work-conserving crossbar switches," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1057–1065, 1999.
- [10] M. A. Marsan, A. Bianco, and E. Leonardi, "RPA: A flexible scheduling algorithm for input buffered switches," *IEEE Trans. on Communications*, vol. 47, pp. 1921–1933, Dec. 1999.
- [11] N. McKeown and P. Varaiya, "Scheduling cells in an input-queued switch," *Electronics Letters*, pp. 2174–2177, Dec. 1994.
- [12] A. Mekkittikul and N. McKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," *Proc. IEEE INFOCOM*, vol. 2, pp. 792–799, 1998.
- [13] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *Proc. IEEE INFOCOM*, vol. 1, pp. 296–302, Mar. 1996.
- [14] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Trans. on Communications*, vol. 47, Aug. 1999.
- [15] R. Ahuja, B. Prabhakar, and N. McKeown, "Multicast scheduling for input-queued switches," *IEEE Journal on Selected Areas in Communications*, vol. 15, pp. 885–866, June 1997.
- [16] C. L. Liu and J. Wayland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *IEEE Trans. on Communications*, vol. 20, pp. 46–61, Jan. 1973.



- [17] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. on Communications*, pp. 1449-1455, Oct. 1979.
- [18] I. S. Gopal and C. K. Wong, "Minimizing the number of switchings in an SS/TDMA system," *IEEE Trans. on Communications*, vol. COM-33, pp. 497-501, June 1985.
- [19] A. Ganz and Y. Gao, "Efficient algorithm for SS/TDMA scheduling," *IEEE Trans. on Communications*, vol. 40, pp. 1367-1374, Aug. 1992.
- [20] I. R. Philp, *Scheduling Real-Time Messages in Packet-Switched Networks*. PhD thesis, University of Illinois at Urbana-Champaign, 1996.
- [21] I. R. Philp and J. W. S. Liu, "SS/TDMA scheduling of real-time periodic messages," *Telecommunications Systems Journal*, pp. 244-251, Mar. 1996.
- [22] J. Giles and B. Hajek, "Scheduling multirate periodic traffic in a packet switch," Master's thesis, Department of Electrical and Electronics Engineering University of Illinois at Urbana-Champaign, 1997.
- [23] V. Tabatabaee, L. Georgiadis, and L. Tassiulas, "QoS provisioning and tracking fluid policies in input queueing switches," *Proc. IEEE INFOCOM*, 2000.
- [24] D. C. Stephens and H. Zhang, "Implementing distributed packet fair queueing in a scalable switch architecture," *Proc. IEEE INFOCOM*, pp. 282-290, Mar. 1998.
- [25] D. Stiliadis and A. Varma, "Efficient fair queueing algorithms for packet-switched networks," *IEEE/ACM Trans. on Networking*, pp. 175-185, 1999.
- [26] N. McKeown, *Scheduling Algorithms for Input-Queued Switches*. PhD thesis, University of California at Berkeley, May 1995.

- [27] A. M. Ali and H. T. Nguyen, "A neural network implementation of an input access scheme in a high-speed packet switch," *Proc. of Globecom*, vol. 17, pp. 1040–1055, June 1989.
- [28] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1030–1039, June 1999.
- [29] H. F. Badran and H. T. Mouftah, "ATM switch architecture with input-output buffering: effect of input traffic correlation, contention resolution policies, buffer allocation strategies and delay in backpressure signal," *Computer Networks and ISDN Systems*, vol. 26, pp. 1187–1213, 1994.
- [30] R. E. Del and R. Fantacci, "Performance evaluation of input and output queueing techniques in ATM switching systems," *IEEE Trans. on Communications*, vol. 41, pp. 1165–1775, 1993.
- [31] J. S. Chen and T. E. Stern, "Throughput analysis, optimal buffer allocation, and traffic imbalance study of a generic nonblocking packet switch," *IEEE JSAC*, vol. 49, pp. 439–449, 1991.
- [32] H. Duan, J. W. Lockwood, and S. M. Kang, "Matrix cell scheduler (MUCS) for input-buffered ATM switches," *Journal of IEEE Communications Letters*, vol. 2, pp. 20–23, 1998.
- [33] A. C. Kam and K.-Y. Siu, "Linear-complexity algorithms for QoS support in input-queued switches with no speedup," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1040–1055, June 1999.
- [34] A. S. Tanenbaum, *Computer Networks*. New Jersey: Prentice-Hall International, Inc., 1996.
- [35] B. Kim, "Simulation of weighted round-robin queueing system," Master's thesis, Chungnan National University, 1991.

- [36] H. S. Hideyiku Schimonishi, "Performance of weighted round robin cell scheduling and its improvement in ATM switches," *Trans. IEICE*, vol. EB1-B, pp. 919–929, May 1998.
- [37] V. N. Sachkov, *Combinatorial Methods in Discrete Mathematics*. Cambridge University Press, 1996.
- [38] B. Li and Y. Qin, "Traffic scheduling with per VC QOS guarantee in WDM networks," *Proc. IEEE INFOCOM*, pp. 339–344, 1998.