

POLITECNICO DI TORINO

Corso di Dottorato in Ingegneria Elettronica e delle Comunicazioni

Tesi di Dottorato

**Queueing and scheduling algorithms
for high performance routers.**

Tutore:
Prof. Marco Ajmone Marsan

Coordinatore:
Prof. Ivo Montrosset

Paolo Giaccone

Febbraio 2002

Acknowledgments

My Ph.D. studies have been an excellent experience.

I have enjoyed my research activity with Prof. Marco Ajmone Marsan, Prof. Fabio Neri, Prof. Andrea Bianco, and especially with Prof. Emilio Leonardi, Prof. Balaji Prabhakar and Devavrat Shah. I wish now to thank all of them for their help, in particular Emilio and Devavrat for their extraordinary patience to guide my research and their support to overcome all the technical problems I met. I thank also Balaji for the opportunity he gave me to visit his research group at Stanford University; I like to mention his gentle attitude and his enthusiastic supervision of my work. My advisor Marco superintended my research and was a scrupulous source of comments about my works. Andrea and Fabio have been crucial to develop some ideas and to provide a careful feedback on my work.

I must acknowledge the generosity of Dott. Ing. Salvatore Bonadonna who offered me an important financial support to visit Stanford for more than one year.

I thank also my parents and Serena, for providing their moral support to my research work during these years. I dedicate this dissertation to them.

Torino, January 2002

Table of contents

Acknowledgments	1
1 Introduction	1
1.1 Towards high speed networks	1
1.2 Role of routers in Internet	2
1.3 Scheduling in high performance routers	3
2 The design of high performance routers	5
2.1 Design issues of Internet routers	5
2.1.1 Backbone routers	6
2.1.2 Enterprise routers	7
2.1.3 Access routers	7
2.2 Router architecture	7
2.2.1 Switching architectures	8
2.2.2 Input queued cell-switches	10
2.2.3 Output queued cell-switches	11
2.2.4 Combined input-output queued cell-switches	11
2.2.5 Integration of the cell switch inside a router	13
3 Scheduling in input queued switches	15
3.1 Problem definition of scheduling	15
3.2 Scheduling algorithms	17
3.3 Simulation study	21
3.3.1 Software of simulation	21
3.3.2 Traffic scenarios	21
3.3.3 Performance indexes	22
3.4 Theoretical study	23
3.4.1 Referred notation	23
3.4.2 Stability of controlled queueing systems	24
3.4.3 Optimal policy for unicast traffic	26

TABLE OF CONTENTS

4	Learning scheduling algorithms for unicast traffic	27
4.1	The learning approach	28
4.1.1	Maximization of the matching	31
4.2	Stability of the learning approach	32
4.2.1	Some preliminary results	32
4.2.2	Stability of <i>epoch-learning</i> scheduling algorithms	33
4.2.3	Stability of a <i>learning</i> scheduling algorithm	36
4.3	Architecture of the <i>weight-boosters</i>	38
4.3.1	MERGE procedure	38
4.3.2	ARR-MERGE procedure	40
4.4	Computation of the <i>probe-matching</i>	42
4.4.1	Exploiting the randomization	42
4.4.2	Exploiting the Hamiltonian walk	45
4.4.3	Exploiting the arrivals	46
4.4.4	Exploiting the neighbors	46
4.5	Schedulers with parallel learning schemes	47
4.6	LAURA approach	48
4.7	LAURA performance	50
4.7.1	Stationary analysis of LAURA	50
4.7.2	Transient analysis of LAURA	55
4.7.3	Simplified version of LAURA	58
4.7.4	Derandomized version of LAURA	58
4.8	APSARA approach	60
4.8.1	APSARA-B: basic version	60
4.8.2	APSARA-L: linear version	61
4.8.3	APSARA-R: randomized version	61
4.8.4	APSARA properties	61
4.8.5	Implementation	62
4.8.6	Comments about APSARA stability	63
4.9	APSARA performance	63
4.10	SERENA approach	69
4.10.1	SERENA-B: basic version	69
4.10.2	SERENA-S: simplified version	70
4.11	SERENA performance	70
4.12	Comparison among different learning schemes	75
5	Scheduling multicast traffic	79
5.1	Multicast traffic	81
5.2	Scheduling disciplines	82
5.3	Optimal scheduling	84
5.3.1	<i>MC-VOQ</i> queueing system	84

TABLE OF CONTENTS

5.3.2	Switch description	85
5.3.3	Admissible region	87
5.3.4	Throughput definition	88
5.3.5	Optimal scheduling and capacity region	88
5.4	<i>K</i> -complex traffic definition	89
5.5	Traffic sustainability conditions: preliminary results	92
5.6	Traffic sustainability conditions: main results	93
5.7	Greedy matching algorithm	95
5.7.1	Multicast scheduling	96
5.8	Simulation study	97
5.8.1	Traffic patterns for simulation experiments	97
5.8.2	Simulation results	98
6	Scheduling variable-size packets	101
6.1	Input queued packet switches	101
6.1.1	Packet-mode scheduling algorithms	102
6.1.2	Stability of packet-mode scheduling	103
6.1.3	Delay of packet-mode scheduling	109
6.2	Output queued packet switches	110
6.3	Simulation results	111
6.3.1	Traffic scenarios	111
6.3.2	Performance indices	113
6.3.3	Uniform packet scenario	113
6.3.4	Spotted packet scenario	114
6.3.5	Diagonal packet scenario	116
6.3.6	Packet-mode gains	117
6.4	Combined input-output queued packet switches	119
6.4.1	Performance of combined input-output queued packet switches	119
7	Conclusions	125
	Bibliography	127
A	Proofs about scheduling multicast traffic	137
A.1	Optimality of the max-scalar policy for multicast traffic	137
A.2	Multicast departure vectors	139
A.3	On the minimum frame lengths for complete multicast schedules . . .	140
B	Theoretical performance of an output queued switch	147

Chapter 1

Introduction

1.1 Towards high speed networks

In the recent years, Internet has become the standard network infrastructure to allow a global communication in the world. Now Internet is not only world-wide, but it is also “content-wide” since it supports a very large number and variety of communication services.

Born as a research and university network, providing basic services like e-mail and file transfer, Internet has been growing at an exponential rate. Important research and commercial efforts have supported this expansion. Furthermore, a “new-economy” has arisen to exploit the possibilities of this new mean of communication, able to reach a lot of people in the world at low costs. Lately, Internet has also changed the communication habits of millions of people, nowadays familiar with sending e-mails and with browsing the World Wide Web (WWW). During the last couple of years, Internet has been proposed also as a feasible alternative to the Public Switching Telephony System (PSTN), through transport services like “Voice Over IP”.

The American economical crisis during 2001 has shown also the limit of Internet as mean of business. Internet was initially designed as a low cost and global communication mean, but its intrinsic features seem now to limit its scalability in terms of dimension, services and business. For example, Internet is a “best-effort” network; this implies that service providers cannot ask users to pay a significant fee, given that the quality of service over time is not guaranteed. On the contrary, advanced communication systems, like video and audio distribution, require a minimum guaranteed level of service quality to be practical. In general, whenever the network can accommodate the communication requirements of “Quality-of-Service” (QoS), Internet will improve its business potentials.

Starting from its birth, several technical considerations rose doubts that Internet would have collapsed after few years, due to the very large number of users and the

constantly growing traffic. Internet is a packet switched network based on the “statistical multiplexing” paradigm, which means that the resources are shared among users and not dedicated. Hence, Internet could collapse if the offered traffic would become greater than the traffic that the available resources are able to serve. Since no “admission control” mechanism is adopted to limit user traffic, this risk is real. Several solutions have been envisaged to deal with this foreseen collapse. One solution is a more thoughtful design of the network architecture, maybe exploiting some fair resource sharing and admission control systems. Another solution is the continuous increase of communication links and switching systems, to increase the overall bandwidth available in the network.

The main topic of my Ph.D. work has been the design of high performance switching systems, able to accommodate the exponential traffic growth in Internet by guaranteeing high throughput.

1.2 Role of routers in Internet

Internet is a very large packet switched network, built around a large variety of transmission and switching systems. The information to transfer through the network is aggregated into packets; each packet is individually forwarded and switched towards its destination through the packet switching systems.

The most important switching systems in Internet are the routers. The main tasks of a router are to receive the packets from input ports, to find their local destination port on the basis of the routing table (built by exchanging information related to network topology through routing protocols) and finally to transfer the packets to output ports. These two basic functions, *routing* and *switching*, are very difficult to implement when the aggregate bandwidth is very large, since complex algorithms should run in a very short time. Furthermore, routers must deal with different link technologies, must distribute routing tables and can provide packet classification and filtering services. For all these reasons, high performance routers are very complex systems, whose design is more and more pushed to the edge of the latest technology.

Being able to design high performance routers is particularly important, given that today’s Internet is composed by a relative small number of very fast *backbone networks*, which connect a very large number of smaller networks. Backbone links rates are evolving from today’s OC-48 (2.5 Gbps) to OC-192 (10 Gbps) and even OC-768 (40 Gbps) [16], with a rate of increase of about 30% per year [16]. This means that, for a minimum size TCP/IP packet of 40 bytes, the number of packets to be processed is evolving from 8 to 32 and even 125 million of packets/s. So the average time spent to elaborate a single packet (doing at least routing and switching) is decreasing from 125 ns to 30 ns and even 8 ns: hence, the switching process at high speed must be implemented in hardware.

1.3 Scheduling in high performance routers

This Ph.D. dissertation is on the design of high performance routers with a very large aggregate bandwidth. Different switching architectures can be employed in high performance routers. This studies focuses mainly on one of the most promising switching architecture, with packet buffers at input ports; this architecture will be introduced in Section 2.2.2 and discussed deeply in Chapter 3.

For very fast switching architectures with buffers at input ports, the main performance bottleneck is given by the *scheduler* module, which selects the packets to transfer through the switching fabric. In this dissertation it will be discussed mainly the problem of designing very efficient scheduling algorithms, which show very good performance and can be implemented in hardware. The scheduler architecture will be discussed only from the algorithmic point of view, independent from any particular electronic technology to implement it.

The main topics and results discussed in this dissertation are the following:

- design of implementable schedulers for unicast traffic: we will propose a large class of algorithms able to achieve very good performance with a low implementation complexity;
- design of schedulers for multicast traffic: we will define the optimal scheduler and show some intrinsic performance limitations for switches with input buffers;
- design of schedulers for variable-size packets: we will show how to integrate efficiently the scheduling function with the processing of variable size packets, like IP packets.

This Ph.D. dissertation is organized into chapters. Chapter 2 discusses the main design issues involved in designing high performance IP routers and motivates the need for high performance switching fabrics. It introduces the main features of different categories of routers and also highlights some of the most significant bottlenecks in their performance. In addition, it provides a taxonomy of the switching fabrics.

Chapter 3 describes deeply the problem of scheduling. It is a sort of preface to all the subsequent chapters, since it describes the methodological approaches (based mainly on the performance analysis through simulations and stochastic modeling) to study the scheduling problem. The final part of that chapter recalls some well-known results of stochastic modeling, essential to understand the theoretical modeling approach used in the other chapters.

Chapters 4, 5 and 6 discuss respectively each of the three main topics of our research work, presented earlier in this section. Each chapter motivates the outlined

approach, comparing it with results known in literature. The approach is then described and its performance are discussed first by theoretical models and then with simulation studies.

After the conclusions and the bibliography, an appendix reports some of the proofs which were skipped in the main context of this dissertation for an easier reading.

Chapter 2

The design of high performance routers

As we saw in the Introduction, one strategy to deal with the exponential growth of the traffic in Internet is to design router with higher and higher switching capacity. In this chapter, we will describe the main issues involved in designing high performance routers, taking into account their position inside the network. We will also explain the model to abstract the switching process of a router. The last part of this chapter will be devoted to a taxonomy of the architectures for packet switching.

2.1 Design issues of Internet routers

Internet is a very large network, world-wide. It is very difficult to give a precise description of its architecture, but some flavor of its architecture can be given by exploiting the fact that today's Internet is quite hierarchically structured.

Internet is composed by hundreds of *backbone networks*, which connect some ten of thousands of smaller networks, called *enterprise networks*. Enterprise networks are composed by smaller networks, where the single hosts (or users) are connected. Large service providers can have an *access network* to collect the traffic from the users and to send their traffic to the network.

Routers transfer the packets through all these networks, hence can be classified mainly in three categories: *backbone routers*, *enterprise routers* and *access routers*. The reader should refer to [50] to understand the main issues and trends in the design of all these kinds of routers.

We now discuss some issues of those three categories of routers. This division in categories is loose, since some routers can share some design issues of different router categories.

2.1.1 Backbone routers

As we saw in the introduction, backbone routers switch the traffic among enterprise networks, allowing the communication among endpoints located in different enterprise networks. In the backbone the traffic is given by aggregating low speed channels. Hence, backbone routers are built to connect few links at very high speed, like OC-48 (2.5 Gbps) up to OC-768 (40 Gbps).

The main issues in their design is their reliability and their speed. Reliability is achieved as in the telephone switches, by using hot-spares, redundant power supplies and duplicate data-paths.

Speed in these routers is usually limited by these factors:

- *Routing operations.* When a packet arrives, the router reads its header and chooses the destination on the basis of the routing information obtained by *routing tables*. Routing tables are very large and a packet can match multiple destinations: the packet is forwarded following some rules, like the *longest prefix matching*. The main approach to deal with fast routing is to use fast lookup caching, usually implemented in hardware. The efficiency of caching depends on the correlation of the traffic traveling the network. Another approach uses protocols like *IP-switching* and *tag-switching*, which are able to reduce the dimension of the routing tables by allowing wider coordination at network level.
- *Memory bandwidth.* Packets are stored in electronic buffers (optical buffers cannot be adopted since their technologies have not been well established so far) to be processed and to solve contentions when multiple packets try to access to the same switching resource. Electronic memories are not able to deal with the faster and faster optical communication links. Static RAM (SRAM) memories offer faster access times than Dynamic RAM (DRAM) memories, but their density is lower. Whereas the density of memories is always growing and their price is dropping by 60 percent [50] every year, their access time is almost the same during the last years, with about 1 ns for SRAM and 10 ns for DRAM. The main approaches to deal with insufficient memory bandwidth is to use memories in parallel [45] and to combine the use of SRAM and DRAM [46].
- *Switching architecture.* As we will discuss later in Section 2.2.1, the queuing architecture and the scheduling algorithms affect the throughput of the overall system. If not well designed, the throughput of a switching architecture can be very low (also less than 60%).

An additional issue in the design of backbone routers is the stability of routing tables when backbone routers are connecting smaller networks running different routing protocols. This is not yet a well understood problem, according to [50].

2.1.2 Enterprise routers

Enterprise routers connect endpoints or segments of endpoints (like Ethernet segments), allowing to partition the network in several broadcast domains. They usually have a high number of ports, with a lower speed than backbone routers. Their main design issues are related to the packet classification and the packet filtering; the classification is aimed to satisfy the QoS requirements of the packets, whereas the filtering is usually built to provide native security features in the router. Lookup tables related to routing and classification can support an update rate on the order of hundreds of updates per second [16]. At the same time, enterprise routers support multiple protocols and tunneling features. In enterprise networks, also the available administrative tools (in software) are very important, since they allow to configure the numerous features involved.

2.1.3 Access routers

Access routers connect customers at home or at work with an Internet Service Provider (ISP). In the past, non-intelligent concentrators were collecting the communications from pools of modems connected to the Public Switching Telephony Network (PSTN). Now, a large variety of access technologies to Internet are employed (like high-speed modems, ADSL modems or cable modems) and transmission speeds are growing. Furthermore, voice and data are transmitted on the same physical channel, and some access routers are now able to bypass the voice signal. Hence, all these service requirements can be met only through specialized routers. The main design issue of access routers is to provide the connection among a very large number of ports, at different speed, and running heterogeneous access technologies and protocols.

2.2 Router architecture

Although different design approaches characterize all the cited router categories, the main components of a generic router are, according to [50, 72, 95]:

- **Network/routing processor.** It manages the overall routing process. It computes the forwarding tables, implements the routing protocols and runs the software to configure and manage the router.
- **Input/output ports.** They receive and transmit packets through the communication channels. They provide data-link encapsulation/decapsulation and can classify packets on the basis of some rules specified by the routing processor. In some cases, a *port processor* and a *processing engine* are present in each port

and they are able to perform active processing of incoming packets: they provide flow classification, assign a tag to each packet, segment incoming packets into fixed-size cells and reassemble cells to constitute the outgoing packets.

- **Forwarding engines.** They are optional modules, designed to find quickly the destination port of a packet, on the basis of the packet destination address and of some classification rules. Forwarding engines provide fast lookup, using high speed caching systems. To improve the throughput, multiple engines can run independently and concurrently. For example, the Cisco 12000 Series backbone routers [23] can use up to 15 forwarding engines and the Juniper M160 backbone router up to 4.
- **Switch.** It transfers packets from one port to another, on the basis of informations derived by the forwarding engines. Section 2.2.1 reports an overview of the main switching architectures.

Note that all these components can be combined into one or multiple cards, and their role can be different depending on the actual router architecture. Furthermore, the communication among all the described modules is supported either by some specialized hardware architecture or by the same switch used to transfer the data units.

2.2.1 Switching architectures

Several architectures have been proposed for packet switches, some of them inspired by the telephony world. For a good survey of these architecture, the reader can refer to two books [42, 79] and to some tutorial papers [13]. All the architectures for packet switching are constituted by a switch fabric and some buffers. The switch fabric forward the packets at the inputs to their destination ports. The most common architectures of switch fabrics are the following:

- *crossbar* : this architecture is inspired on the old electro-mechanic telephony crossbars, with N^2 contact points, if N is the number of ports. The crossbar is a strictly non-blocking switching fabric, since whenever a free input port should be connected to a free output port, they can be connected always (property of “non-blocking”) and without reconfiguring all the other connections (property of “strictly non-blocking”). Although the crossbar does not scale very well for a large number of ports, it is considered as the basic switching fabric for high speed routers. A crossbar is able to transfer in parallel up to N packets coming from different inputs and destined to different outputs; the set of the packets that are transferred according to the crossbar constraints is called a *non-conflicting set*. When the rate of transfer in a crossbar is comparable with the aggregate

arrival rate (this case will be referred later with the concept of *speedup one*), the performance bottleneck is mainly the control which coordinates the transfer of packets, especially when packets coming from different inputs are destined to the same output. In this case, a contention for the common destination arises, and it is solved by the scheduler: only one of the contending packets is transferred to the desired output, whereas all the other contending packets are stored in some queues.

- *multistage switch* : this architecture is non-blocking like a crossbar, but it scales well with the dimensions of the switch. Although, from the scalability point of view, a multistage fabric is better than a crossbar, it requires some internal coordination to transfer packets from one stage to another. From a model point of view, a strictly non-blocking and bufferless multistage switch (like a Clos network or a Batcher-Banyan network [42]) is functionally the same as a crossbar, since it is able to transfer in parallel a non-conflicting set of packets. Hence, all the results obtained in this Ph.D. dissertation hold also for a multistage switching fabric, which is bufferless and strictly non-blocking.
- *bus* : this architecture is strictly non-blocking, but it allows at most one packet to be transferred at the same time, hence it requires a coordination among the ports. The performance of a bus are mainly limited by the bus capacitance and the arbitration process.

From now on, we consider only a non-blocking and bufferless switch fabric, like a crossbar. The switching fabric is integrated with a buffering scheme to store packets contending for the same switching resource (for example, an output port or the ingress of the switching fabric). We can classify the switching architectures on the position of the buffers with respect to the input or output ports. The main buffering schemes are input-queued (IQ), output-queued (OQ) or combined input-output-queued (CIOQ). In the next Sections 2.2.2, 2.2.3 and 2.2.4 we will discuss these buffering schemes and their integration with the switch fabric.

Note that buffers can be shared among different ports, at input or output level; in this case *shared memory fabrics* are referred [13]. Usually, the main limit of the buffer performance is the memory bandwidth, as discussed previously in Section 2.1.1. Here we are not considering these kinds of architectures.

When the switching architecture is implemented in hardware at very high speed, it usually works with data-unit of fixed size, called *cells*. For *packet* we mean a data-unit of variable size, like an IP-datagram or an TCP/IP packet. For a clearer exposition, the basic switching architectures will be referred as *cell-switches*, whereas the overall switching system of a router will be referred as *packet-switch*. Section 2.2.5 will show how to build the packet-switch around a basic cell-switch. Note also that Chapter 6

will propose an optimized architecture to integrate cell-switching inside a packet-switching system.

2.2.2 Input queued cell-switches

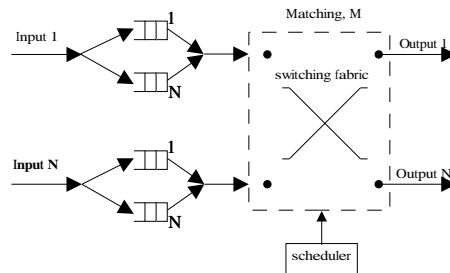


Figure 2.1. Logical structure of an input-queued cell switch

Fig. 2.1 shows the logical structure for an input-queued (IQ) **cell** switch. The switch operates on fixed-size data units, which can be ATM cells, or have any other convenient format. Borrowing from the ATM jargon, we shall use the term *cells* to identify the internal fixed-size data units. Actually, our results do not strictly require that fixed-size cells are used, but more generally refer to any switch that takes switching decision at equally-spaced time instants. The distance between two switching decisions is called *slot*, and the slot is the granularity in allocating switch resources.

We do not deal with the problem of partial slot filling due to the variable size of IP packets arriving at inputs, even if its impact on performance may be significant, depending on the slot size and the input packet length distribution.

We consider a switch with N inputs and N outputs (indeed, one input and one output interface usually reside on the same “line card”). We also assume for simplicity that all input and output lines run at the same speed.

Packets are stored at input interfaces. Each input manages one queue for each output, hence a total of $N \times N = N^2$ queues are present. Each queue can store up to L cells and excess packets are dropped. This queue separation permits to avoid performance degradations due to head-of-the-line blocking [49], and is called Virtual Output Queuing (VOQ) or Destination Queuing [10, 61, 89, 88].

The switch fabric is non-blocking and memoryless, and introduces no delay: at most one cell can be removed from each input and at most one cell can be transferred to each output in every slot. Since the speed at which cells are fed into output interfaces is equal to the speed at which cells are fed to input interfaces, we have

a speedup equal to 1. The scheduling algorithm decides which cells are transferred from the inputs to the outputs of the switch in every slot. Scheduling algorithms will be discussed in details in Chapter 3.

2.2.3 Output queued cell-switches

Output queued (OQ) cell-switch needs no input buffers (neglecting buffers used at the inputs to store the newly arrived cell) because the switching fabric has enough capacity to transfer to the desired output all the cells received in one time slot. In the worst case (i.e., when a cell arrives at each input, and all cells are directed to the same output), this means that the bandwidth towards each output must be equal to the sum of the bandwidths available on all input lines. We thus say that the switching fabric must have a speedup equal to N . This internal speedup can be obtained in different manners. We assume here that it is achieved in the time domain, by setting the switching fabric clock N times faster than on input/output line interfaces. We therefore assume that the slot time is subdivided into N mini-slots, and that each input can use a different mini-slot to transfer a cell to a given output. The allocation of mini-slots to inputs can be fixed, time-varying, or random. We implemented a fixed round-robin of inputs in our simulation programs.

At each output, cells are stored in a single FIFO queue. For a fair comparison with input-queued switches, we assume that the total amount of buffer space is kept constant, i.e., that the FIFO output queue can store $N \times L$ cells. This assumption gives some advantage to the OQ schemes, which can exploit some degree of buffer sharing.

2.2.4 Combined input-output queued cell-switches

Combined input-output queued (CIOQ) architectures are a compromise between IQ and OQ, for which some degree of speedup is available, but not as much as for OQ. These architectures require buffering at both input and output line interfaces.

The buffer space at each input can be organized in either one FIFO queue, or several queues, like in the case of VOQ. We call S_{in} the number of cells per slot that can be read from the queue(s) at each input, and S_{out} the number of cells per slot that can be written into the output queue at each output.

Several CIOQ architectures were proposed in the literature. Unfortunately, different authors often assume different definitions of speedup. The following cases are possible.

- $S_{in} = 1$ and $S_{out} = S$ [18, 36, 44, 75, 76]. The switch can transfer up to S cells to the same output, but no more than a cell can be read from each input in one slot. This definition $S^{(1)}$ of speedup derives from *input bus* switching

architectures, in which a shared bus can be read by all output ports, but can be accessed by each input for the transmission of at most one cell per slot. A well-known example of this architecture is the *Knockout switch* [97].

Analytical models [13] show that the maximum throughput for $S^{(1)} = 2$ equals 0.885 in uniform traffic conditions when $N \rightarrow \infty$. To achieve a throughput larger than 99%, $S^{(1)}$ must be larger than 4. Only with $S^{(1)} = N$ it is possible to reach the maximum throughput in general traffic conditions.

- $S_{in} = S$ and $S_{out} = S$ [76, 22]. When this speedup $S^{(2)}$ equals S , up to S cells can be read from each input and written to each output of the switch in one time slot.
- $S_{in} = S$ and $S_{out} = 1$. This definition $S^{(3)}$ of speedup was investigated in [30]. It is dual with respect to $S^{(1)}$, in the sense that at most one cell per slot can reach an output card. When coupled with suitable scheduling algorithms for variable-size packet switching, this form of speedup has the advantage of preventing the interleaving of cells belonging to different packets.

Of course, when $S^{(1)} = 1$, or $S^{(2)} = 1$, or $S^{(3)} = 1$, we have an IQ architecture. We define the speedup according to $S^{(2)}$ in the remainder of the dissertation. A CIOQ architecture with VOQ and speedup $S^{(2)} = S$ is shown in Fig. 2.2.

It is well-known that the maximum normalized throughput achievable in an IQ switch using one FIFO queue per input port, with uniform traffic and an infinite number of ports, is limited to 0.586 by the head-of-the-line blocking phenomenon [49]. One can argue that, by executing the scheduling algorithm twice in each slot, thereby using a speedup $S^{(2)} = 2$, the maximum throughput becomes 100% [76], as it is for OQ architectures. This simple statement holds only for uniform traffic, and does not provide guarantees related to delays.

It has been shown [22] that a CIOQ architecture with VOQ and speedup $S^{(2)} = 2$ can mimic exactly the OQ behavior, under general traffic patterns, using a quite complex scheduling algorithm; the same architecture can achieve the work-conserving property of an OQ switch through a simpler scheduler [55].

In this dissertation we shall compare IQ and OQ architectures (in the final chapter, also CIOQ). With the aim of being fair in the comparison, we accept high scheduling complexity for IQ, high hardware complexity (i.e., speedup N) for OQ, and intermediate scheduling and hardware complexity for CIOQ. We therefore limit our attention to CIOQ switches with limited speedup increase $S^{(2)} = 2$, and very simple FIFO scheduling.

Although a more detailed discussion of scheduling algorithms for IQ architectures will be provided in Section 3.2, we anticipate here the description of the simple algorithm that we consider in the CIOQ case. We assume that each input and each output

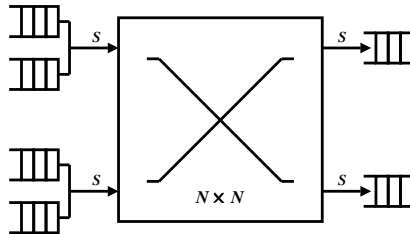


Figure 2.2. CIOQ architecture with VOQ and speedup $S^{(2)} = S$

of the CIOQ switch maintains a single FIFO queue, and that the following scheduling algorithm is executed twice in each slot.

At every execution of the algorithm, inputs are cyclically scanned, starting from a different input every time, selected by a round-robin scan. In this scan each input attempts to transfer the cell at the head of its input FIFO queue. If the corresponding output was not already engaged by a preceding input in the round-robin scan, the transfer is enabled, otherwise it is deferred to the next execution of the algorithm. Since the algorithm is executed twice in each slot time, at most the first two cells are removed from input queues, and at most two cells are delivered to each output queue in each slot. We call this scheduling algorithm FIFO-2.

2.2.5 Integration of the cell switch inside a router

Given the cell-switch logical architectures described in previous sections, we can now build a packet switch around them. We use as a reference model the case of a high-performance IP router built around an ATM cell-switch, but the same analysis can be referred also to a router built around a generic (possibly proprietary) cell-based switch.

Each router port has line interfaces where any data-link and physical layer protocol can be used to receive and transmit IP datagrams. The IP protocol sits on top of the data-link protocol at the input and at the output of the router. Within the IP layer at the input, routing functions are activated to associate an output port with the destination IP address (we neglect here issues related to the implementation of these functions, such as table look-up). Input IP datagrams are segmented into ATM cells, that will be transferred to output ports by a high-performing ATM switching fabric.

Once cells are delivered to an output port, they are reassembled into the IP datagram, which is transmitted on the output line according to possibly different line formats.

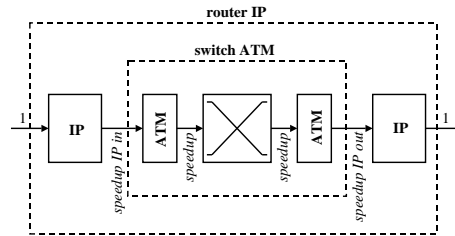


Figure 2.3. Speedup definitions for the packet switch built around a cell switch

Fig. 2.3 emphasizes the possible speed variations inside the architecture of an IP router incorporating an ATM switching fabric. We take as a reference the bit rates on the input and output lines, which we assume to be the same and equal to 1.

- SPEEDUP-IP-IN is the speed at which cells are transferred from the input IP module to the input of the ATM switch. Unitary SPEEDUP-IP-IN means that, if an input IP datagram is segmented in k cells, all these cells are sequentially transferred to the cell-switch input in k time slots. Note that this takes into account segmentation overheads and partial filling of the last cell.
- SPEEDUP is the number of cells per slot that can be read from the inputs of the cell-switch. Since we assume the definition $S^{(2)}$ of speedup for the cell-switch, this is also the number of cells per slot that can be written onto a switch output.
- SPEEDUP-IP-OUT is the speed at which cells are transferred from the output of the cell-switch to the output IP module.

Input and output IP modules in Fig. 2.3 consist of queues for variable-size packets, and operate in store-and-forward mode; they comprise segmentation and reassembly functions. ATM modules comprise cell queues at the input and/or at the output of the switch.

Chapter 3

Scheduling in input queued switches

In this chapter we discuss the scheduling problem in IQ cell-switches. In the first two sections, we define formally the scheduling problem and describe some known solutions proposed in literature. Section 3.3 examines the simulation methodology adopted in the following chapters. Section 3.4 presents the theoretical foundation of the analytical approach adopted and introduces the main mathematical notations referred in this dissertation.

3.1 Problem definition of scheduling

This section describes scheduling algorithms, which avoid blocking and solve contention within the switching fabric. We assume a VOQ buffering scheme. The queue storing the packets from input i to output j is called VOQ_{ij} and its occupation is X_{ij} .

The scheduling algorithm selects a *matching* M , i.e., a set of input-output pairs with no conflicts, such that each input is connected with at most one output, and each output is connected with at most one input. In each slot, if input i is connected with output j , a cell is removed from VOQ_{ij} , and transferred to output j by properly configuring the non-blocking switching fabric.

In the technical literature, scheduling problems in IQ switches are modelled as matching problems in bipartite graphs. The switch state can be described as a bipartite graph $G = [V, E]$ (see Fig. 3.1) in which the N left-most nodes correspond to the N input interfaces of the switching fabric, whereas the N right-most nodes correspond to the N output interfaces. Edges indicate the needs for cell transfers from an input to an output (an edge from left-node i to right-node j indicates the need for cell transfers from input i to output j), and can be labeled with a weight (denoted by w_{ij}), whose intuitive meaning is the “urgency” of serving queue VOQ_{ij} . The adopted metrics to assign weights to edges is a key part of the scheduling algorithm, as we shall see. The weight can be binary to simply indicate that at least one cell to be transferred exists,

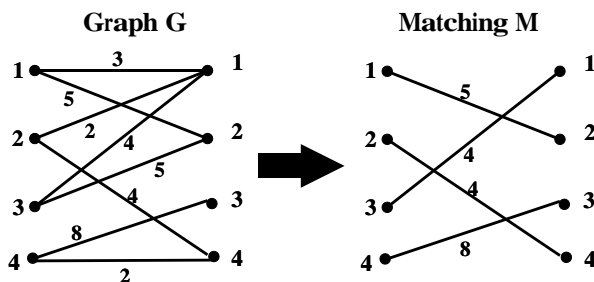


Figure 3.1. Bipartite graph description of the scheduling problem

or it can refer to the number of cells to be transferred, to the time waited by the oldest cell, or to other state indicators.

A matching M is a subset of edges of G such that no vertex has two connected edges and represents an admissible switching configuration if a cell is transferred from input i to output j when the edge from left-node i to right-node j is present: indeed, it can never happen that two cells are extracted from the same input, or that two cells are transferred to the same output. The *weight* of a matching is the sum of the metrics corresponding to the edges included in the matching. A matching has *maximum size* if the number of edges is maximized; a matching has *maximum weight* if its weight is maximized. A matching is *heavier* than another matching if its weight is greater than the other one. The complexity is $O(N^3)$ for the maximum weight matching (MWM) [90, Chapt. 8] algorithm, that can be proved to yield the maximum achievable throughput using as metrics either the number of cells to be transferred, or the time waited by the oldest cell; it is $O(N^{5/2})$ for the simpler and less efficient maximum size matching algorithm [28]. Well known maximum size matching algorithms were proposed by Dinic [90] and Hopcroft [40].

The $N \times N$ matrix whose elements are the edge metrics in graph $G = [V, E]$ is called the *weight matrix*, denoted by $\mathbf{W} = [w_{ij}]$. This weight matrix \mathbf{W} varies with time, according to the changes in the system parameters from which its elements are computed. We assume $w_{ij} = 0$ for missing edges in G , i.e., when no cells from input i to output j are waiting in input queues.

We call A_{ij} the arrival process at input i for output j ; the average arrival rate is denoted by λ_{ij} . The aggregation of all arrival processes is $A = \{A_i, 1 \leq i \leq N\}$.

Definition 1. The arrival process A is termed *admissible* if no input and no output is

overloaded, i.e.:

$$\sum_{i=1}^N \lambda_{ij} < 1, \quad 1 \leq j \leq N$$

$$\sum_{j=1}^N \lambda_{ij} < 1, \quad 1 \leq i \leq N$$

Otherwise A is *inadmissible*.

For later use, we define the average *cell arrival rate matrix* $\mathbf{A} = [\lambda_{ij}]$, and the *normalized cell arrival rate matrix* $\mathbf{\Gamma} = [\gamma_{ij}]$, with $\gamma_{ij} = \lambda_{ij} / (\sum_{i=1}^N \sum_{j=1}^N \lambda_{ij})$. The *maximum input load* is $\rho = \max_i \{\sum_j \lambda_{ij}\}$.

3.2 Scheduling algorithms

There are two main quantities for measuring the performances of a switch scheduling algorithm: throughput and delay. Past theoretical works on packet switches have been concerned with studying algorithms that achieve 100% throughput. Such algorithms are referred to as “stable” algorithms. In particular, the papers [62, 92], showed that under Bernoulli i.i.d. packet arrival processes the MWM is stable so long as the arrival process is admissible¹. More recently, other algorithms have been proposed for providing exact delay bounds [22, 55, 82]. These algorithms in fact provide something much stronger: they allow a switch whose fabric runs at a speedup of between 2 and 4 to exactly emulate an output-queued switch. Thus, they are stable and permit the use of sophisticated algorithms for supporting quality-of-service (QoS).

But, all of the above algorithms are too complicated for implementation in high aggregate bandwidth switches. They require too many iterations (for example, the MWM requires $O(N^3)$ iterations in the worst-case), and the computation of weights used in the algorithms of [22, 55, 82] requires too much information to be communicated between inputs and outputs.

Implementation considerations have therefore seen the proposal of a number of practicable scheduling algorithms; notably, iSLIP [60], iLQF [61, 66], RPA [2], MUCS [27] and WFA [88]. However, these algorithms perform poorly compared to MWM when the input traffic is non-uniform: they induce very large delays and their throughput can be less than 100%.

More recently, some particularly simple-to-implement scheduling algorithms have been proposed [17, 45] and proven to be stable. But, [17] introduces an extra

¹The weights were taken to be the length of VOQ_{ij} originally and later work [68] took the weights to be the age of the oldest packet in VOQ_{ij} .

packet resequencing problem and [45] needs multiple switching fabrics. Nevertheless, these algorithms make a significant point: Delivering 100% throughput does not complicate the scheduling problem.

On the other hand, in order to keep delays small, it seems necessary to find good matchings; and finding good matchings takes many iterations and consumes time. But, high aggregate bandwidth switches do not leave much time for scheduling, because they are either connected to very high speed lines or they have too many ports.

A number of scheduling algorithms for IQ switch architectures have appeared in the technical literature. In this dissertation we consider only iSLIP (for its simplicity), iLQF and RPA (for being an approximation of the MWM with the weight equal to the queue length, with good performance under unbalanced traffic) and, finally, MUCS (for being an approximation of the MWM with the weight related non-linearly to the queue length, with very good performance under unbalanced traffic). The reader is referred to the original works for a detailed description of these algorithms. In this section we recall the general taxonomy for scheduling algorithms proposed in [7], and classify accordingly the considered proposals.

The output of the scheduling algorithm at each time slot k is a cell transfer matrix $\mathbf{D} = [d_{ij}]$, whose elements provide the result of the matching computation:

$$d_{ij} = \begin{cases} 1 & \text{if a cell is transferred from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases} \quad (3.1)$$

Any IQ scheduling algorithm can be viewed as operating according to three phases:

1. *Metrics computation.* Computation of the weight matrix \mathbf{W} . Each one of the possible N^2 edges in the bipartite graph is associated with a metrics depending on the current state of the corresponding queue. This metrics will act as a priority for the cell transfer.
2. *Heuristic matching.* Computation of the cell transfer matrix \mathbf{D} . Since it is know [65] that the Maximum Weight Matching is the optimal policy for unicast traffic (the precise statement is reported in Section 3.4.3). Hence this phase must try to maximize the following sum, in order to mimic the MWM:

$$\sum_{i=1}^N \sum_{j=1}^N d_{ij} w_{ij} \quad (3.2)$$

with the constraints:

$$\sum_{i=1}^N d_{ij} \leq 1 \quad \sum_{j=1}^N d_{ij} \leq 1 \quad (3.3)$$

Since the cost for the computation of the optimum matching (maximum size or maximum weight) is too high, all scheduling algorithms resort to heuristics with variable effectiveness and complexity. When the matching is not optimum, but no cells can be added without violating the constraints (3.3), the terms *maximal size matching* and *maximal weight matching* are used in the literature.

3. *Contention resolution.* In the execution of the heuristic algorithm for the determination of a maximal match, a strategy is necessary to solve contentions due to edges with equal metrics, source or destination. The contention resolution typically is either random (RO, random order), or based on a deterministic scheme; round robin (RR) and sequential search (SS) are frequent choices in the latter case, the difference lying in the starting point chosen in the selection process, which is state-dependent for RR, and state-independent for SS.

The first phase is preliminary to the other two, that are instead interlaced.

As we show in [7], the phase that has the most profound impact on performance is the metrics computation, whereas the different heuristics to obtain good matchings have a deep impact on the algorithm complexity. As regards metrics, we consider the following alternatives:

- **QO** (*Queue occupancy*). In this case $w_{ij} = 1$ iff $X_{ij} > 0$, otherwise $w_{ij} = 0$. This is the metrics adopted by iSLIP. The adoption of this metrics leads to the search for a maximal size match.
- **QL** (*Queue length*). The metrics in this case is the number of cells in the queue: $w_{ij} = X_{ij}$. This is the metrics adopted by RPA and iLQF.
- **CA** (*Cell age*). The metrics in this case is the time already spent in the queue by the cell at the queue head. This is the metrics adopted by iOCF.
- **ML** (*MUCS Length*). MUCS uses a metrics that is derived from queue lengths as:

$$w_{ij} = \frac{X_{ij}}{\sum_{k=1}^N X_{ik}} + \frac{X_{ij}}{\sum_{k=1}^N X_{kj}} \quad (3.4)$$

As regards the heuristic matching, the choices adopted in the considered IQ scheduling algorithms are the following:

- **IS** (*Iterative search*). In this case, during a first step, each input interface sends all its transmission requests with the associated metrics to the relevant output

Algorithm	Metrics	Heuristics	Contention resolution	
			Input	Output
iSLIP	QO	IS	RR	RR
iLQF	QL	IS	RO	RO
iOCF	CA	IS	RO	RO
MUCS	ML	MG	SS	RO
RPA	QL	RV	RO	SS

Table 3.1. Characterization of the considered IQ scheduling algorithms

interfaces (w_{ij} is sent from input interface i to output interface j). These select one among the arriving requests by choosing the largest metrics value, and resolving ties according to a contention resolution scheme (output contention). The accepted requests are then sent back to the input interfaces. If an input interface receives more than one acceptance, it selects one by choosing that with the largest metrics value, and resolving ties according to a contention resolution scheme (input contention). The successive steps are equal to the first one, but they concern only the transmission requests from input interfaces that received no acceptance, as well as those that were satisfied in previous steps (the repetition of these requests is necessary to progressively freeze the match configuration). This heuristics was proposed in [61], and it is adopted by iSLIP, iLQF and iOCF.

- **MG** (*Matrix greedy*). Consider the weight matrix \mathbf{W} . The algorithm consists of (up to) N steps, in each of which the largest element(s) w_{ij} of \mathbf{W} is (are) selected, and the corresponding cell transmissions are enabled (provided that no conflict arises if ties for the largest metrics exist; otherwise a conflict resolution is necessary) before reducing the matrix by eliminating the entries corresponding to enabled cell transfers. This is the heuristics adopted by MUCS.
- **RV** (*Reservation vector*). In this case the algorithm is based on a sequential access to a reservation vector with N records, where input interfaces sequentially declare their cell transfer needs and the associated metrics, possibly overwriting requests with lower metrics values. A second sequential pass over the vector allows the confirmation of requests, or the reservation of other transfers for the inputs whose original reservations were overwritten. This is the heuristics adopted by RPA.

Table 3.1 gives a synoptic view of the considered IQ scheduling algorithms, where for each algorithm we give the type of metrics used, the heuristic algorithm adopted for the identification of matchings, and the approach used for contention resolution.

3.3 Simulation study

To evaluate the performance of IQ, OQ, and CIOQ switching architectures, we conducted quite a large number of simulation experiments. In IQ switches, each input queue VOQ_{ij} can store a finite number of cells Q_{\max} ; when a cell directed to output j arrives at input i , and queue VOQ_{ij} is full, the cell is lost. No buffer sharing among queues is allowed.

Simulation runs were executed until the estimate of the average cell delay reaches with probability 0.95 a relative width of the confidence interval equal to 2%. The estimation of the confidence interval width is obtained with the *batch means approach* [80].

3.3.1 Software of simulation

A numerical simulator has been developed to study the performance of all the considered architectures. The simulator has been written in C and is compiled under *Linux* (some older versions were also tested under *Windows 95*). It is freely available. Main features of the simulator (which is about 850 KBytes of source code) are the following:

- support of pure IQ, pure OQ and CIOQ switching architectures;
- support of the router architecture;
- flexible traffic generation, at cell and packet level, for unicast and multicast traffic;
- support of about 25 different families of scheduling algorithms;
- automatic statistical analysis of simulation results;
- support of large switches (switches with more than 1,000 ports were studied for unicast traffic; switches with more than 500,000 output ports were studied for multicast traffic).

3.3.2 Traffic scenarios

In the following chapters, we consider traffic generated either at *cell level* or at *packet level*. When the studied architecture is the overall router switching system, the traffic is generated at packet level. Section 6.3.1 will describe how the traffic is generated at packet level. When the studied architecture is the cell-switch, the traffic is generated at cell level. In this case, each cell is generated according to an i.i.d. Bernoulli

process such that it satisfies the constraints about the maximum input load ρ and the normalized cell arrival rate matrix Γ (both defined in Section 3.1).

We describe different types of traffic patterns that were used to test the performance of the cell switches. By $|k|$ we denote the computation $k \bmod N$.

- **Uniform scenario.** This scenario is the common test-bed in literature. In this case, for the normalized arrival matrix Γ , $\gamma_{ij} = 1/N^2 \forall i, j$.
- **Diagonal scenario.** In this case, $\gamma_{ii} = 2/(3N)$ and $\gamma_{i|i+1|} = 1/(3N) \forall i$. For all other i, j , $\gamma_{ij} = 0$. Only two diagonals of the traffic matrix receive packets, and one diagonal receives on average twice packets than the other. This scenario is derived from the unbalanced traffic pattern presented in [7].
- **Logdiagonal scenario.** In this case $\gamma_{ij} = 2\gamma_{i|j+1|}$ are non-zero, $\forall i, j$. For example, for the first row $\gamma_{1j} = \beta 2^{N-j}$, where β is a normalization constant. Intuitively, every diagonal of the traffic matrix is loaded twice than its own right diagonal.
- **Sparse scenario.** The following traffic matrix is considered (here, in the case $N = 6$). This matrix is designed to be very critical, as we will see in our simulations:

$$\Gamma = \frac{1}{3N} \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

3.3.3 Performance indexes

Results are presented with graphs where the following performance indices are plotted versus the switch load. The latter is defined as the ratio between the input traffic load and the total capacity of input/output lines (hence, the switch load is comprised between 0 and 1).

- **IQ length.** This is the value of the occupation of each VOQ queue. We consider the average value over time and over all queues, which is denoted as *Mean IQ Len*. We also consider the average over time of the queue length of the maximum size queue, denoted as *Max IQ Len*.
- **Cell delay.** This is the time spent by cells in the cell-switch queues. We consider the average delay over all packets. Note that we can use Little's Law (that

holds for non-workconserving systems like IQ switches) to compute the average cell delay D from the average IQ length L under an input load ρ (assuming all the inputs uniformly loaded): $D = NL/\rho$.

- **Relative weight.** If M is the matching computed by our algorithm, the relative weight is defined as: weight of matching M divided by the weight of the MWM. It is positive and always less or equal to 1: when its value is 1 means that M is a maximum weight matching.
- **Packet delay.** This is the overall delay of a packet, considering the ISM module, the internal cell-switch queues, the ORM module, and the final packet FIFO. It is computed only for packets completely delivered at switch outputs, measuring the time from the ingress of the last cell of the packet into the ISM module until the egress of that same last cell from the final packet FIFO. Constant delay components are removed; hence a single-cell packet traverses an empty packet switch in null time, and a packet comprising k cells has a best-case delay equal to $2(k - 1)$ slots, due to wait in the segmentation and reassembly phases. We shall consider only the average value of packet delay.

Since the aim of the performance evaluation is a comparison of IQ and CIOQ architectures with OQ switches, we consider also *relative* performance indices, i.e., we divide the absolute value taken by a performance index in the case of IQ and CIOQ architectures by the value taken by the same performance index in an OQ packet switch loaded with the same traffic pattern.

3.4 Theoretical study

Several analytical approaches have been used to study IQ switches. An IQ switch can be modeled as a controlled queueing system and can be studied using stochastic modeling approaches.

The most common theoretical question is if the queueing system is stable, that it no queue is growing to infinity when the arrivals are “admissible”. To deal with this question, usually the “Lyapunov” function method is used. This method has been also extended to the estimation of delays, as presented in [57].

We introduce now the notation used in the following proofs, in Section 3.4.2 we show the basic theorems about the stability of controlled queueing systems. Section 3.4.3 states the stability of switching systems using the MWM for scheduling.

3.4.1 Referred notation

The following notation is used in the proofs in the subsequent chapters. Note that, by default, a vector is meant as row vector.

- N is the size of the switch and M is the total number of queues in the switch: $M = N^2$.
- t or n is the discrete time variable.
- let $X, Y \in \mathbb{R}^n$, with $X = [x_i]$ and $Y = [y_i]$; $XY \triangleq XY^T = \sum_i x_i y_i$ is the scalar product among vectors X and Y . $\|X\|$ is the Euclidean norm of X , i.e. $\|X\|^2 = \sum x_i^2$.
- X_t is the queue occupancy vector at time t ; its size is N^2 and the i -th element is the queue occupancy of the i -th queue.
- A_t is the arrival vector at time t ; its size is N^2 and the i -th element is the arrival at the i -th queue at time t . Note that it is a binary vector.
- D_t is the departure vector at time t ; its size is N^2 and the i -th element is the departure from the i -th queue at time t . It is a binary vector and corresponds to a matching. Note that a complete matching can be seen as a permutation of the N outputs with respect to the N inputs, hence there is a bijective relation between matchings of size N and permutations of N elements. For example, the identity matching joining input i with output i , $1 \leq i \leq N$, corresponds to the permutation $\pi = (\pi_1, \pi_2, \dots, \pi_N) = (1, 2, \dots, N)$.
- \mathcal{M} is the set of all possible departure vectors, or possible matchings of size N ; clearly, $|\mathcal{M}| = N!$.
- D_X^* is the MWM with state of the queue X : $D_X^* X = \max_{D \in \mathcal{M}} \{DX\}$.
- $X_{t+1} = [X_t + A_t - D_t]^+$ describes the system evolution, with the following convention about the time slot t : X_t is sampled, then the arrivals occur and finally the departures occur.
- $\Pr(H)$ is the probability of event H and $E(Z)$ is the expected value of the random variable Z .
- $\Lambda \triangleq E[A_t] = E[A]$, with λ_{ij} is the rate from input i to output j , assuming stationary traffic.

3.4.2 Stability of controlled queueing systems

We are now ready to recall some useful results about the stability of controlled queueing systems.

Definition 2. A system of queues achieves *100% throughput*, or is *rate stable*, if

$$\lim_{t \rightarrow \infty} \frac{X_t}{t} = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^{t-1} (A_i - D_i) = 0 \quad \text{with probability 1}$$

Definition 3. A system of queues is said to be *weakly stable* if, for every $\epsilon > 0$, there exists $B > 0$ such that

$$\lim_{t \rightarrow \infty} \Pr\{\|X_t\| > B\} < \epsilon$$

Definition 4. A system of queues is said to be **strongly stable** if

$$\limsup_{t \rightarrow \infty} E\|X_t\| < \infty$$

We consider now the theorem studied by Tweedie [93] in the form considered by Tassiulas [91].

Theorem 1 (Tweedie). *Suppose that $\{X_t\}_{t=1}^{\infty}$ is an aperiodic and irreducible Markov chain with countable state space \mathcal{X} . Let $f(X)$ and $g(X)$ be real non-negative functions. Consider T a finite subset of \mathcal{X} and T^c its complement. If*

$$g(X) \geq f(X) \quad X \in T^c \tag{3.5}$$

$$E[g(X_2)|X_1 = X] < \infty, \quad X \in T \tag{3.6}$$

$$E[g(X_2) - g(X_1)|X_1 = X] < -f(X), \quad X \in T^c \tag{3.7}$$

then the Markov chain is ergodic and

$$Ef(\tilde{X}) < \infty$$

where the random variable \tilde{X} has the steady state distribution of the Markov Chain $\{X_t\}_{t=1}^{\infty}$.

We assume that the stochastic process describing the evolution of the system of queues is an irreducible Discrete Time Markov Chain (DTMC), whose state vector at time n is $Y_t = (X_t, K_t)$, $Y_t \in \mathbb{N}^{M''}$, $X_t \in \mathbb{N}^M$, $K_t \in \mathbb{N}^{M'}$ and $M'' = M + M'$. Y_t is the combination of the queue length vector X_t and a vector K_t of integer parameters. Most systems of discrete-time queues of practical interest can be described with models that fall in the DTMC class. The following general criterion for the strong stability of systems falling into this class is therefore useful.

Theorem 2. Given a system of queues with state vector $Y_t=(X_t,K_t)$, and a function $V(X_t) = X_t W X_t^T$ (called Lyapunov function), if there exists a symmetric copositive² matrix $W \in \mathbb{R}^{M \times M}$, and two positive real numbers $\epsilon \in \mathbb{R}^+$ and $B \in \mathbb{R}^+$, such that:

$$E[V(X_{t+1}) | Y_t] < \infty \quad (3.8)$$

$$E[V(X_{t+1}) - V(X_t) | Y_t] < -\epsilon \|X_t\| \quad \forall Y_t : \|X_t\| > B \quad (3.9)$$

then the system of queues is strongly stable. In addition, all the polynomial moments of the queue lengths distributions are finite.

Proof. This is a re-phrasing of the results presented in [56, Sect.IV] and can be proved using Theorem 1. Consider $g(X,K) = V(X)$, $f(X,K) = \epsilon \|X\|$ and $T = \{X : V(X) \leq B\}$. Note that for $X_t \in T^c$,

$$g(X_t, K_t) = X_t W X_t \geq \epsilon X_t X_t \geq \epsilon \|X_t\| = f(X_t, K_t)$$

Now using Theorem 1 we can state:

$$E[\epsilon \|\tilde{X}\|] < \infty \quad \Rightarrow \quad E[\|\tilde{X}\|] < \infty$$

where $X_t \rightarrow \tilde{X}$ in distribution. □

Being the identity matrix I a symmetric positive semidefinite matrix, hence a copositive matrix, it is possible to state that:

Corollary 1. Given a system of queues with state vector $Y_t=(X_t,K_t)$, if there exists $\epsilon \in \mathbb{R}^+$, $B \in \mathbb{R}^+$ such that:

$$E[X_{t+1} X_{t+1} | Y_t] < \infty \quad (3.10)$$

$$E[X_{t+1} X_{t+1} - X_t X_t | Y_t] < -\epsilon \|X_t\| \quad \forall Y_t : \|X_t\| > B \quad (3.11)$$

then the system of queues is strongly stable, and all the polynomial moments of the queue lengths distributions are finite.

3.4.3 Optimal policy for unicast traffic

Thanks to the previous theorems, it was proved [62, 65, 92] the following theorem, which is the first significant result about the stability of IQ cell-switches.

Theorem 3. In a switch with VOQ buffering scheme, fed by admissible i.i.d. Bernoulli traffic, if the scheduler computes the MWM at each time slot, that is:

$$D_t = D_{X_t}^* = \arg \max_{D \in \mathcal{M}} \{D X_t\}$$

then the system is strongly stable, i.e. it achieves 100% throughput.

²An $M \times M$ matrix Q is copositive if $X Q X^T \geq 0 \quad \forall X \in \mathbb{R}^{+M}$.

Chapter 4

Learning scheduling algorithms for unicast traffic

In this chapter, we discuss a new approach to design practical and efficient schedulers for high-speed switching fabrics. We wish to answer to the following question: Is it possible for an algorithm to compete with the throughput and delay performance of MWM and yet be simple to implement? If yes, what feature of the scheduling problem should be exploited?

The answer lies in recognizing the main feature of the high speed switch scheduling problem: the correlation of the state of the system along the time. Note that packets arrive (depart) at most one per input (output) per time slot. This means queue-lengths, taken to be the weights by MWM, change very little during successive time slots. Thus, an heavy matching will continue to be heavy for a few more time slots, suggesting that carrying some information, or retaining **memory**, between iterations should help simplify the implementation while maintaining a high level of performance. In [91], Tassiulas was the first to exploit this fact to design a scheduler for an IQ switch, as we will see later.

We shall see that this feature considerably simplifies the implementation and provides a high-performance. We also use some novel techniques for simplifying the implementation.

Hardware parallelism: Finding good heavy matchings essentially involves a search procedure, requiring a comparison of the weight of multiple matchings. In Section 4.8 we propose an algorithm, called APSARA, that exploits a natural structure on the space of matchings and uses parallelism in hardware to conduct this search efficiently. In particular, it requires a *single iteration*, is stable, and its delay is comparable to that of MWM.

Randomization: In a variety of situations where the scalability of deterministic algorithms is poor, randomized algorithms are easier to implement and provide a

surprisingly good performance. The main idea is simply stated: Basing decisions upon a few random samples of a large state space is often a good surrogate for making decisions with complete knowledge of the state. See [71] for a general exposition of randomized algorithms, and [70, 83] for applications of load-balancing and of web-cache replacement. In [91], Tassiulas proposed a pure randomized scheme for scheduling in IQ switches and its stability properties were stated. This scheme can be considered one of the first simple and implementable schedulers with 100% throughput, and it is included in our general framework of “learning algorithms”. Some other hybrid schemes were also proposed to exploit randomization in scheduling. For example, in [34] the *Shakeup* technique was introduced, which can be used in conjunction with some other scheduling algorithm, in order to improve the weight of the matching. The *Shakeup* technique is based on the idea of randomly changing some edges in the initial matching, obtained by the scheduler. From the complexity point of view, this hybrid approach introduces an additional complexity to the tradition scheduler, but also an improvement of the performance. In our work, our goal is to exploit the possibilities of a pure random algorithm, with lower complexity than the known algorithms.

Observation of arrivals: Since the increase in queue-lengths is entirely due to arrivals, it might help to use a knowledge of recent arrivals in finding a matching. Being the arrival process a stochastic process, arrivals can be considered also as source of randomness.

The rest of this chapter exploits the above observations and proposes some new algorithms. In Section 4.1 a general framework to design 100% throughput schedulers is considered; it is based on a learning approach. Section 4.2 proves theoretically that the learning approach achieves 100% throughput. Sections 4.3 and 4.4 show how to design some particular modules of the architecture, in order to achieve also good delays. Section 4.5 show how to extend the architecture and to consider multiple stored matchings. The last part of the chapter discusses three possible schemes to implement the learning approach. The three schemes are *LAURA*, *APSARA* and *SERENA*. They are individually described and their performance are discussed from Section 4.6 to Section 4.11. The final section 4.12 discusses the performance of all the tree approaches together, also for very large switch size. We shall comment upon the suitability of these approaches for use in different categories of routers.

4.1 The learning approach

We start with a formal definition:

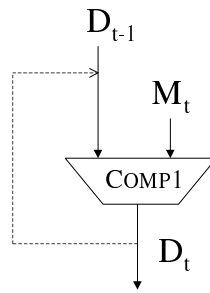


Figure 4.1. Basic architecture of a learning scheme

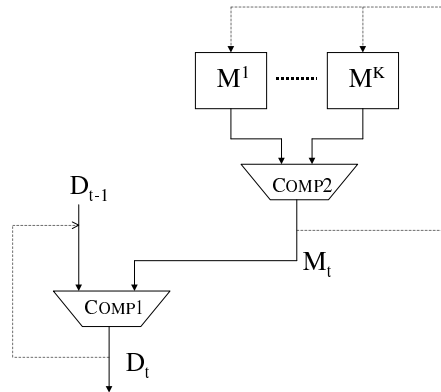


Figure 4.2. Possible architecture implementing a stable learning scheme

Definition 5. A scheduling algorithm is defined *learning* if $D_t X_t \geq D_{t-1} X_t$ for all t .

Intuitively, an algorithm belongs to the learning approach if, at each time, tries to improve the weight of the matching used during the previous time, which is stored in memory. Of course, this is a very general definition of class of algorithms. From some point of view, it is a kind of genetic approach.

The scheme in Fig. 4.1 represents the basic architectural block of the *learning approach*. It implements the concept of *memory*, since the old matching D_{t-1} is considered and it is improved by means of the given matching M_t , called **probe-matching**. This improvement process is similar to a genetic process, where the final solution is obtained by iterative steps of improvement (or learning).

Informally, the probe-matching M_t should be a “good” sample in \mathcal{M} , the space

of all possible matchings. As good sample we mean that it can hit the MWM with a finite probability, as we will explain in details in Section 4.2.3.

The first example of learning approach was proposed by Tassiulas [91], who studied the throughput of the switch when M_t is picked randomly among all the possible $N!$ matchings and D_t is the heavier among M_t and D_{t-1} . Tassiulas’s scheme achieves 100% throughput under any admissible i.i.d. Bernoulli traffic pattern. From a practical point of view, it is also very simple to implement, but the packet delays can be unacceptable for real traffic, as we will show in Section 4.7. For this reason, the scheduler in [91] can be considered a notable scheme but not of practical relevance.

Our contribution has been to devise a general framework to study learning algorithms, leading to new stable and practical scheduling algorithms.

Consider now the diagram in Fig. 4.2. It represents a general scheme to design a scheduler algorithm in hardware. Up to K modules (M^1, \dots, M^K) run in parallel, exploiting all the possible parallelism degree available in the hardware architecture. Several variants can be envisaged, depending on computation of the probe-matching M_t .

The two functional modules COMP1 and COMP2 are called **weight-boosters** and have the general property that the weight of the output matching should be greater or equal to the input matchings. Formally, given a set of matchings $H \subset \mathcal{M}$ and the state of the queues X_t as inputs, the output matching $Y \in \mathcal{M}$ of a weight-boosters is such that $Y X_t \geq M X_t$ for all $M \in H$.

Some possible schemes for the weight-boosters are:

- $\text{MAX}(H; X_t)$: $Y = \arg \max_{M \in H} \{M X_t\}$, i.e. the output matching is the heaviest among all the input matchings;
- $\text{MERGE}(H; X_t)$, i.e. the output matching is obtained by “combining” the best edges of the input matchings; this scheme will be described in Section 4.3.
- $\text{ARR-MERGE}(H; A_t; X_t)$, i.e. the output matching is obtained by combining the heaviest edges of the input matchings with the edges that have just received an arrival; this scheme exploits the information present in the arrival process and it will be described in Section 4.3.

The MWM scheduling algorithm belongs to our learning framework, since in this case $M_t = D_t^*$ and $D_t = \text{MAX}(D_{t-1}, M_t; X_t)$.

Note that, in Fig. 4.2, the feedback of D_t is necessary to guarantee the learning property of the algorithm, despite the mechanism used to compute M_t (the feedback of M_t is optional).

A minor point regards how to initialize D_0 whenever the learning scheme is started or reset. D_0 can be set equal to either a void matching or a random matching. The average performances are not affected by this choice.

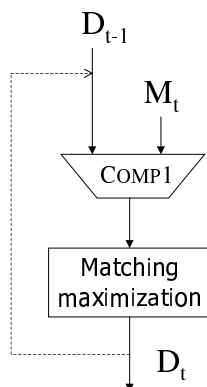


Figure 4.3. To maximize the matching, a new module is added at the egress of COMP1.

4.1.1 Maximization of the matching

If the output matching D_t is not maximal, the performance of the algorithm can be improved by making D_t maximal, at the expense of an additional computational complexity. Fig. 4.3 shows the additional module to maximize the output matching of COMP1.

Consider the graph given by the set I of unmatched inputs and the set O of unmatched outputs. Let $|I| = |O| = k$ be the number of unmatched inputs (or outputs). Two main approaches can be envisaged to maximize a matching:

- **MAX-DEP.** This is a *state-dependent maximization*. It finds a matching M' among I and O such that each edge of M' has non-null weight. The complexity to implement this module is $O(k^2)$ since it is necessary to compute a maximal matching probing the state of k^2 queues (to check if they are empty or not). In the worst case, $O(k^2)$ comparisons are needed. Note that a maximal size matching scheduler like iSLIP could be used in this module, whose time-complexity (exploiting a parallel implementation) is $O(\log k)$.
- **MAX-IND.** This is a *state-independent maximization*. It finds a matching M' among I and O without taking into account the state of the queues. The complexity of this process is $O(k \log k)$, equivalent of computing a random permutation. From a practical point of view, since the matching selection is state independent, it can easily be pipelined or it can be obtained by static tables of pre-computed matchings. In other words, the real complexity can decrease also to $O(1)$, although k can vary with the time.

The modules of matching maximization are optional and their effect is significant only for low traffic load, as we will observe later.

4.2 Stability of the learning approach

In this section we prove formally that the learning approach is able to achieve 100% throughput under any admissible i.i.d. Bernoulli arrival process. The following two subsections state some preliminary results to the main result, presented in Section 4.2.3. We refer always to the notations introduced in Section 3.4.1. A reader uninterested in the formal proof can jump directly to Section 4.3.

4.2.1 Some preliminary results

This first lemma is an important (but easy) application of Birkhoff's Theorem considered in combinatorics [58]. The proof is well known in literature; here it is reported only for the sake of completeness.

Lemma 1 (Birkhoff's). *Let $\Lambda = [\lambda_{ij}] = E[A]$ be the rate vector and*

$$\lambda_M = \max \left\{ \sum_i \lambda_{ij}, \sum_j \lambda_{ij} \right\}.$$

Assume the traffic is admissible, i.e. $\lambda_M < 1$. Given the queue length vector X , it exists $\epsilon > 0$ such that it holds:

$$(\Lambda - D_X^*)X < -\epsilon\|X\|.$$

Proof. Since the traffic is admissible, $\lambda_M < 1$ and it exists a finite set $\{D_i : D_i \subseteq \mathcal{M}\}$ and a finite set $\{\alpha_i : 0 \leq \alpha_i \leq 1\}$ such that:

$$\Lambda < \frac{1}{\lambda_M} \Lambda \leq \sum_i \alpha_i D_i \quad \text{with} \quad \sum_i \alpha_i = 1 \quad (4.1)$$

For definition, $D_X^* X \geq D_i X$, then:

$$\begin{aligned} \Lambda X - D_X^* X &\leq \lambda_M \left(\sum_i \alpha_i D_i \right) X - D_X^* X \leq \\ &\leq \lambda_M \sum_i \alpha_i D_X^* X - D_X^* X = (\lambda_M - 1) D_X^* X \leq -(1 - \lambda_M) \|X\| \end{aligned} \quad (4.2)$$

Choose now $\epsilon = (1 - \lambda_M)/2$ and note that: $-(1 - \lambda_M) \|X\| < -\epsilon \|X\|$. □

Lemma 2. Given a matching $D' \in \mathcal{M}$, a set of z matchings $\{D_i, 0 \leq i \leq z-1\} \subseteq \mathcal{M}$ and a set of z arrival vectors $\{A_i, 0 \leq i \leq z-1\}$, for any finite $z \in \mathbb{N}$, it holds:

$$\left| \sum_{i=0}^{z-1} (A_i - D_i) D' \right| \leq zM \quad (4.3)$$

Proof. By Chauchy's inequality, assuming $N \geq 2$, we have:

$$(A - D)D' \leq \|A - D\| \|D'\| \leq \sqrt{2N} \sqrt{N} = \sqrt{2}N < N^2 = M \quad (4.4)$$

Summing z terms less than M , we obtain the lemma. \square

Lemma 3. For any finite $z \in \mathbb{N}$, it holds:

$$|D_{X_t}^* X_t - D_{X_{t-z}}^* X_{t-z}| \leq 2zM \quad (4.5)$$

Proof. From one time slot to another, the weight of the maximum weight matching can increase or decrease by, at most, $2N$. Hence,

$$\begin{aligned} |D_{X_t}^* X_t - D_{X_{t-z}}^* X_{t-z}| &\leq \left| \sum_{k=0}^{z-1} D_{X_{t-k}}^* X_{t-k} - D_{X_{t-k-1}}^* X_{t-k-1} \right| \leq \\ &\leq \sum_{k=0}^{z-1} 2N \leq 2zN \leq 2zN^2 = 2zM \quad (4.6) \end{aligned}$$

\square

4.2.2 Stability of *epoch-learning* scheduling algorithms

To study the stability of the learning approach, we need to study the properties of the class of epoch-learning scheduling algorithms, to which learning algorithms will be related. An *epoch-learning* algorithm ψ works through *epochs*. At the beginning of the n -th epoch, at time t_n , the algorithm stores the state of the queue and then, by using a learning approach, finds the MWM in subsequent time slots. At time $s_n \geq t_n$ it finds the MWM, referred to the state of the queues at time t_n . At time $s_n + 1$ the algorithm stores the new state of the queue and a new epoch starts.

The following definitions determine better these concepts.

Definition 6. Sequences $\{t_n\}$ and $\{s_n\}$ of stopping times are defined as follows:

$$\begin{aligned} t_0 &= 0 & s_0 &= 0 \\ s_n &= \inf\{t \geq t_n : D_{s_n} X_{t_n} = D_{X_{t_n}}^* X_{t_n}\} & \text{for } n \geq 1 \\ t_{n+1} &= s_n + 1 & \text{for } n \geq 0 \\ z_n &= s_n - t_n & \text{for } n \geq 0 \end{aligned}$$

Fig. 4.4 shows an example of this definition of stopping times.

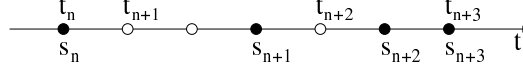


Figure 4.4. Example of definition of stopping times: the dot is full when the weight of the current matching is maximum, otherwise is empty

Definition 7. Epoch n -th is defined as the interval $[t_n, s_n]$, for $n \geq 1$, and it lasts z_n time slots.

Definition 8. Tracking phase n -th, for $n \geq 1$, is defined as the interval $\mathcal{Z}_n = \{i : i \geq 0, t_n + i < s_n\}$. Note that $|\mathcal{Z}_n| = z_n$ and it can happen that $\mathcal{Z}_n = \emptyset$ when $z_n = 0$. During this phase, the algorithm searches the MWM referred to the state of the queue at time t_n , i.e. it is learning $D_{X_{t_n}}^*$. Note also that for $i \in \mathcal{Z}_n$, it holds: $D_{t_n+i} X_{t_n} < D_{X_{t_n}}^* X_{t_n}$.

Theorem 4. If the algorithm ψ has the following properties:

$$\exists B \in \mathbb{R}^+ : B < \infty \quad \text{and} \quad E[z_n^k] < B \quad \text{for } k = 1, 2, 3 \quad (\text{P.1})$$

$$D_{t_n+i} X_{t_n} \geq D_{t_n} X_{t_n} - \alpha i^2 \quad \text{for } i \in \mathcal{Z}_n \quad (\text{P.2})$$

$$D_{t_n} X_{t_n} \geq D_{t_{n-1}} X_{t_n} = D_{s_{(n-1)}} X_{t_n} \quad (\text{P.3})$$

$$D_{s_n} X_{t_n} = D_{X_{t_n}}^* X_{t_n} \quad (\text{P.4})$$

with $\alpha \geq 0$, then algorithm ψ is stable, i.e. it achieves 100% throughput, under any admissible i.i.d. Bernoulli traffic pattern.

Note that properties (P.2), (P.3) and (P.4) define formally an epoch-learning algorithm. The meaning of this theorem is that, if for an epoch-learning algorithm the duration of an epoch has the first three moments finite, then the algorithm is stable.

Proof. Note that the sequence $\{t_n\}$ is proper because of property (P.1). In this case, the proof of this theorem has a similar to the one described in [4].

The system evolution satisfies the following equation:

$$X_{t_{n+1}} = X_{t_n} + \sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i})$$

Observe that $E[A_t A_t] < \infty$. By using the Lyapunov function $V(X_{t_n}) = X_{t_n} X_{t_n}$, we want to show that it exists $\epsilon > 0$ such that:

$$\lim_{\|X_{t_n}\| \rightarrow \infty} \frac{E[V(X_{t_{n+1}}) | X_{t_n}] - V(X_{t_n})}{\|X_{t_n}\|} < -\epsilon \quad (4.7)$$

Consider now the numerator of (4.7).

$$\begin{aligned}
 & E [V(X_{t_{n+1}}) | X_{t_n}] - V(X_{t_n}) = \\
 & = E \left[\left(X_{t_n} + \sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i}) \right) \left(X_{t_n} + \sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i}) \right) \right] - V(X_{t_n}) = \\
 & = 2E \left[\sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i}) X_{t_n} + z_n D_{s_n} X_{t_n} - z_n D_{s_n} X_{t_n} \right] = \\
 & = 2E[z_n](E[A] - D_{s_n})X_{t_n} + 2E[D_\delta]X_{t_n} \quad (4.8)
 \end{aligned}$$

having used Wald's Lemma [96, §2-13] and having defined: $D_\delta = z_n D_{s_n} - \sum_{i=0}^{z_n-1} D_{t_n+i}$. We want to show now that $D_\delta = o(\|X_{t_n}\|)$, that is:

$$\lim_{\|X_{t_n}\| \rightarrow \infty} \frac{E[D_\delta]X_{t_n}}{\|X_{t_n}\|} = 0 \quad (4.9)$$

Because of property (P.2), $D_{t_n+i}X_{t_n} \geq D_{t_n}X_{t_n} - \alpha i^2$, $i \in \mathcal{Z}_n$, we can state:

$$\sum_{i=0}^{z_n-1} D_{t_n+i}X_{t_n} \geq \sum_{i=0}^{z_n-1} D_{t_n}X_{t_n} - \alpha \sum_{i=0}^{z_n-1} i^2 = z_n D_{t_n}X_{t_n} - g(z_n) \quad (4.10)$$

where $g(z_n) = \alpha(z_n - 1)z_n(2z_n + 3)/6 = O(z_n^3)$.

Hence:

$$D_\delta X_{t_n} \leq z_n(D_{s_n} - D_{t_n})X_{t_n} + g(z_n) \quad (4.11)$$

Because of property (P.3):

$$\begin{aligned}
 D_{t_n}X_{t_n} & \geq D_{t_{n-1}}X_{t_n} = D_{t_{n-1}}X_{t_{(n-1)}} + D_{t_{n-1}} \sum_{i=0}^{z_{(n-1)}-1} (A_{t_{(n-1)}+i} - D_{t_{(n-1)}+i}) \geq \\
 & \geq D_{X_{t_{(n-1)}}}^* X_{t_{(n-1)}} - z_{(n-1)}M \quad (4.12)
 \end{aligned}$$

where the last inequality is due to Lemma (2) and the property $D_{t_{n-1}} = D_{X_{t_{(n-1)}}}^*$. Recall that $D_{s_n} = D_{X_{t_n}}^*$; by combining equation (4.11) with (4.12) and using Lemma 3:

$$\begin{aligned}
 D_\delta X_{t_n} & \leq z_n(D_{X_{t_n}}^* X_{t_n} - D_{X_{t_{(n-1)}}}^* X_{t_{(n-1)}}) + z_{(n-1)}M + g(z_n) \leq \\
 & \leq z_n(2z_{(n-1)}M) + z_{(n-1)}M + g(z_n) = g'(z_n)M \quad (4.13)
 \end{aligned}$$

where $E[g'(z_n)] = E[O(z_n^3)] = O(B)$, since $\{z_n\}$ are i.i.d. for definition of $\{t_n\}$ as stopping times and for property (P.1).

Now we can finally show relation (4.9):

$$\lim_{\|X_{t_n}\| \rightarrow \infty} \frac{E[D_\delta]X_{t_n}}{\|X_{t_n}\|} = \frac{E[g'(z_n)]M}{\|X_{t_n}\|} = \frac{O(B)M}{\|X_{t_n}\|} = 0 \quad (4.14)$$

if property (P.1) holds.

By definition of s_n , $D_{s_n}X_{t_n} = D_{X_{t_n}}^* X_{t_n}$, hence by combining (4.8), (4.14) and Lemma 1, we obtain:

$$\lim_{\|X_{t_n}\| \rightarrow \infty} \frac{E[V(X_{t_{n+1}}) | X_{t_n}] - V(X_{t_n})}{\|X_{t_n}\|} < \frac{-2\epsilon E[z_n] \|X_{t_n}\|}{\|X_{t_n}\|} < -2\epsilon E[z_n] \quad (4.15)$$

which is what we were looking for. \square

4.2.3 Stability of a learning scheduling algorithm

One immediate application of the Theorem 4 is the following Theorem 5, which is the main result about learning algorithms. It states a kind of “test-bed” by which a learning algorithm can be tested to understand if it is stable, in other words it states sufficient conditions for a learning algorithm to be stable.

Definition 9. A scheduling algorithm *works off-line* if the queue evolution of the system is given by $X_t = X_{t-1} = \dots = X_0$ (no departures and no arrivals effect the state of the system, “frozen” at time $t = 0$) and the departure vector D_t is computed at each time.

We are interested in the first time the algorithm, working off-line, finds the MWM for the initial (and fixed) state of the queue X_0 .

Definition 10. Given a scheduling algorithm working off-line and a state of the queues $X_0 \in \mathbb{Z}_+^M$, with $\|X_0\| < \infty$ and any initial matching D_0 . The *tracking time* $T(X_0)$ is defined as follows:

$$T(X_0) = \inf\{t \geq 0 : D_t X_0 = D_{X_0}^* X_0\}$$

Note that, in general, the tracking time is a random variable which depends also on D_0 . Now we are ready for the main result:

Theorem 5. Consider a learning algorithm ψ working off-line. If for any $X_0 \in \mathbb{Z}_+^M$, with $\|X_0\| < \infty$, and for any initial matching D_0 , $T(X_0)$ is such that: $E[T(X_0)^k] < \infty$, $k = 1, 2, 3$, then the algorithm ψ is stable, i.e. achieves 100% throughput under any admissible traffic pattern.

Proof. The proof can be done directly from Theorem 4. We show how why all the assumptions are satisfied. Property (P.2) of Theorem 4 means that the epoch-learning *behaves like* working off-line during an epoch, since an epoch-learning algorithm “freezes” the state of the system at the beginning of the epoch and all the D_{t_n+i} are computed, for $i \in \mathcal{Z}_n$, *independently* from X_{t_n+i} and A_{t_n+i} . Thanks to the following Lemma 4, assumption (P.2) is satisfied, with $\alpha = 2M$. According to the notation of Theorem 4, $z_n = T(X_n)$ and assumption (P.1) holds. The algorithm is learning, hence assumption (P.3) holds. Since the learning algorithm finds the MWM after $T(X_n)$, also assumption (P.4) holds. \square

Lemma 4. *For a learning algorithm it holds:*

$$D_{t+k}X_t \geq D_tX_t - 2k^2M \quad 0 < k < \infty$$

Note that this theorem is referred (implicitly, for $k = 1$) by Tassiulas in Property (16) of his paper [91].

Proof. X_{t+k} , for $k \in \mathbb{N}$, can be written as:

$$X_{t+k} = X_t + \sum_{i=0}^{k-1} (A_{t+i} - D_{t+i}) \quad (4.16)$$

Now evaluate:

$$\begin{aligned} D_{t+k}X_t - D_{t+k-1}X_t &= (D_{t+k} - D_{t+k-1}) \left(X_{t+k} - \sum_{i=0}^{k-1} (A_{t+i} - D_{t+i}) \right) = \\ &= (D_{t+k} - D_{t+k-1})X_{t+k} - (D_{t+k} - D_{t+k-1}) \sum_{i=0}^{k-1} (A_{t+i} - D_{t+i}) \geq \\ &\geq -(D_{t+k} - D_{t+k-1}) \sum_{i=0}^{k-1} (A_{t+i} - D_{t+i}) \geq -2kM \quad (4.17) \end{aligned}$$

where the last line is due to the learning properties ($D_{t+k}X_{t+k} \geq D_{t+k-1}X_{t+k}$) and to Lemma 2. Now we can expand recursively:

$$D_{t+k}X_t \geq D_{t+k-1}X_t - 2kM \geq (D_{t+k-1}X_t - 2kM) - 2kM \geq \dots \quad (4.18)$$

$$D_{t+k}X_t \geq D_{t+k-j}X_t - j(2kM) \quad \text{for } j \in [0, k] \quad (4.19)$$

By setting $j = k$, we obtain:

$$D_{t+k}X_t \geq D_tX_t - 2k^2M \quad (4.20)$$

\square

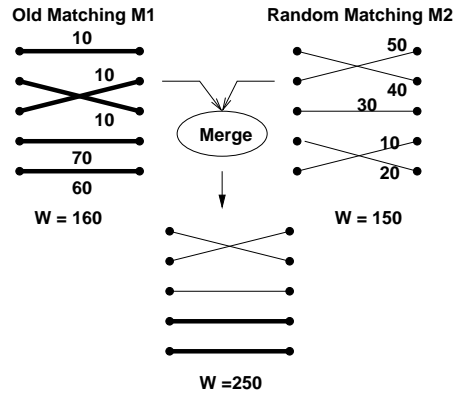


Figure 4.5. An illustration of the MERGE applied to matchings M1 and M2. The final matching is the maximum weight matching on the subgraph defined by edges of M1 and M2.

4.3 Architecture of the *weight-boosters*

In Section 4.1 we presented three modules to implement the weight-booster. The simplest structure is the MAX one, which finds the heaviest matching among the input matchings. Here we describe the two other procedures, MERGE and ARR-MERGE; they are more complex of MAX but they can improve the performance of the scheduler.

4.3.1 MERGE procedure

Let $M_1, M_2 \in \mathcal{M}$. Aim of this module is to combine the best edges of M_1 and M_2 to find the final matching \tilde{M} ; in other words, this algorithm finds all the possible augmenting paths in a graph through the edges corresponding to two matchings. It can be shown that the final matching is the MWM on the graph given by merging M_1 and M_2 .

The interesting property of this procedure is that it finds the MWM computed on the graph given by merging the edges of M_1 with M_2 ; its complexity is only $O(N)$.

Given a bipartite graph and two matchings M_1 and M_2 for this graph, the MERGE procedure returns a matching \tilde{M} whose edges belong either to M_1 or to M_2 . MERGE works as follows.

Color the edges of M_1 red and the edges of M_2 green. Start at output node j_1 and follow the red edge to an input node, say i_1 . From input node i_1 follow the (only) green edge to its output node, say j_2 . If $j_2 = j_1$, stop. Else continue to trace a path of alternating red and green edges until j_1 is visited again. This gives a “cycle” in the

subgraph of red and green edges.

Suppose the above cycle does not cover all the red and green edges. Then there exists an output j outside this cycle. Starting from j repeat the above procedure to find another cycle. In this fashion find all cycles of red and green edges. Suppose there are m cycles, C_1, \dots, C_m at the end. Then each cycle, C_i , contains two matchings: G_i which has only green edges, and R_i which has only red edges. The MERGE procedure returns the matching

$$\tilde{M} = \bigcup_{i=1}^m \arg \max_{S \in \{G_i, R_i\}} SX_t$$

Fig. 4.5 illustrates the MERGE procedure. It is easy to see that the final matching \tilde{M} is the maximum weight matching on the subgraph defined by edges of M_1 and M_2 .

The algorithm can be also described in the following way. Let $G' = M_1 \cup M_2$, in other words G' is a bipartite graph with the edges obtained by the union of matchings M_1 and M_2 . Let \tilde{M} be initially a bipartite graph with no edges.

Phase A. Mark all N input and output nodes of G' as *unmarked*. Repeat the next six steps till all nodes in G' are *marked*; after visiting at most $2N$ edges, the phase ends:

- (a) Let v be an *unmarked* input node. Set path $P_v = \emptyset$. Let $\psi(P_v)$ denote weight of path P_v . Initially, set $\psi(P_v) = 0$.
- (b) Let $(v, w) \in M_1$. Add (v, w) to P_v , and set $\psi(P_v) = \psi(P_v) + \psi(v, w)$.
- (c) Let $(w, u) \in M_2$. Add (w, u) to P_v , and set $\psi(P_v) = \psi(P_v) - \psi(w, u)$.
- (d) If $u = v$ stop. Else, repeat (b)-(c) for with u in place of v , and update P_v accordingly.
- (e) Let $M_1(P_v)$ denote the edges of M_1 that belong to P_v , and similarly denote $M_2(P_v)$. If $\psi(P_v) \geq 0$, set $\tilde{M} = \tilde{M} \cup M_1(P_v)$; else $\tilde{M} = \tilde{M} \cup M_2(P_v)$.
- (f) If any node q is *unmarked*, start from (a) with q in place of v .

Phase B. Output \tilde{M} as the solution, which has property:

$$\psi(\tilde{M}) \geq \max\{\psi(M_1), \psi(M_2)\}$$

Note that the complexity of this algorithms is $O(N)$, since each one of the $2N$ edges is visited at most once.

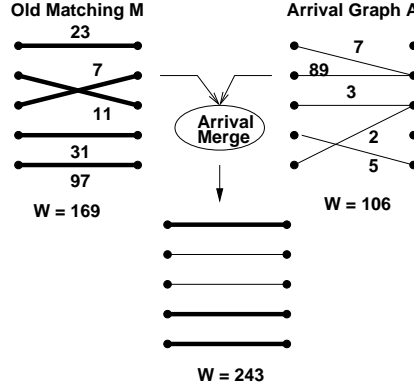


Figure 4.6. An illustration of the ARR-MERGE procedure, given the matching M and the arrival graph A .

4.3.2 ARR-MERGE procedure

ARR-MERGE runs on a complete matching M and an arrival vector A ; it returns a matching \tilde{M} . Let $G' = M \cup A$, that is G' is a graph with edges obtained by the union of M and A . Goal of ARR-MERGE is to approximate the maximum weight matching on G' , by combining the best edges of M and A . Whereas M is a matching, A is not necessarily a matching. This is because multiple edges can be incident on the same output node due to multiple arrivals to that output. Therefore, we cannot simply combine M and A by using the MERGE procedure.

Let \mathcal{U}^* be the outputs not having arrivals, that is, no input has arrival for these outputs. Note that, if $\mathcal{U}^* = \emptyset$, then A forms a perfect matching. Let \tilde{M} be initially a bipartite graph with no edges. The following steps describe the algorithm:

- (i) Mark all N input and output nodes of G' as *unmarked*.
- (ii) Compute $\tilde{M}' = \text{ARR-AUGMENT}(G')$. This will mark all the nodes of \mathcal{U}^* .
- (iii) Consider the input-output nodes of G' that are *unmarked* after $\text{ARR-AUGMENT}(G')$. Let G'' be subgraph of G' induced by these unmarked nodes and the edges of M, A among them. Note that, none of the output node in G'' belongs to \mathcal{U}^* . Hence, $G'' = M_1 \cup M_2$, with $M_1 \subseteq M, M_2 \subseteq A$ and both M_1, M_2 are perfect matchings on the nodes of G'' .
- (iv) Compute $\tilde{M}'' = \text{MERGE}(M_1, M_2; X_t)$.
- (v) Output $\tilde{M} = \tilde{M}' \cup \tilde{M}''$ as solution, which has the property $\psi(\tilde{M}) \geq \psi(M)$. Note that, M' and M'' are matchings on disjoint set of input-output nodes.

The ARR-AUGMENT(G') procedure follows the phases below:

1. Build-Path Phase. For each output u in \mathcal{U}^* , do the following:

- (o) Set path $P_u = \emptyset$ and the *potential* of the path $\psi(P_u) = 0$, $\psi(u) = 0$. Mark u as visited.
- (a) Let $(u,v) \in M$. Add (u,v) to P_u and set $\psi(P_u) = \psi(P_u) + \psi(v,u)$. Mark v as visited, and set $\psi(v) = \psi(u) + \psi(v,u)$.
- (b) **If** no packet arrived to v , that is, there is no edge incident on v that belongs to A , then consider edge (v,u) . Close the path P_u by adding this edge and set $\psi(P_u) = \psi(P_u) - \psi(v,u)$. This is a complete cycle with u as its identifier. Start from (o) with new $u' \in \mathcal{U}^*$.
- (c) **else if** an arrival occurred, then let $(v,x) \in A$. There are three cases as follows:
 - (c.1) **If** x is unmarked (because does not belong to any cycle), add (v,x) to P_u and set $\psi(P_u) = \psi(P_u) - \psi(v,x)$, $\psi(x) = \psi(P_u)$. Mark x as visited.
 - (c.2) **else if** x is marked and belongs to P_u . Then close the path P_u into cycle by adding appropriate edge. Let $y \in P_u$ be predecessor of x . Consider

$$V_1 = |\psi(v) - \psi(v,u)|$$

$$V_2 = |\psi(y) + \psi(y,x) - \psi(y,u)| + |\psi(v) - \psi(v,x) - \psi(x)|$$

If $V_1 \geq V_2$, then a cycle is defined by adding the edge (v,u) to P_u ; the identifier of the cycle is u and its potential is set equal to $\psi(v) - \psi(v,u)$. **Else if** $V_1 < V_2$, then two cycles are created. One cycle has identifier u with potential set to $\psi(y) + \psi(y,x) - \psi(y,u)$. The other cycle has identifier x , and potential set to $\psi(v) - \psi(x) - \psi(v,x)$. Note that, edge like $(v,u) \notin A$ was used, which we call as *virtual edge*. Go to (o) with new $u' \in \mathcal{U}^*$.

- (c.3) **else** x is marked and $x \notin P_u$ but belongs to already created cycle with identifier ζ of potential V . Let predecessor of x in the cycle ζ is y . Consider the following:

$$\begin{aligned} V_{\text{OPEN}} &= |V + \psi(v) - \psi(v,x) + \psi(y,x) - \psi(y,u)| \\ V_{\text{CLOSE}} &= |\psi(v) - \psi(v,u)| + |V| \end{aligned}$$

If $V_{\text{OPEN}} > V_{\text{CLOSE}}$ then we merge cycle ζ with P_u , by removing (y,x) and adding (y,u) , (v,x) . We identify new cycle with u . Set the

potential of cycle u to $[V + \psi(v) - \psi(v,x) + \psi(y,x) - \psi(y,u)]$. **Else** **if** $V_{\text{OPEN}} \leq V_{\text{CLOSE}}$, create a new cycle with identifier u by adding virtual edge (v,u) with potential $[\psi(v) - \psi(v,u)]$. Go to (o) with new $u' \in \mathcal{U}^*$.

(d) Let $(x,w) \in M$. Add (x,w) to P_u , set $\psi(w) = \psi(x) + \psi(x,w), \psi(P_u) = \psi(w)$. Go to (b) by setting $v = w$.

2. Find-Best-Path Phase. By construction in the previous phase, the nodes belonging to different cycles are disjoint. This phase finds whether to use edges from A or from M that belong to these cycles alternatively, in order to get higher weight. This is done as follows:

- I.** Consider all the cycles created in previous phase one by one. Let one such cycle is \mathcal{C} .
- II.** If potential of \mathcal{C} is ≥ 0 then select all the edges of \mathcal{C} belonging to M , else select all the edges of \mathcal{C} not included in M .
- III.** Add all the selected edges to \tilde{M}' .

The complexity of ARR-MERGE procedure is $O(N)$ since at most $3N$ are considered and visited.

4.4 Computation of the *probe-matching*

Several techniques can be envisaged to compute the probe-matching M_t . Better the weight of M_t is, lower the delays will be experienced by packets traveling the switch. On the basis of the hardware complexity constraints, it is possible to find the best compromise among performance and implementation complexity. In the following subsections, different approaches are discussed to compute a “good” M_t .

4.4.1 Exploiting the randomization

In a variety of situations where the scalability of deterministic algorithms is poor, randomized algorithms are easier to implement and provide a surprisingly good performance. The main idea is simply stated: Basing decisions upon a few random samples of a large state space is often a good surrogate for making decisions with complete knowledge of the state. Depending on the problem of scheduling, two kind of randomized strategies can be envisaged:

1. A random scheme independent from the state of the switch (e.g., the occupation of the queues, the waiting time of the buffered cells), since its implementation

complexity is low (e.g., it can be implemented by using pre-computed random matchings). For example, at each time a matching is chosen at random and it is used for serving the queues. But such algorithm performs very poorly, due to ignorance of the state of switch and hence of the incoming traffic.

2. A random scheme dependent on the switch state. For example, the probability of serving a queue can be set proportional to its occupation. This gives better performance but expensive random selection scheme, and hence it is not useful in reducing the complexity of the scheduling algorithm.

From the last two observations, our goal is to come up with a randomized scheme which is state independent but gives a matching efficient for the actual state of the switch.

In several scheduling algorithms, during some phases of the algorithm some decisions (usually, the contention resolution phase) are made by random choices (see [7] for a general taxonomy and a list of algorithms using a random contention resolution). But all these schemes are deterministic heuristics and they use randomness only in “negligible” phases, when a random choice can be emulated by a deterministic round robin without affecting the performance. Few *inherently* randomized scheduling algorithms have been proposed so far.

We want to discuss here some basic techniques to find a “good” random probe-matching.

A randomized scheduler without memory

Before doing that, in order to understand better the meaning of the learning approach, consider this simple algorithm, called ALGO1: At each time t , let the matching D_t used by ALGO1 be the heaviest of d ($d > 1$) matchings chosen uniformly at random.

The following theorem shows that ALGO1 is not stable, even when $d = O(N)$.

Theorem 6. *For an $N \times N$ switch and for any $d \leq cN$, where $c > 0$, ALGO1 does not deliver 100% throughput.*

The proof can be found in [31]. Note that the above theorem has a much stronger implication: *Any* scheduling algorithm that only uses $d = O(N)$ random matchings cannot achieve 100% throughput. Further, there is no assumption about the distribution of the packet arrival process, only a rate assumption.

RND-MATCH1

This previous results adds strength to the learning approach, using Tassiulas’s scheme [91]. Let RND-MATCH1 a matching uniformly picket at random in \mathcal{M} .

Corollary 2. *Let M_t be given by RND-MATCH1. The learning approach is stable, under any admissible Bernoulli i.i.d. traffic pattern.*

Proof. Fix $X_0 \in \mathbb{Z}_+^M$, with $\|X_0\| < \infty$. Referring to Theorem 5, if $p = 1/N!$, $P[T(X_0) = k] = (1 - p)^{k-1}p$ for any X_0 ; hence the first three moments of T are finite. \square

RND-MATCH2(X_t)

Aim of procedure RND-MATCH2 is to obtain unknown heavy edges with low complexity, given the state of the queues X_t . It is an iterative procedure working as follows. At each iteration, it chooses a random permutation and retains few heavy edges of the selected permutation. This gives a matching which is weight matrix dependent, biased towards higher weights. Note that, though the random selection done at every stage is independent of the weight matrix, the final matching has higher weight than the initial one with high probability. It finds random samples dependent on the weight matrix. At the same time, the random scheme should have low complexity, hence the random selection should be done independently of the weight matrix. Our iterative algorithm has both of these features.

Let $\mathcal{F}_\eta(M)$ denote the minimal set of edges in a matching M carrying at least a fraction η ($0 < \eta \leq 1$) of its weight. We shall call η the *selection factor*.

RND-MATCH2 is the following iterative procedure: Initially, all inputs and outputs are marked as *unmatched*. The following steps are repeated in each of I iterations, where I is typically $\log_2 N$:

- (i) Let i be the current iteration number. Let $k \leq N$ be the number of *unmatched* input-output pairs. Out of the $k!$ possible matchings between these unmatched input-output pairs, a matching $S_i(k)$ is chosen uniformly at random.
- (ii) If $i < I$, retain the edges corresponding to $\mathcal{F}_\eta(S_i(k))$ and mark the nodes they cover as *matched*. If $i = I$, then retain all edges of $S_i(k)$.

Corollary 3. *Let M_t be a matching given by RND-MATCH2(X_t). The learning approach is stable, under any admissible Bernoulli i.i.d. traffic pattern.*

Proof. Let I be the number of iterations and fix any $X_0 \in \mathbb{Z}_+^M$, with $\|X_0\| < \infty$. The probability of selecting all the edges corresponding to the MWM in k steps is given by:

$$P[T(X_0) = k] = p_k$$

(it can be computed by a chain) with $p_k > \delta$ for all $1 \leq k \leq I$. Hence, the first three moments of $T(X_0)$ are finite and Theorem 5 can be applied. \square

We now discuss the complexity of this procedure. Since the selection of each random sample is independent from the state of the system, we consider its complexity equal to finding a random sample. Let us suppose that there are I iterations in the random sample selection. In each iteration, a random matching on the “remaining nodes” is chosen uniformly at random. Since this random selection does not depend on the state of the weight matrix, we assume that it is a $O(1)$ operation. If this assumption cannot hold, the complexity of finding a random permutation of N elements is equal to $O(N \log_2 N)$, but this does not affect the overall complexity of the algorithm. The edges that constitute the “heaviest” η fraction of this random matching are kept and the rest of the nodes are used for the next iteration. If it was the last iteration, then the selection of heavier edges is avoided. The selection of the heavy edges involves:

- (a) Ordering the edges according to their weights, which is $O(N \log_2 N)$.
- (b) Choosing the heavier edges that constitute the η fraction of the net weight, which is $O(N)$.

Since there are I iterations, the *Random Selection Phase* is $O(IN \log_2 N)$. If V random samples are chosen, then the total running time will be $O(VIN \log_2 N)$.

4.4.2 Exploiting the Hamiltonian walk

We now discuss the concept of an Hamiltonian walk on the set of all matchings. Consider a graph with $N!$ nodes, each corresponding to a distinct matching, and all possible edges between these nodes. Let $Z(t)$ denote an Hamiltonian walk on this graph; that is, $Z(t)$ visits each of the $N!$ distinct nodes exactly once during times $t = 1, \dots, N!$. We extend $Z(t)$ for $t > N!$ by defining $Z(t) = Z(((t - 1) \bmod N!) + 1)$. One simple algorithm for such an Hamiltonian walk is described, for example, in Chapter 7 of [73]. This is a very simple algorithm that requires $O(1)$ space and $O(1)$ time, to generate $Z(t + 1)$ given $Z(t)$. Under this algorithm $Z(t)$ and $Z(t + 1)$ differ in exactly two edges. For $N = 3$ this algorithm generates the matchings (permutations): $Z(1) = (1, 2, 3)$, $Z(2) = (1, 3, 2)$, $Z(3) = (3, 1, 2)$, $Z(4) = (3, 2, 1)$, $Z(5) = (2, 3, 1)$, $Z(6) = (2, 1, 3)$, $Z(7) = Z(1)$, and $Z(8) = Z(2)$, etc. . We can now define the following probe-matching:

HAM-MATCH

The procedure HAM-MATCH returns at time t the matching $Z(t)$.

Corollary 4. *Let M_t be given by HAM-MATCH. The learning approach is stable, under any admissible Bernoulli i.i.d. traffic pattern.*

Proof. Fix $X_0 \in \mathbb{Z}_+^M$, with $\|X_0\| < \infty$. Referring to Theorem 5, if $p = 1/N!$, $P[T(X_0) = k] = (1 - p)^{k-1}p$; hence the first three moments of $T(X_0)$ are finite. \square

4.4.3 Exploiting the arrivals

Arrivals are a significant source of randomness and help to find heavy edges. The randomization in LAURA is used to obtain unknown *heavy* edges with low complexity. Note that an edge becomes *heavy* if the corresponding queue receives many arrivals but fewer services. Hence probing edges that receive arrivals can help in finding *heavy* edges. Hence, the randomness provided by arrivals can be captured and exploited to find heavy edges.

A quite complex way to exploit the information in arrivals, is to use a special weight-booster procedure, called ARR-MERGE and discussed in Section 4.3.2.

Another procedure is the following:

RND-ARRIVAL

The simplest way to exploit arrivals is to consider the graph corresponding to A_t . A_t is not necessarily a matching. This is because multiple edges can be incident on the same output node due to multiple arrivals to that output. When A_t is not a matching, let \mathcal{U}^* denote collection of outputs which have one or more arrival edges incident on them. For every $u \in \mathcal{U}^*$ do the following: among the arrival edges incident on output u , pick the edge (for example, one edge at random or the edge with the highest weight) and discard the remaining edges. At the end of this process, each output in \mathcal{U}^* is matched with exactly one input.

Corollary 5. *Let M_t be RND-ARRIVAL applied to A_t . The learning approach is stable, under any admissible Bernoulli i.i.d. traffic pattern.*

Proof. Fix $X_0 \in \mathbb{Z}_+^M$, with $\|X_0\| < \infty$. The arrival process is stationary and Bernoulli i.i.d.. Hence, there is a finite probability $\delta > 0$ such that A_t is the MWM. Referring to Theorem 5, if $p = \delta$, $P[T(X_0) = k] = (1 - p)^{k-1}p$; hence the first three moments of $T(X_0)$ are finite. \square

4.4.4 Exploiting the neighbors

A set of “good” candidates for M_t is the set of all the possible *neighbors* of D_{t-1} . To define formally the concept of neighbor, we use some concepts from the algebra theory. We consider now the two following definitions:

Definition 11 (Generator set). Let Π the group of all possible permutations of size N ; hence, $|\Pi| = N!$. If the set $\Phi \subseteq \Pi$ is a generator of the whole group Π , Φ is called *generator set*.

Definition 12 (Neighbor in Φ). Given two matchings $M \in \mathcal{M}$ and $M_0 \in \mathcal{M}$. M is said to be *neighbor of M_0 with respect to the generator set Φ* if it exists $\pi \in \Phi$ such that π applied to M_0 (seen as permutation) gives M (seen as permutation).

Let us consider two examples of generator set:

- Φ given by *general swapping*: a neighbor of $M_0 \in \mathcal{M}$ is obtained by swapping two edges in M_0 ; $|\Phi| = \binom{N}{2} \approx N^2/2$. Let us define $\mathcal{N}(M_0)$ the set of neighbors of $M_0 \in \mathcal{M}$, with respect to Φ ; note that $|\mathcal{N}(M_0)| = |\Phi| \approx N^2/2$. For example, the matching M_0 for a 4×4 switch generates the following 6 neighbors:

$$M_0 = (1,2,3,4) \rightarrow \\ (2,1,3,4), (3,2,1,4), (4,2,3,1), (1,3,2,4), (1,4,3,2), (1,2,4,3)$$

- Φ given by *adjacent swapping*: a neighbor of $M_0 \in \mathcal{M}$ is obtained by swapping two adjacent edges in M_0 ; $|\Phi| = N$.

$$M_0 = (1,2,3,4) \rightarrow \\ (2,1,3,4), (4,2,3,1), (1,3,2,4), (1,2,4,3)$$

In the theory of algebraic groups, several generator sets of Π are known. Note that it is not trivial to find a generator set. For example, the set of all the N permutations in the form $\pi_i = |i+1|_N$, $\pi_{|i+1|_N} = |i+2|_N$ and $\pi_{|i+2|_N} = i$ (which correspond to N “cycles” of size 3) is not a generator set.

From a practical point of view, referring to the architecture presented in Fig. 4.2, each module M^i could compute the i -th neighbor of D_t with respect to the set Φ . The number of parallel modules running is $|\Phi|$ and it can be $O(N^2)$ if the generator set is given by general swapping or can be $O(N)$ if the generator set is given by adjacent swapping. Intuitively, larger $|\Phi|$ is, smaller the average delay is; the best choice is given by the best compromise between hardware complexity and performance. Note also that M_t could be either (at least) one randomly chosen neighbor of D_{t-1} or the heavier among all $|\Phi|$ possible neighbors. At the same time, the best performance can be achieved when Φ is the set of all possible permutation; this is a trivial case, it is usually impractical, but for very small size switch it remains the best (simple) solution to the scheduling problem. As we will see later, in Section 4.8.6, considering only a number of neighbors much less than $N!$ cannot be proved to be sufficient to guarantee the stability of the system.

4.5 Schedulers with parallel learning schemes

Consider the case that multiple learning schemes are running in parallel, like in Fig. 4.7.

$D_{1,t-1} \dots D_{S,t-1}$ are S stored matchings, which are tried to be augmented through $M_{1,t} \dots M_{S,t}$. The final matchings $D_{1,t} \dots D_{S,t}$ should be “purged” to

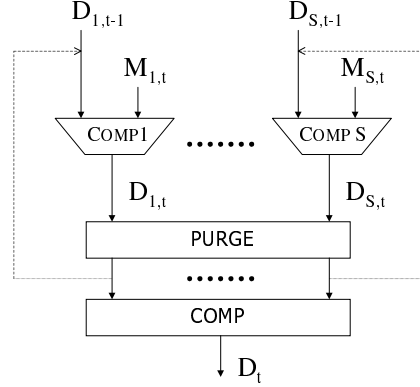


Figure 4.7. Multiple learning schemes working in parallel. To exploit the multiple memories, it is necessary to include a “purging” module.

avoid bad cases like: $M_{1,t} \dots M_{S,t}$ are equal to the MWM at time t and all the S stored matchings becomes the same matching, hence the multiple memories are useless. To preserve some degree of diversity among the stored matchings, “redundant” matchings are purged.

Definition 13. The *distance* among two matchings M_1 and M_2 (non, necessarily, complete) is equal to N minus the number of common edges they share. Hence, two *complete* matchings equal have null distance, whereas two void matchings have distance equal to N .

To purge redundant matchings, we propose to adopt the following procedure. Since matchings with high weight are desirable, start with the lowest weighted matching, say \tilde{M} . Consider now in increasing order of weight all the stored matchings heavier than \tilde{M} . Assume that the stored matching is M' . If the distance among \tilde{M} and M' is not greater than d_{MIN} (called *minimal distance*), then the procedure purges (removes) \tilde{M} and initializes the corresponding learning scheme. At the end, among the retained matchings, select the heaviest one as D_t . In the worst case, $O(S^2)$ matchings are compared.

4.6 LAURA approach

Consider the learning approach where RND-MATCH1 computes M_t and COMP1 is MAX; we call this scheme RANDOM. As shown by Tassiulas [91], RANDOM achieves 100% throughput. However, its delay performance is quite poor (as we will observe in Fig. 4.13). In RANDOM, when the weight of a heavy matching resides

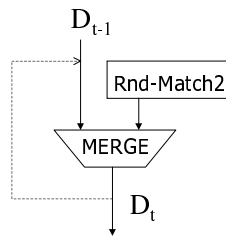


Figure 4.8. Basic architecture of LAURA-M1

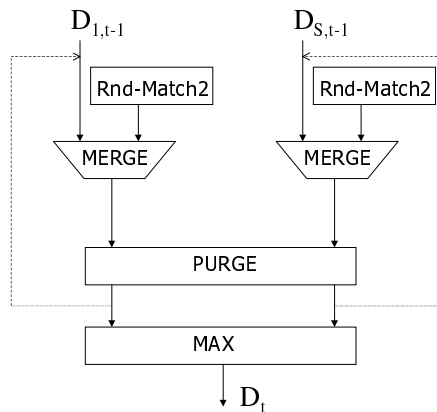


Figure 4.9. Architecture of LAURA

in a few heavy edges, it is more important to remember the heavy edges than it is to remember the matching itself.

The last observation motivates our algorithm LAURA (*Low-complexity Algorithm Using Randomized Augmentation*), which is a scheduling algorithm exploiting randomization. LAURA iteratively augments the weight of the current matching by combining its heavy edges with the heavy edges of a (non-uniformly) randomly chosen matching. There are three main features in the design of LAURA:

1. learning scheme;
2. non-uniform random sampling through RND-MATCH2 (refer to Section 4.4);
3. weight-boosting through MERGE (refer to Section 4.3.1).

Refer to Fig. 4.8 for the “minimal” architecture of LAURA, called LAURA-M1, i.e.

Parameter	Symbol	Value
switch size	N	32
stored matchings	S	2
iterations	I	5
selection factor	η	0.5
minimal distance	d_{MIN}	16
buffer size	Q_{max}	10,000

Table 4.1. Simulation parameters for LAURA and Max-LAURA used with multiple parallel learning schemes

with unitary memory. Let D_t be the matching used by LAURA-M1 at time t . At time t , LAURA-M1 does the following:

- (a) $M_t = \text{RND-MATCH2}$.
- (b) $D_t = \text{MERGE}(D_{t-1}, M_t; X_t)$.

Refer to Fig. 4.9 for the “normal” architecture of LAURA, with S parallel learning schemes (see Section 4.5).

It can be shown that the running time of LAURA is bounded by $O(IN \log_2 N + N)$. In our simulation study, we set $I = \log_2 N$. Thus running time of algorithm is $O(N \log_2^2 N)$.

A variant of LAURA is Max-LAURA, which uses a MAX-DEP module to maximize the matching.

4.7 LAURA performance

In the following sections we discuss the performance of LAURA (and its variant) for stationary traffic and for transient traffic. LAURA is simulated with the settings of Table 4.1.

4.7.1 Stationary analysis of LAURA

In this section, we study the performance of different algorithms on different test cases, in terms of different performance measures. The traffic patterns used to obtain these results are stationary in distribution.

We study the performance of the following algorithms: MWM, iSLIP, iLQF, MUCS, RPA, RANDOM along with LAURA and Max-LAURA.

First we consider the uniform traffic scenario, which is the most basic traffic scenario considered in literature. Figs. 4.10, 4.11 and 4.12 present performance of different algorithms with respect to different measures. Note that the average delay in

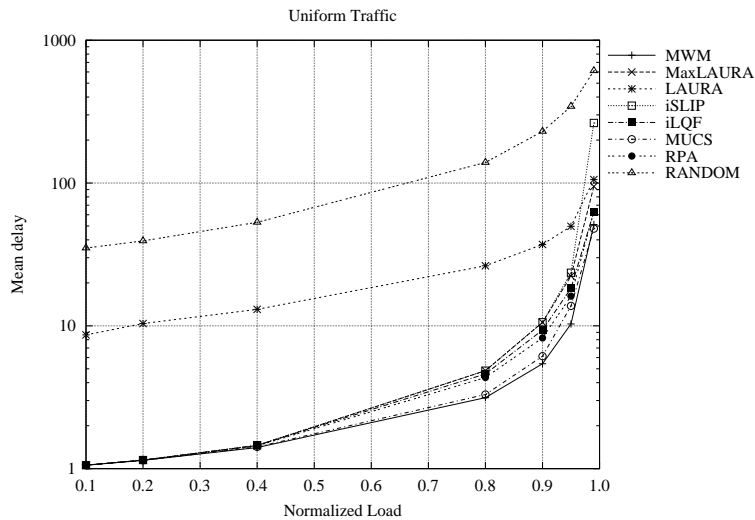


Figure 4.10. Mean delay for uniform traffic for LAURA and other scheduling algorithms

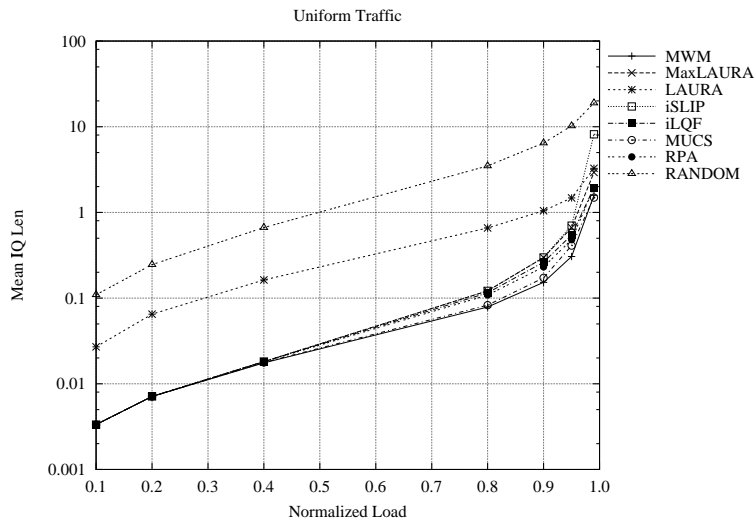


Figure 4.11. Mean IQ length for uniform traffic for LAURA and other scheduling algorithms; this graph is related to the previous one thanks to Little's formula

Fig. 4.10 is related to the average queue length in Fig. 4.11, thanks to Little's Law. With respect to all measures, all algorithms show the same qualitative behavior. The

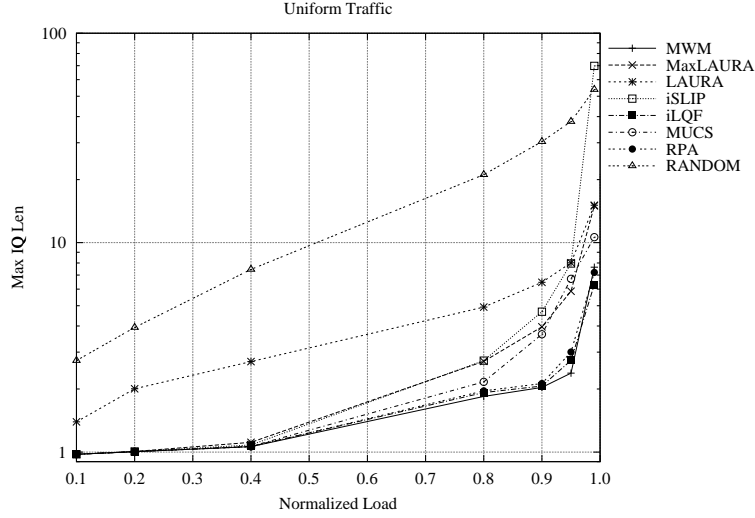


Figure 4.12. Mean Max IQ length for uniform traffic for LAURA and other scheduling algorithms

following are interesting results to note:

1. Both LAURA and Max-LAURA are stable and well-behaved, which is the basic property of efficient algorithms.
2. For small values of load, LAURA performs worse than most of the other algorithms, but this misbehavior is practically negligible. But this can be easily explained: LAURA is a randomized algorithm which does not necessarily find a maximal matching at lower values of load. In Max-LAURA, we force it to be a maximal matching. Hence, it performs as well as any other algorithm, at lower load values. Further note that, for high values of load, LAURA and Max-LAURA behave identically, since LAURA is not maximal.
3. RANDOM is stable but its delay is much worse than the other algorithms.

We consider now diagonal traffic, since it is more critical than uniform traffic for testing the performance of scheduling algorithms. With respect to the different performance metrics *Mean IQ Len*, *Max IQ Len* (recall its definition in Section 3.3.3) and *Mean Delay*, relative performances of the algorithms remain the same. Hence we will present only results with respect to *Mean IQ Len*. Fig. 4.13 plots *Mean IQ Len* for different algorithms. iSLIP, iLQF, RPA are *unstable* for high load values and experiences losses. MUCS and MWM perform identically.

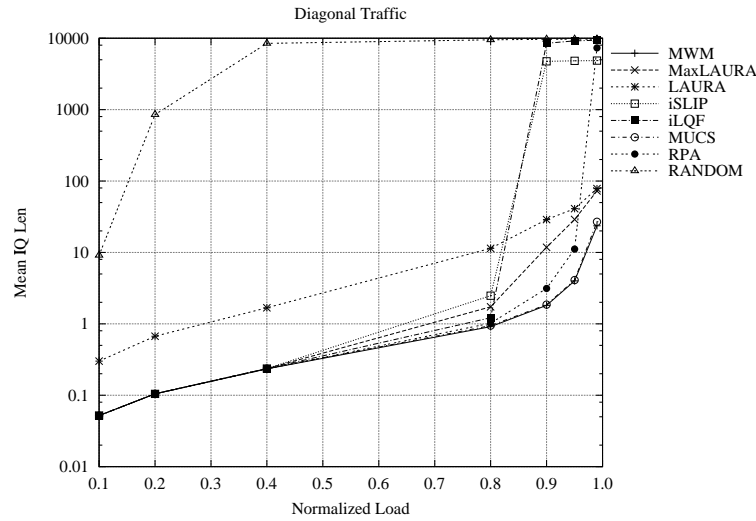


Figure 4.13. Mean IQ length for diagonal traffic for LAURA and other scheduling algorithms

RANDOM starts to experience losses for a load 0.4; this does not contradict Theorem 5, since this theorem proves only the stability of RANDOM when the queues are large enough. Hence, RANDOM experiences unacceptable average delays (although bounded, coherently with Theorem 5) from a practical point of view. Note also that this is not a surprising result. Some stable algorithms can experience very large delays/queue lengths, much greater than some other unstable algorithms under particular traffic patterns. For example, the scheduler that computes the MWM only among the queues larger than an arbitrary x_{MIN} is stable, but of course the average delay will be greater or equal to Nx_{MIN}/ρ , with ρ the maximum normalized load among all the inputs.

Both LAURA and Max-LAURA are instead stable and perform almost as good as MWM. Also if they can be considered pure random algorithms, their performances are much better than RANDOM. As before, LAURA performs worse than Max-LAURA for lower load values but performs identical for higher values. Note that below we deliberately omit our comments about RANDOM, whose behavior is practically unacceptable in all the studied traffic scenarios.

Fig. 4.14 plots the *Mean IQ Len* for algorithms under logdiagonal scenario. As in diagonal traffic scenario, iSLIP, iLQF and RPA are unstable. In this scenario, MUCS performs little worse than Max-LAURA and LAURA. Again, LAURA and Max-LAURA perform a little worse than MWM.

Fig. 4.15 shows the *Mean IQ Len* for sparse scenario. This scenario separates

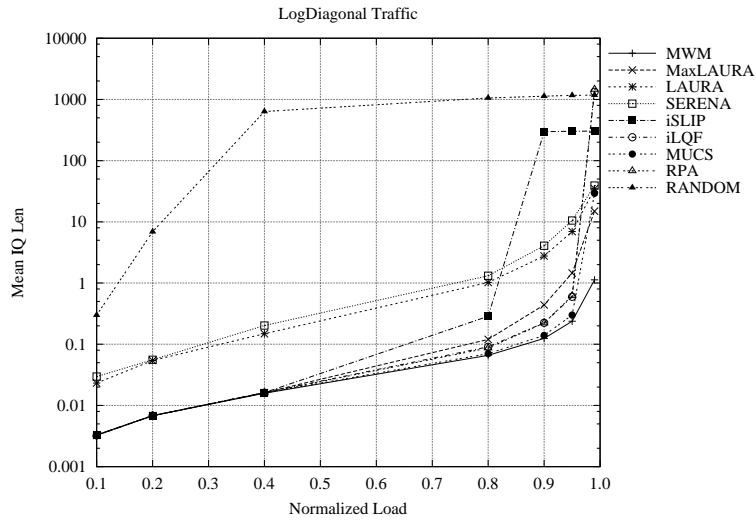


Figure 4.14. Mean IQ length for logdiagonal traffic for LAURA and other scheduling algorithms

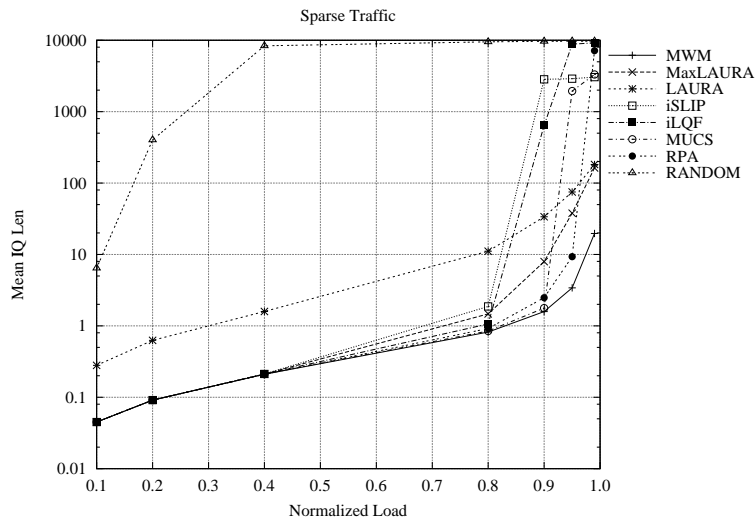


Figure 4.15. Mean IQ length for sparse traffic for LAURA and other scheduling algorithms

all algorithms very distinctively. In that respect, this traffic pattern is very critical. Other than MWM, LAURA and Max-LAURA, all other algorithms are unstable for

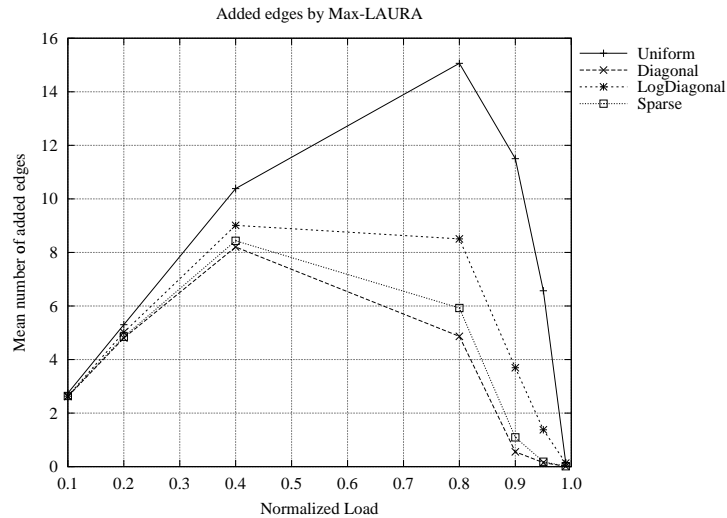


Figure 4.16. Mean number of edges added by Max-LAURA with respect to LAURA

high load values. LAURA and Max-LAURA perform a little worse than MWM but their performance are comparable to MWM. As usual, LAURA performs worse than Max-LAURA for smaller loads, but for higher loads both performs almost identical.

The only difference between LAURA and Max-LAURA is in the extra edges added to make the matching *maximal*. We plot the number of edges added on average by Max-LAURA with respect to the load in Fig. 4.16, under the four traffic scenarios considered. For very low load, since the queues are almost empty, only few edges are added. For higher load, more and more edges are added. But near the saturation load, the number of edges added is almost negligible, since almost all the queues are non-empty.

The stationary analysis shows a very important property. LAURA and Max-LAURA are the *only* two stable algorithms (except, of course, MWM) under *all* the traffic scenarios we studied.

4.7.2 Transient analysis of LAURA

LAURA is a learning algorithm. Hence by changing traffic behavior, the old stored-learned matching becomes “useless”. It is important that, when traffic is not stationary, at the time of transition, the algorithm *learns* new heavy matchings quickly. To analyze this feature of LAURA, we study the transient behavior of LAURA for a case of high load ($\rho = 0.99$).

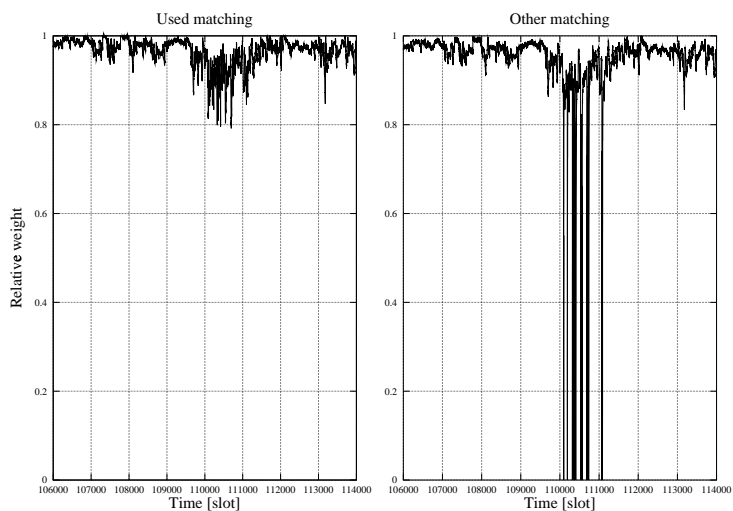


Figure 4.17. Relative weight of the two stored matchings in LAURA

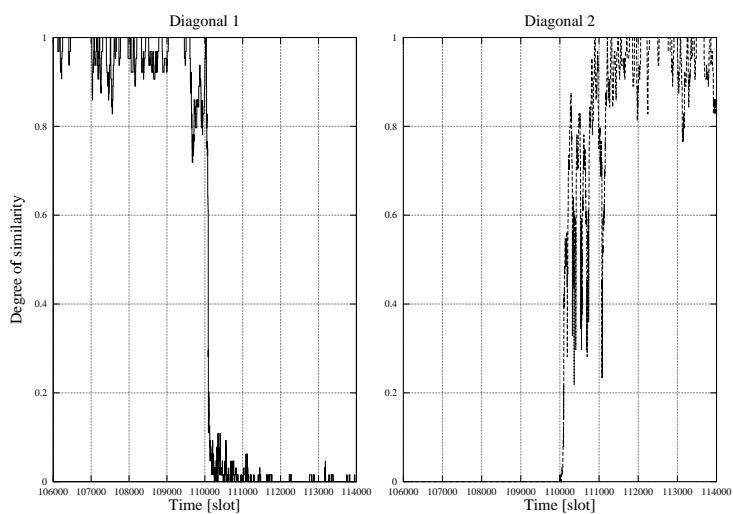


Figure 4.18. Degree of similarity of the two stored matchings in LAURA

To simulate this scenario, the switch receives packets according to the traffic matrix $\Gamma^{(1)}$ till time t_0 and then according to the traffic matrix $\Gamma^{(2)}$. In $\Gamma^{(1)}$, $\gamma_{ii}^{(1)} = \gamma_{i|i+1}^{(1)} = 1/2N$ are the only non-zero elements. In $\Gamma^{(2)}$, $\gamma_{i|i+2}^{(2)} = \gamma_{i|i+3}^{(2)} = 1/2N$

are the only non-zero elements. Intuitively, $\Gamma^{(1)}$ is the traffic pattern where only the first two diagonals are loaded, while in the case of $\Gamma^{(2)}$, the other two diagonals are loaded.

Call M_1 the *used matching*, i.e. $M_1 X_t = \max\{D_{1,t} X_t, D_{2,t} X_t\}$, and M_2 the *other matching*, i.e. $M_2 X_t = \min\{D_{1,t} X_t, D_{2,t} X_t\}$. Fig. 4.17 shows the *relative weight* (for its definition, refer to Section 3.3.3) values of LAURA during the interval of time $t \in [106000, 114000]$, with the transition time being $t_0 = 110000$. The first curve presents the relative weight of M_1 , i.e. $M_1 X_t / D_{X_t}^* X_t$, and the second curve the relative weight of M_2 , i.e. $M_2 X_t / D_{X_t}^* X_t$. Note that the relative weight of M_1 is close to 1 most of the time. Even during the transition, its higher than 0.8. This means that the weight of the matching used is always very close to the maximum weight matching and can react very quickly to the changes of the traffic in case of such non-stationary scenarios. In other words, LAURA is efficient even during the transients, and learns the MWM quickly.

As seen in Fig. 4.17, sometimes the curve corresponding to M_2 falls to zero, which looks like erratic behavior. But such behavior is expected because of the purge phase in the algorithm. When the distance between M_1 and M_2 is less than $d_{MIN} = 16$, M_2 is deleted and there is no matching as M_2 , and hence its weight becomes 0. But this remains only for one time, since M_2 quickly acquires heavy edges.

Fig. 4.18 shows the *degree of similarity* s between the two stored matchings, for the case of the above diagonal traffic scenario $\Gamma^{(1)}$ and $\Gamma^{(2)}$. Since the only loaded parts are the diagonals of the traffic matrix, we study the degree of similarity for M_1 and M_2 by comparing the fraction of diagonals retained by the two matchings. Let $D_{(k)} \in \mathcal{M}$ be a matching connecting input i to output $|i + k|$ (recall $|k| = k \bmod N$). Being $s(M_1, M_2)$ the number of common edges among matchings M_1 and M_2 , the degree of similarity is defined as:

$$s_1 = \frac{1}{N} \max\{s(M_1, D_{(0)}) + s(M_2, D_{(1)}), s(M_1, D_{(1)}) + s(M_2, D_{(0)})\}$$

$$s_2 = \frac{1}{N} \max\{s(M_1, D_{(2)}) + s(M_2, D_{(3)}), s(M_1, D_{(3)}) + s(M_2, D_{(2)})\}$$

that is, s_1 represents the similarity of the stored matching with the matching corresponding to the first two diagonals, whereas s_2 represents the similarity of the stored matching with the matching corresponding to the second two diagonals.

For $t < t_0$, $s_1 \approx 1$ and $s_2 = 0$, since the stored matchings have “learned” the only two possible efficient matchings. At time $t = 110887$, which means 887 time slots after t_0 , we observe $s_1 = 0$ and $s_2 = 1$ which indicates that the system has learned the two other matchings completely.

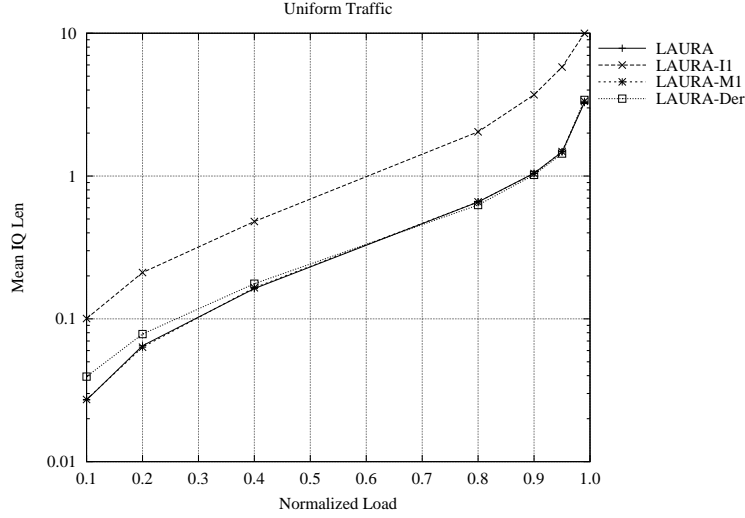


Figure 4.19. Mean IQ length for uniform traffic for the three variants of LAURA. All the algorithms considered show a similar delay, except for LAURA-II

4.7.3 Simplified version of LAURA

We explored the sensibility of LAURA to several parameters, in order to understand its robustness when its architecture is simplified in order to reduce its computational complexity. The following versions of LAURA have been studied by simulation:

- LAURA-II1: in this case $I = 1$, i.e. the number of iterations of RND-MATCH2 is one;
- LAURA-M1: in this case $S = 1$; it is represented in Fig. 4.8;
- LAURA-Der: it is a derandomized version of LAURA, described in Section 4.7.4.

The simulation results are reported in Figs. 4.19 and 4.20 respectively, the first for uniform traffic and the other for diagonal traffic.

All the versions considered of LAURA show a similar behavior for uniform traffic. For diagonal traffic all the versions are stable, although their delays can be different depending on the algorithm.

4.7.4 Derandomized version of LAURA

At each iteration of the random selection phase, a random permutation of the unmatched outputs is provided. In the derandomized version of LAURA a deterministic

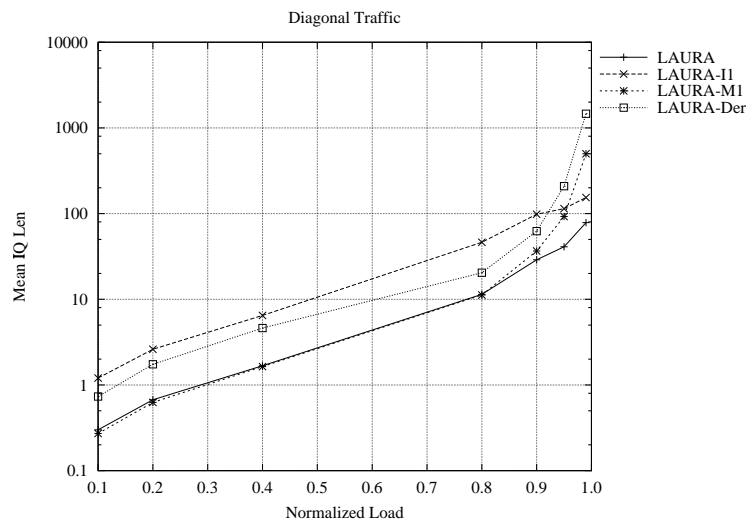


Figure 4.20. Mean IQ length for diagonal traffic. All the algorithms considered are stable also if they are simplified version of LAURA.

scheme is used to find the permutation of the unmatched outputs. Without loss of generality, consider the case when n inputs and outputs are left unmatched during iteration k . Let F be a counter which is incremented (modulo N) at each time slot. Let f be a counter which is set equal to F at the first iteration and incremented (modulo n) at the end of each iteration. During each iteration, input i is matched with output $j = |i + f|_n$. The two counters provide a round-robin mechanism to find the permutations among the unmatched outputs.

For diagonal traffic the derandomized version show worse performance than the randomized one. For diagonal traffic, the round robin mechanism is unfair, as can be understood in the following case. Consider N unmatched inputs/outputs, the input pointer set equal to i and the output pointer set equal to $|i + 2|_N$. In this case, for $N - 2$ updates of the output pointer, always the queue from input i to output i will be served. This is because of the simple policy adopted to update the pointers. If a more complex pointers update is adopted (e.g., setting the pointers next to the last served queue, like in iSLIP), the performance will improve. This topic will be further investigated in our future research work.

4.8 APSARA approach

As pointed out earlier, finding the maximum weight matching essentially involves a search procedure, which can take many iterations and be time-consuming. Since our goal is to design high-performance schedulers for high aggregate bandwidth switches, algorithms involving too many iterations are unattractive.

Here, our goal is to design a high-performance scheduler that only requires a *single* iteration. Therefore, we must devise a fast method for finding heavy matchings. One method for speeding up the scheduling process is to search the space of matchings in parallel. Fortunately, the space of matchings has a nice combinatorial structure which can be exploited to conduct efficient searches. In particular, it is possible to query the “neighbors” of the current matching in parallel and use the heaviest of these as the matching for the next time slot. This observation inspires APSARA (*A Parallel Scheduling Algorithm and its Random Approximation*) algorithm, which mainly uses the following two ideas:

1. learning scheme;
2. exploration in parallel of the neighbors (see Section 4.4.4) of the past matching.

4.8.1 APSARA-B: basic version

Let D_t be the matching determined by APSARA-B at time t and $\mathcal{N}(M_0)$ the set of neighbors of $M_0 \in \mathcal{M}$ with respect to a generator set given by *general swapping* (please refer to Section 4.4.4 for the definitions). Let $Z(t)$ the matching corresponding to the Hamiltonian walk at time t (refer to Section 4.4.2). APSARA-B works as follows:

- (i) $M_t = \text{MAX}(Z(t), \mathcal{N}(D_{t-1}); X_t)$.
- (ii) $D_t = \text{MAX}(D_{t-1}, M_t; X_t)$.

APSARA-B requires the computation of the weight of all the neighbor matchings. Each such computation is easy since a neighbor M' differs from the matching D_{t-1} in exactly two edges. However, computing the weights of all $\binom{N}{2}$ neighbors, if done in parallel as shown in Fig. 4.21, requires a lot of space in hardware for large values of N .

Say that hardware space constraints allow the use of at most $K \ll \binom{N}{2}$ modules, then how can the search procedure required by APSARA-B be conducted efficiently?

One obvious solution is to search the neighborhood set over multiple iterations by reusing the K modules. After all, at low line speeds there is more time for scheduling packets, allowing one to conduct more iterations. However, if line speeds are high and one is only allowed *one iteration*, then the question arises as to which K

neighbors should be chosen. A deterministic procedure for choosing the K neighbors will usually result in poor choices since, a priori, it is not clear which neighbors are heavy. It is better to choose K neighbors *at random* and use the heaviest of these. This motivates the following variants of APSARA, one based on a smaller generator set and the other based on a randomized approach.

APSARA generates all the matchings in the neighborhood set oblivious of the current queue-lengths. The queue-lengths are only used to select the heaviest matching from the neighborhood set. It is therefore possible that the matching determined by APSARA, while being heavy, is not of maximal size. Hence, a module of matching maximization can be added to the architecture. Max-APSARA uses MAX-DEP (refer to Section 4.1.1) to maximize the matching.

4.8.2 APSARA-L: linear version

Consider the set $\mathcal{N}'(D_{t-1})$ of all the N possible neighbors of D_{t-1} , given by adjacent swapping (refer to Section 4.4.4). In this case, only N modules are necessary to compute all the neighbors. APSARA-L works as follows:

- (i) $M_t = \text{MAX}(Z(t), \mathcal{N}'(D_{t-1}); X_t)$.
- (ii) $D_t = \text{MAX}(D_{t-1}, M_t; X_t)$.

4.8.3 APSARA-R: randomized version

Suppose hardware constraints only allows to query K neighbors. Let $\mathcal{N}_K(D_{t-1})$ denote the set of K elements picked uniformly at random from the set $\mathcal{N}(D_{t-1})$. APSARA-R determines the matching D_t as follows:

- (i) Determine $\mathcal{N}_K(D_{t-1})$ (note that it is not necessary to generate $\mathcal{N}(D_{t-1})$).
- (ii) $M_t = \text{MAX}(Z(t), \mathcal{N}_K(D_{t-1}); X_t)$.
- (iii) $D_t = \text{MAX}(D_{t-1}, M_t; X_t)$.

4.8.4 APSARA properties

We state now some significant properties of the algorithm.

Theorem 7. *Both APSARA-B and APSARA-R are stable under admissible Bernoulli i.i.d. inputs.*

Proof. Both versions use the Hamiltonian walk. Therefore, the learning approach is stable thanks to Theorem 5. \square

Theorem 8. Let D_t denote the matching obtained by APSARA at time t , and let $W^D(t) = D_t X_t$ denote its weight. If $D_t = D_{t-1}$, that is the matching of APSARA does not change from time $t - 1$ to time t , then

$$W^D(t) \geq \frac{1}{2}W^*(t),$$

where $W^*(t) = D_{X_t}^* X_t$.

Proof. Without loss of generality, assume that the maximum weight matching, $D_{X_t}^*$, at time t is the identity permutation; that is, input i is matched to output i under the maximum weight matching. Let the permutation corresponding to the matching D_t be π , i.e. D_t matches input i to output $\pi(i)$. Let w_{ij} denote the weight of VOQ_{ij} at time t . Consider any i , $1 \leq i \leq N$. Suppose $\pi(i) \neq i$. Let $\pi^{-1}(i)$ be the input matched to output i under D_t . Since $D_{t-1} = D_t$, from the property of APSARA, it follows that

$$w_{i\pi(i)} + w_{\pi^{-1}(i)i} \geq w_{ii}.$$

Note that above is true for all i , even if $\pi(i) = i$. Now summing over i , we obtain

$$\sum_i w_{i\pi(i)} + w_{\pi^{-1}(i)i} \geq \sum_i w_{ii}.$$

But, $\sum_i w_{i\pi(i)} = \sum_i w_{\pi^{-1}(i)i}$, since π is a permutation, and hence

$$\sum_i w_{i\pi(i)} \geq \frac{1}{2} \sum_i w_{ii}.$$

Now $\sum_i w_{i\pi(i)}$ is the weight of the APSARA matching and $\sum_i w_{ii}$ is the weight of the maximum weight matching. Thus, $W^D(t) \geq \frac{1}{2}W^*(t)$ and the theorem is proved. \square

4.8.5 Implementation

Both APSARA-B and APSARA-R involve an Hamiltonian walk. This was done for purely theoretical reasons: to ensure their stability (Theorem 7). We have found that, in practice, the Hamiltonian walk is not necessary; that is, both APSARA-B and APSARA-R provide virtually the same delay and throughput even without it. Thus, while the walk is extremely simple to implement, we do not consider it either in implementation or in performance evaluation¹.

The main feature of APSARA is that it can be implemented in a parallel architecture very efficiently. Fig. 4.21 shows a schematic for the implementation of APSARA with K modules.

¹Note that eliminating the Hamiltonian walk can only worsen the performance.

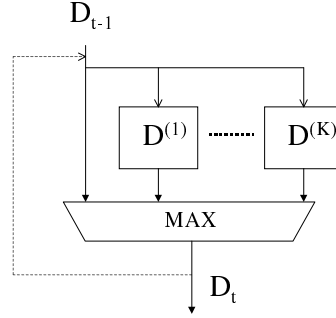


Figure 4.21. A schematic for the implementation of APSARA. The old matching, D_{t-1} , and the new arrivals, A_t , are used to compute the weights of neighbor matchings in parallel. The new matching, D_t , is computed by the weight-booster.

4.8.6 Comments about APSARA stability

APSARA without Hamiltonian walk cannot be shown to be stable, since its local search can be stuck in local maximums. For example, consider the following X_t , written as a matrix:

$$X_t = \begin{bmatrix} 51 & 100 & 0 \\ 0 & 51 & 100 \\ 100 & 0 & 51 \end{bmatrix}$$

Assume $D_{t_0} = (1,2,3)$ (written as permutation) and $A_t = (1,2,3)$ for $t > t_0$. Since all the neighbors of D_t have the same weight, we have $D_t = D_{t_0}$ for all $t > t_0$. ($D_t X_t = 153$, $D_{X_t}^* X_t = 300$ and for all neighbors D_t^N we have $D_t^N X_t = 151$). So, D_t cannot reach the MWM configuration, $T = \infty$ and Theorem 4 cannot be applied. In other words, D_t can be stuck forever because of a malicious adversary traffic. Note that in this case, Theorem 8 of Section 4.8.4 holds.

4.9 APSARA performance

In this section, we compare the performance of the following algorithms:

- APSARA-B, which is the basic version of APSARA, presented in Section 4.8.1;
- MaxAPSARA-B, which is the maximal version of APSARA-B, using the MAX-DEP module of Section 4.1.1;

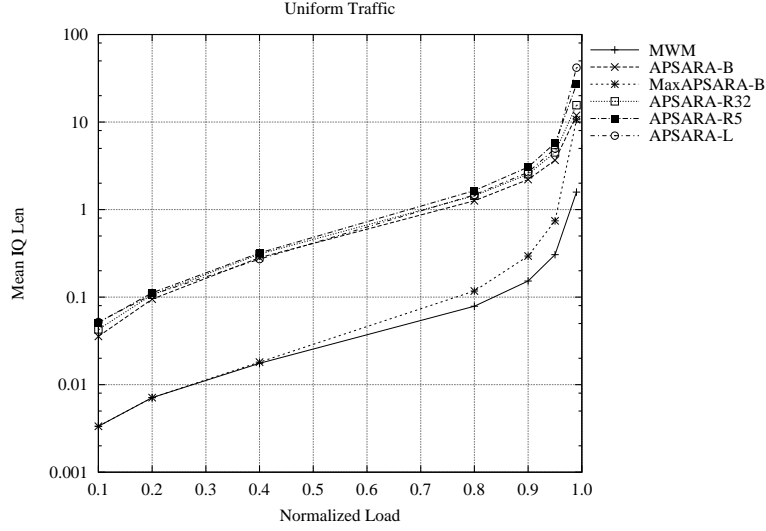


Figure 4.22. Mean input queue length for APSARA and its variants under uniform traffic

- APSARA-R32, which is the randomized version of APSARA, presented in Section 4.8.3, with $N = 32$ samples picked uniformly at random among all the neighbors of the past matching;
- APSARA-R5, which is the randomized version of APSARA, presented in Section 4.8.3, with $\log N = 5$ samples picked uniformly at random among all the neighbors of the past matching;
- APSARA-L, presented in Section 4.8.2.
- MWM, iLQF, iSLIP and RPA.

Fig. 4.22 compares the variants of APSARA for uniform traffic. All the algorithms are well-behaved, although their IQ length is larger than the MWM by about one order of magnitude. The reason of this is that all these variants are not maximal, in fact the Max-APSARA show queue lengths very close to MWM. Fig. 4.23 compares APSARA-B with all the other scheduling algorithms. Again, because it is not maximal, APSARA performs the worst.

Fig. 4.24 and 4.25 compare the performance of the algorithms considered for diagonal traffic. APSARA-B and its maximal version perform better than the other variants (note that for high load the matching maximization phase becomes useless).

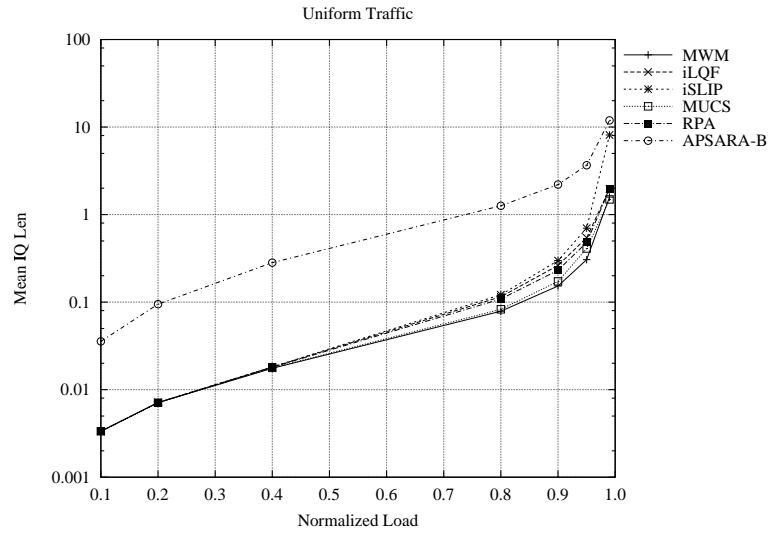


Figure 4.23. Mean input queue length for APSARA and other scheduling algorithms under uniform traffic

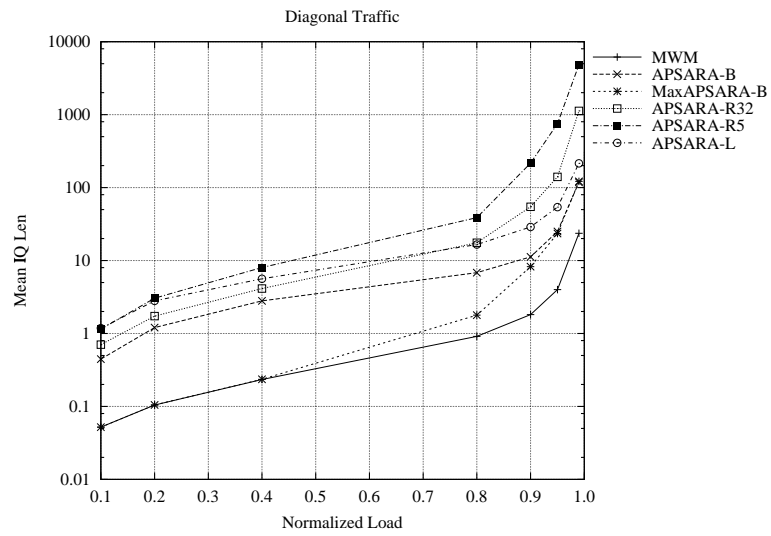


Figure 4.24. Mean input queue length for APSARA and its variants under diagonal traffic

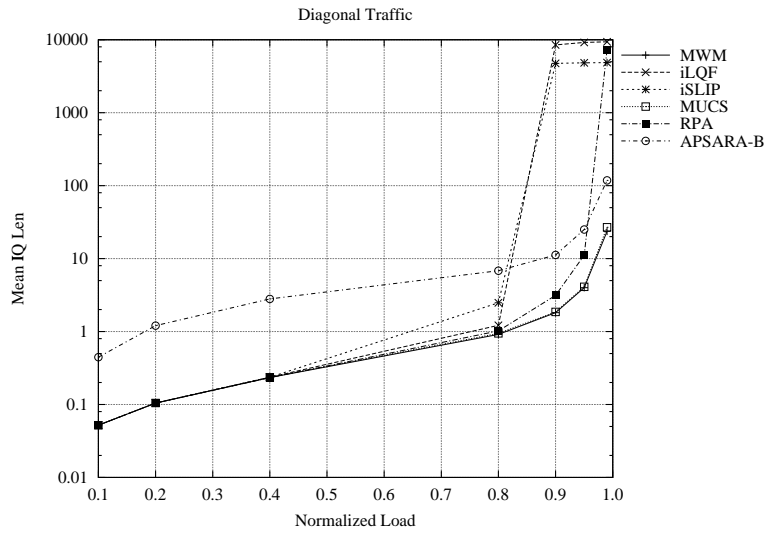


Figure 4.25. Mean input queue length for APSARA and other scheduling algorithms under diagonal traffic

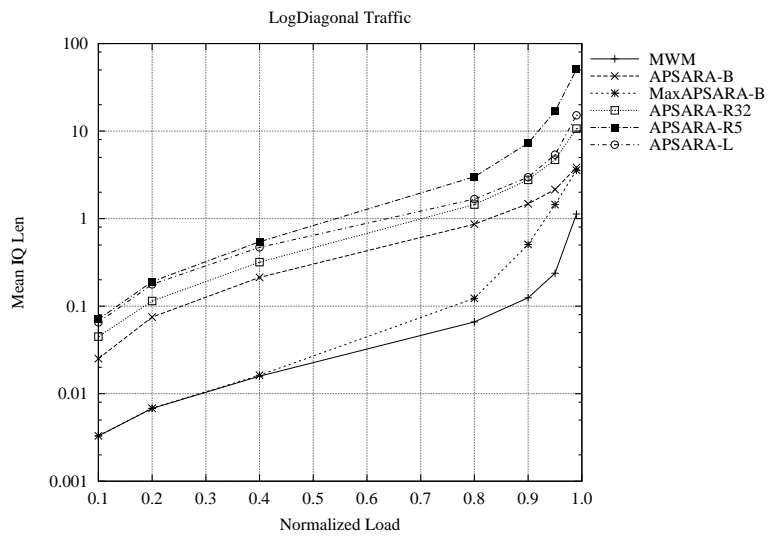


Figure 4.26. Mean input queue length for APSARA and its variants under logdiagonal traffic

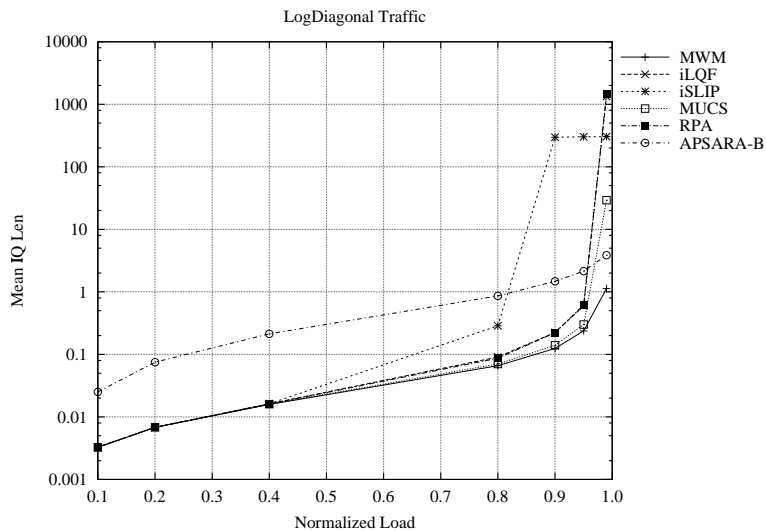


Figure 4.27. Mean input queue length for APSARA and other scheduling algorithms under logdiagonal traffic; note that iLQF behaves as RPA

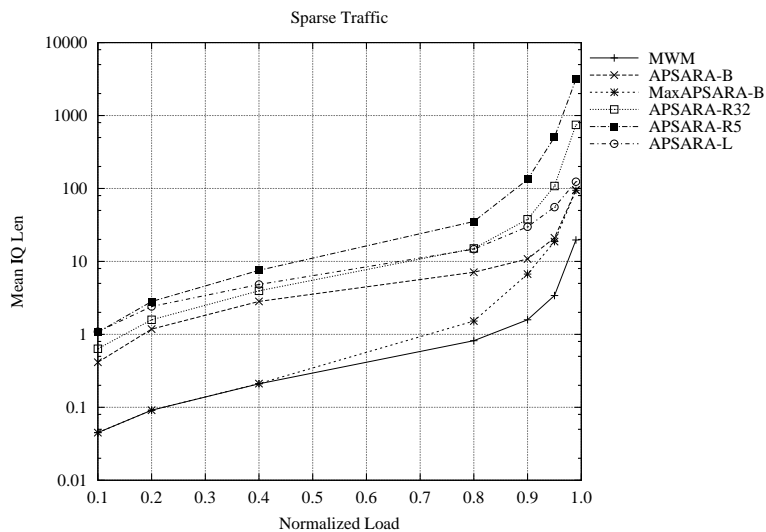


Figure 4.28. Mean input queue length for APSARA and its variants under sparse traffic

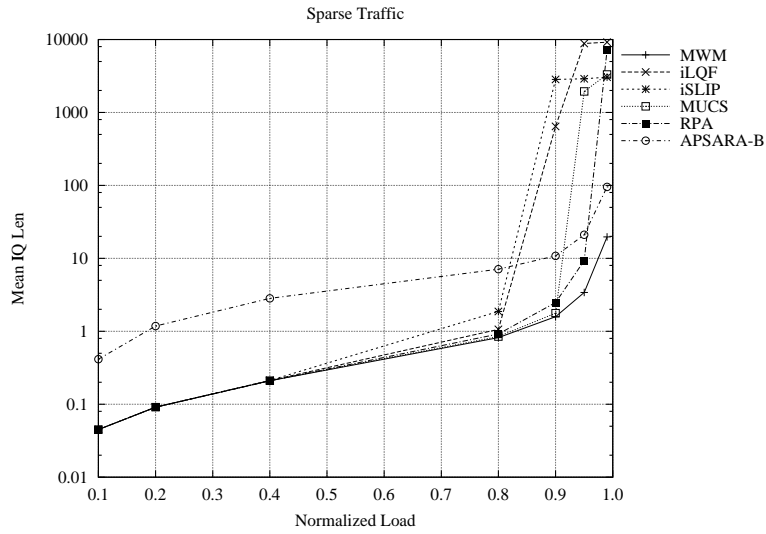


Figure 4.29. Mean input queue length for APSARA and other scheduling algorithms under sparse traffic

APSARA-L performs very close to APSARA-B because of the diagonal traffic pattern. For the two randomized variants, as the number of random samples decreases (from 32 to 5), the average queue length increases, but in both cases no losses are experienced. Fig. 4.25 is significant, because for diagonal traffic APSARA shows IQ lengths which are, at least, two orders of magnitude better than iSLIP, iLQF and MUCS. *All* the variants of APSARA outperforms all the other practical scheduling algorithms.

Under logdiagonal traffic, which is a more realistic traffic than diagonal one, APSARA continues to outperform all the other scheduling algorithms which approximate MWM. This fact can be drawn by observing Fig. 4.26 and 4.27.

The performance for sparse traffic can be inferred to be very similar to diagonal and logdiagonal traffic by observing Fig. 4.28 and 4.29.

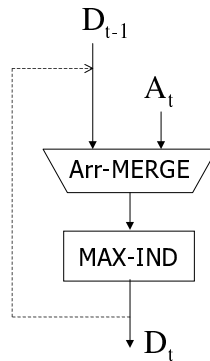


Figure 4.30. Basic architecture of SERENA

4.10 SERENA approach

Our final algorithm, SERENA (*Self Randomized algorithm Exploiting New Arrivals*) is based on the following ideas:

1. learning scheme;
2. exploitation of the randomness in arrivals.

One source of randomness available in switches is in the arrivals process. Using arrivals to find matchings also has the big benefit of providing information about recently loaded, and hence likely heavy, *VOQs*. (At least these *VOQs* will certainly be nonempty!) Note that the main drawback of observing arrivals is the possible temporary starvation of queues which are not currently receiving any packet. This can be a critical problem when the traffic is not i.i.d., but this drawback has not been considered in this study. Fig. 4.30 shows the basic architecture of the algorithm.

4.10.1 SERENA-B: basic version

Let D_t be the matching used by SERENA at time t and A_t be the arrival vector.

- (i) $\tilde{D}_t = \text{ARR-MERGE}(D_{t-1}, A_t; X_t)$.
- (ii) $D_t = \text{MAX-IND}(\tilde{D}_t)$

4.10.2 SERENA-S: simplified version

A variant of SERENA is based on removing some edges from the arrival vector A_t , such that A_t becomes a matching and the merging procedure is used. At time t , the algorithm works through these phases:

- (i) $\tilde{M}_t = \text{RND-ARRIVAL}(A_t; X_t)$ (defined in Section 4.4.3).
- (ii) $M_t = \text{MAX-IND}(\tilde{M}_t)$.
- (iii) $D_t = \text{MERGE}(D_{t-1}, M_t; X_t)$.

4.11 SERENA performance

In this section, we compare the performance of the following algorithms:

- SERENA-B, presented in Section 4.10.1;
- MaxSERENA-B, which is the maximal version of SERENA-B, using the MAX-DEP module of Section 4.1.1;
- SERENA-S, presented in Section 4.10.2;
- MWM, iLQF, iSLIP and RPA.

Fig. 4.31 compares the average queue lengths of SERENA-B and SERENA-S under uniform traffic: their performance are similar. Also in this case, the maximal version of the algorithm outperforms the non-maximal versions of SERENA, especially for low load. SERENA is compared with the other scheduling algorithms in Fig. 4.32. For uniform traffic, the basic version of SERENA performs the worst among all the algorithms, although within an order of magnitude in the queue lengths. Note that, since the SERENA is sensible to the arrivals, for low input load, its performance are quite competitive, since it is able to “track” at once new arrivals when the system is almost empty.

Figs. 4.33 and 4.34 show the average queue length for diagonal traffic. Note that SERENA performance, for both basic and simplified version, is very close to the MWM, especially for low loads and high loads. Also SERENA-S performs much better than the other scheduling algorithms, with average queue lengths (hence, delays) several orders of magnitude lower.

Under logdiagonal traffic (refer to Figs. 4.35 and 4.36) and under sparse traffic (refer to Figs. 4.37 and 4.38), SERENA behaves qualitatively as under diagonal traffic.

From these results, SERENA is shown to be an appealing approach, but further work needs to be done to understand the effect of temporary starvation of queues which are not receiving any arrivals for large intervals of times.

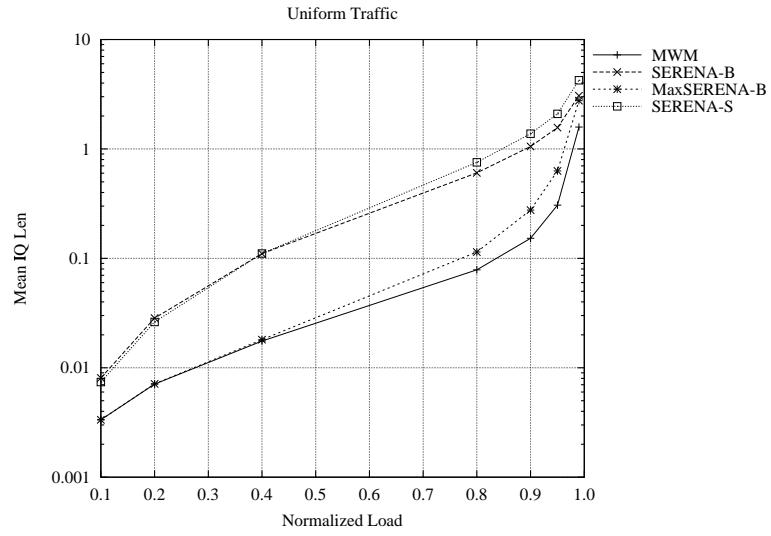


Figure 4.31. Mean input queue length for SERENA and its variants under uniform traffic

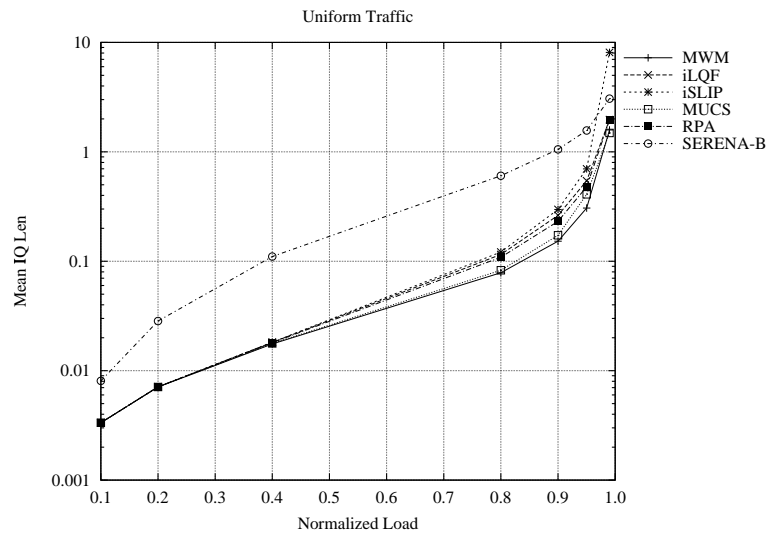


Figure 4.32. Mean input queue length for SERENA and other scheduling algorithms under uniform traffic

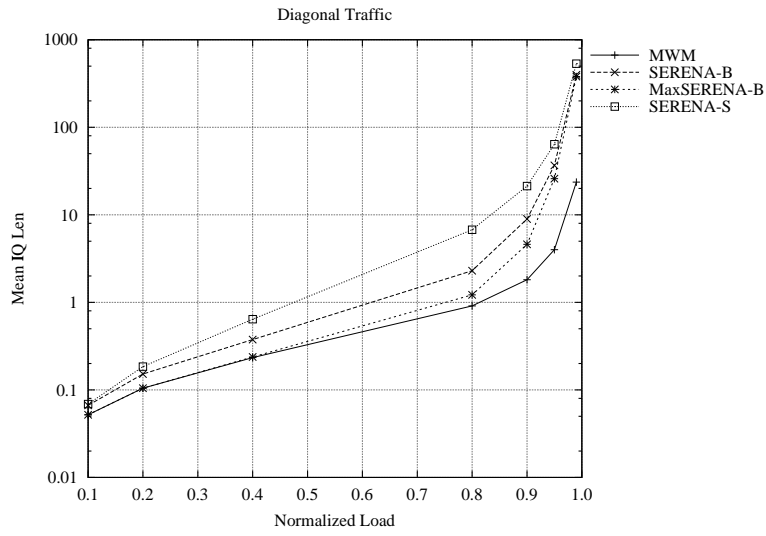


Figure 4.33. Mean input queue length for SERENA and its variants under diagonal traffic

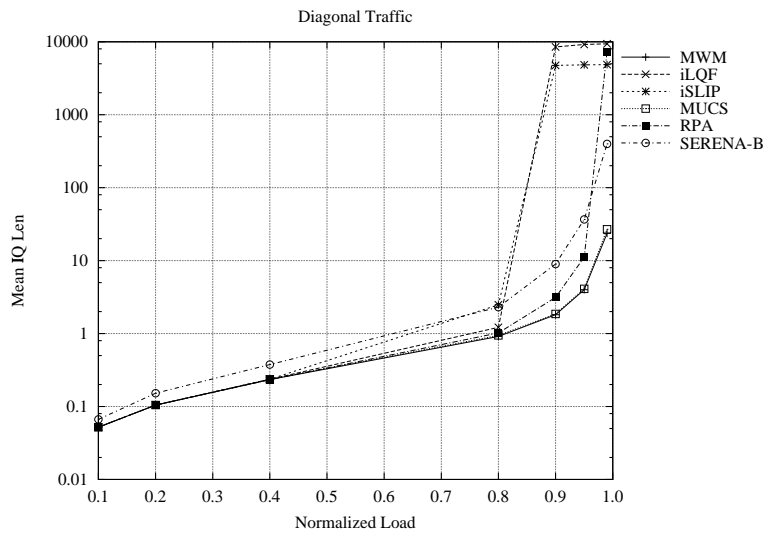


Figure 4.34. Mean input queue length for SERENA and other scheduling algorithms under diagonal traffic

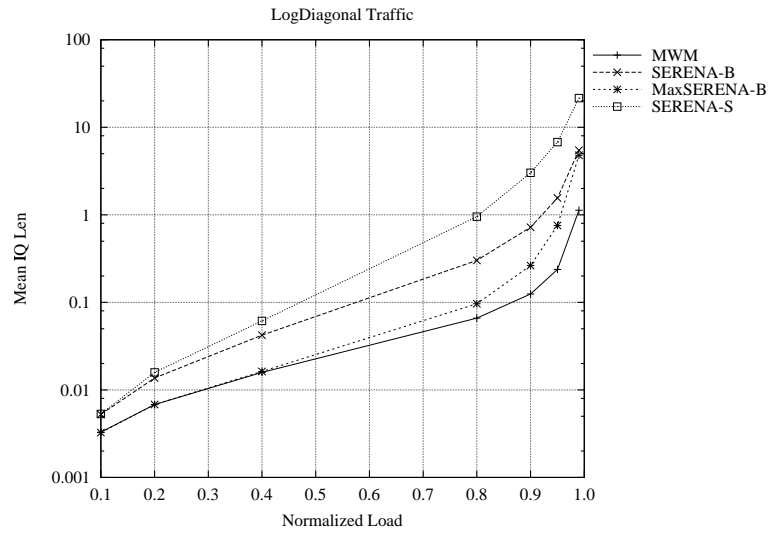


Figure 4.35. Mean input queue length for SERENA and its variants under logdiagonal traffic

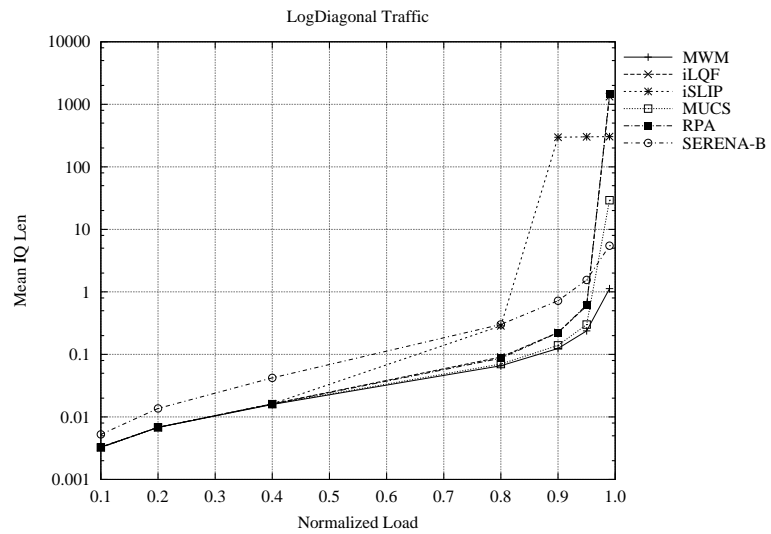


Figure 4.36. Mean input queue length for SERENA and other scheduling algorithms under logdiagonal traffic

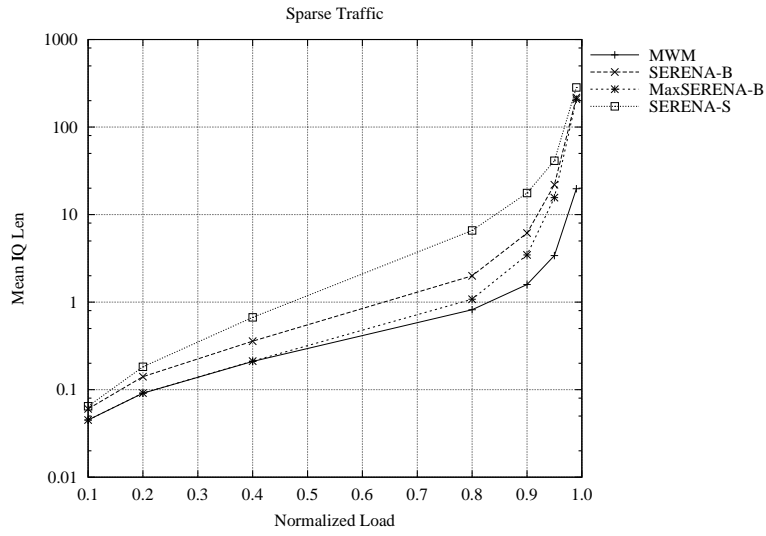


Figure 4.37. Mean input queue length for SERENA and its variants under sparse traffic

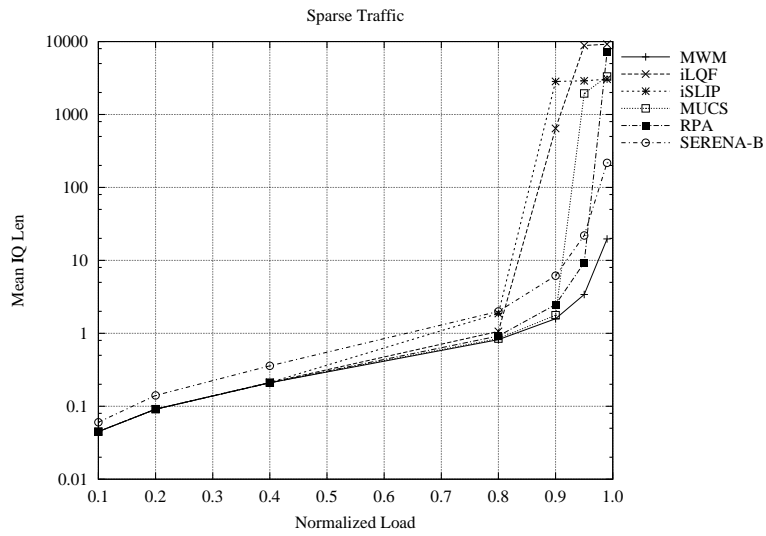


Figure 4.38. Mean input queue length for SERENA and other scheduling algorithms under sparse traffic

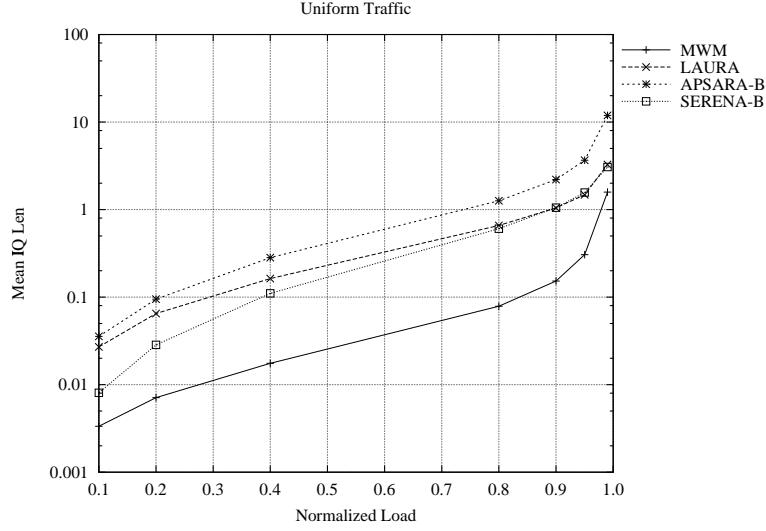


Figure 4.39. Mean input queue length for the three learning schemes under uniform traffic

4.12 Comparison among different learning schemes

We show here some simulation results to compare the three learning algorithms (LAURA, APSARA, SERENA).

Fig. 4.39 and Fig. 4.40 shows the average queue lengths under uniform and diagonal traffic for MWM, LAURA, APSARA-B and SERENA-B. For very low loads, SERENA-B outperforms the two other schemes since it is able to catch at once new arrivals and serve it. For high loads, under uniform traffic, SERENA-B behaves very close to LAURA, because the fact of exploiting the randomness in the arrivals, when the arrivals are uniform among all inputs and outputs, is the same of exploiting an external source of randomness like LAURA is using. Under diagonal traffic, for high loads, SERENA-B performs the worst since the arrivals happen with a high probability of being conflicting, hence the degree of freedom in choosing the probe-matching is lower.

We now discuss the performance for the three schemes for very large switch ($N = 1024$). We set all the parameters regarding to LAURA in conformity with the settings of Table 4.2. Here, Tables 4.3 and 4.4 show the average queue length, the average maximum queue length and the average delay for uniform and diagonal traffic, with an input load 0.99. Note that the performance of MWM could not be evaluated by simulation when $N = 1024$ since its computational complexity is too high. OQ refers

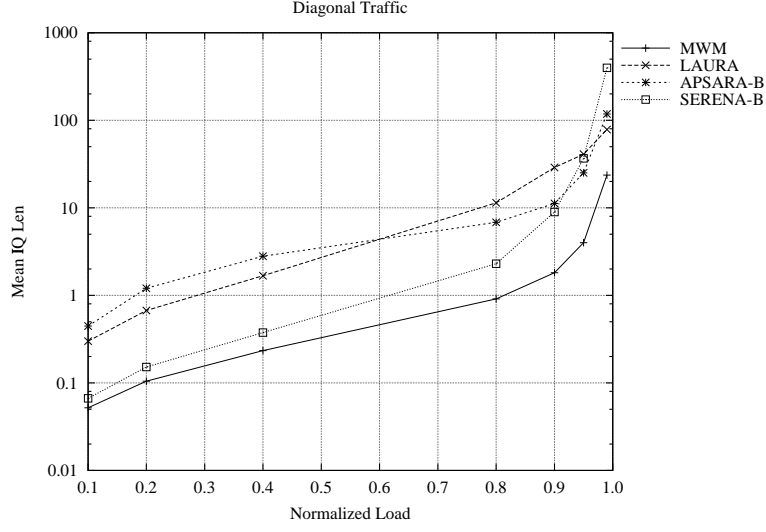


Figure 4.40. Mean input queue length for the three learning schemes under diagonal traffic

Parameter	Symbol	Value
switch size	N	1024
stored matchings	S	2
iterations	I	10
selection factor	η	0.5
minimal distance	d_{MIN}	512
buffer size	Q_{max}	10,000

Table 4.2. Simulation parameters for testing LAURA in large switches

Algorithm	Mean IQ Len	Max IQ Len	Mean Delay	Accuracy
LAURA	1.1	12.0	1185	< 0.1%
LAURA-Der	1.2	12.4	1284	< 0.2%
SERENA-B	2.0	7.9	2117	< 0.2%
APSARA-B	0.1	5.1	110	< 0.2%
MWM	-	-	-	-
OQ	49.95	-	50.45	theo

Table 4.3. Performance under uniform traffic for input load 0.99 and 1024 input ports

Algorithm	Mean IQ Len	Max IQ Len	Mean Delay	Accuracy
LAURA	658	1541	1330	< 0.3%
LAURA-Der	2502	4458	4981	< 0.1%
SERENA-B	444	583	898	< 0.3%
APSARA-S	2163	4084	4359	< 0.6%
MWM	-	-	-	-
OQ	22.77	-	23.00	theo

Table 4.4. Performance under diagonal traffic for input load 0.99 and 1024 input ports

to an output-queued architecture, for which it is possible to compute theoretically the average delay W and the average queue length X under uniform traffic and diagonal traffic, thanks to the two formulas (B.19) and (B.21) computed in Appendix B.

All the three learning schemes are stable for reasonable queue size, under both traffic scenarios. Under uniform traffic, APSARA-B shows a much better performance than the other schemes, comparable with an OQ, but this result is not surprising, since the number of explored neighbors is very large (about 1 millions) and its computational complexity is very large. For diagonal traffic, we were able to simulate only APSARA-S and its performance are almost the worst among all the other algorithms. For diagonal traffic, SERENA-B behaves well, since it is able to reduce the problem of scheduling to only the queues that are receiving arrivals.

The main conclusion from all the simulation results, presented in this chapter, is that the learning approach is quite robust to several traffic patterns and switch size; the difference in the performance among the three learning algorithms presented is not significant to say that one approach is better than the others. It is worth to mention that the learning approach is the only able to achieve performance comparable to MWM under any admissible i.i.d. Bernoulli traffic pattern, and its performance can be much better than the other scheduling algorithm so far presented in literature (like iSLIP, RPA, MUCS).

Of course, the choice of a scheduler to implement in a real router requires to understand the best compromise among the sustainable complexity of the systems (given by the available hardware technology) and the desired performance.

Algorithm	Generation of probe-matching	Weigh-booster	Scalability
LAURA	complex	complex	good
APSARA	simple	simple	bad
SERENA	simple	complex	good

Table 4.5. Design comparison among the three learning schemes considered

We can now refer to Table 4.5 to highlight the main difference among the algorithms. The judgments about their design is questionable, but they can be used as reference. Note that the three learning schemes discussed are only three examples of application of the learning approach. Several other schemes, satisfying the basic properties discussed in Section 4.1, can be envisaged, also combining ideas from different learning schemes. We think that the most reasonable partition of the class of learning schemes can be done on the basis of their scalability. A scalable scheduling algorithm (like LAURA and SERENA) fits the design issues of an access router or an enterprise router, with a very large number of ports. A non-scalable scheduling algorithm (like APSARA) suits very well the design of very fast switches with low number of ports, like backbone routers.

Chapter 5

Scheduling multicast traffic

In a pure (no speedup) IQ switch, the main issue concerning multicast traffic is the capability of transferring a multicast cell in one time slot from an input queue to possibly several output ports. To solve this problem, several switch architectures have been proposed [35], which are based either on internal copy networks or recirculating lines, so that multicast cells are replicated at inputs and treated like unicast cells, or on redundant switching paths, that allow the parallel transfer of multicast cells to their destinations.

From an architectural point of view, the availability within the switch of a switching fabric with *intrinsic multicast capabilities* is extremely important to reduce the cost of multicast traffic management. For example, switching fabrics implemented with a bus or a crossbar offer the possibility of transferring a cell from one input port to many output ports at no extra cost; the cell injected into the switching fabric by the input port can reach any number of output ports within one time slot. We consider in this chapter IQ and CIOQ cell-based switches whose internal fabrics have such intrinsic multicast capabilities.

The problem of scheduling multicast traffic in IQ switches was defined and modeled in [59], using a theory based on stochastic ordering and majorization. In that work, the optimal scheduling discipline is fully characterized for a switch with two and three input ports, based on a queueing structure with only one FIFO queue for each input. Larger switches were not considered, and no results about the maximum achievable throughput were provided.

Several theoretical studies [20, 38, 41, 52] have appeared, that investigate the maximum throughput achievable when arrivals of multicast cells are generated according to a Poisson process, and random services of input queues are assumed. Moreover, cells at the head of input queues are assumed to be served independently across the different inputs, as well as from slot to slot. These models show that the maximum normalized throughput for IQ switches under uniform multicast traffic is always less than one, and that it depends on the multicast traffic distribution.

In [11] the multicast scheduling problem was studied in its possible variants (see section 5.1), and its hardness was proved. Another problem investigated in [11] is the integration of unicast with multicast traffic, suggesting the transfer of unicast cells to their output ports when the multicast schedule leaves those idle, thus treating multicast traffic as a different class from unicast. A greedy heuristic to schedule multicast traffic and to integrate unicast and multicast traffic is provided; its performance was studied by simulation using homogeneous (uniform over both inputs and outputs) traffic patterns; this can be a limit of the analysis, since we will show that only the transfer of non-homogeneous multicast traffic patterns is critical in an IQ switch.

A well-established result in the field of switching is that a CIOQ switch with internal speedup equal to 2 can emulate an OQ architecture [22]. A common belief it is that, by emulating an OQ architecture in an CIOQ switch with speedup equal to 2, it is possible to transfer multicast traffic with no problem. However, this is not true. Unicast traffic has a very “nice” property: to approach saturation on all the N output ports of the switch, it is necessary to receive packets at all the N inputs. Thus, under unicast traffic, the instantaneous aggregate load of the switch is always less or equal to than the total admissible capacity of the switch. On the contrary, with multicast traffic, packets arriving at just one input can bring all the switch outputs close to saturation; this implies that when all the switch inputs are active, for some time periods the instantaneous aggregate switch load can be N times the total admissible capacity of the switch (consider for example the possibility of sequences of broadcast cells arriving at all inputs); in other words, multicast traffic, even if admissible, can temporarily “flood” the switch, and cannot be scheduled with the approach proposed in [22].

In [74] a speedup equal to 2 was proved to be sufficient to obtain the stability of a CIOQ switch, *provided* that the multicast flow satisfies some conditions, corresponding to the fact that the multicast traffic is well regulated, and cannot “flood” the switching fabric. This is a restrictive assumption that cannot be assumed to hold in general.

In this chapter we consider packet switches/routers where arriving unicast and multicast fixed- or variable-size packets are fragmented into fixed-size cells, and stored into buffers at input ports. Cells are transferred from input to output ports through a switching fabric with multicast capabilities, following a scheduling discipline that must avoid contention (no more than one cell for IQ, and k cells for CIOQ with speedup k , can be extracted from an input port in one time slot, and the same constraints applies to the number of cells that can be delivered to an output port in one time slot).

The main goals of this chapter are: i) to discuss the performance achievable by IQ and CIOQ switch schedulers supporting multicast traffic, under any admissible traffic pattern (i.e., under traffic patterns that neither overload inputs nor outputs), ii)

to define the optimal multicast traffic scheduling algorithm, and iii) to devise low-complexity multicast traffic scheduling algorithms yielding good performance. As a performance metric we focus on the maximum throughput achievable by an IQ switch, or, in other words, on the minimum speedup required in a CIOQ switch to achieve the same throughput of an OQ switch. Preliminary results were presented in [5, 6].

In Sections 5.1 and 5.2 we introduce the problem of scheduling multicast traffic in IQ and CIOQ switches. In Section 5.3 we propose an innovative queueing architecture useful for theoretical considerations and we define the optimal multicast scheduling discipline. From the definition of the optimal multicast scheduling discipline, the formal characterization of the sustainable multicast traffic region naturally follows; multicast traffic is named sustainable if it can be transferred through the switch with sufficiently large finite buffers. Then, in Section 3.4.1 we identify a class of “worst-case” traffic patterns, i.e. traffic patterns that lead to a minimization of the switch throughput, and, in Sections 5.5 and 5.6, we analytically prove that any scheduling algorithm leads to poor performance when IQ and CIOQ switches are loaded with this type of traffic. Finally, in Section 5.7, we define a simple greedy multicast scheduling algorithm with low complexity and good performance, and apply it in our simulation study. To ease a first reading of the chapter, most analytical derivations and theorem proofs were moved to the Appendixes.

The results presented in this chapter are quite relevant, since they show that IQ and CIOQ architectures loaded with general multicast traffic patterns are inferior to OQ architectures, contrary to the case of unicast traffic, for which IQ and OQ switches were proved to be equivalent [22].

5.1 Multicast traffic

In this section we introduce some basic definitions about multicast traffic scheduling, and we illustrate the switch architecture considered in this chapter.

Unless otherwise specified, we refer to switches with N input and N output ports, where all input and output lines run at the same data rate. The switching fabric is assumed to have intrinsic multicasting (and broadcasting) capabilities, i.e., the cost of transferring in a time slot a cell from one input to one or more outputs does not depend on the number of destinations.

The average amount of traffic at each input (output) is called the input (output) load, and is measured in cells per time slot. We normalize input (output) loads to line rates: a load equal to 1 means a fully utilized input (output) line (1 cell per time slot). The traffic at the input of a switch is said to be *admissible* if no input load is larger than 1, and no output load is larger than 1. An input traffic is said to be *sustainable* if it can be transferred through the switch. Note that, contrary to some previous works,

we define a load exactly equal to 1 to be admissible. Since admissibility is related to the growth of queues occupancies with traffic, called queue stability, including or excluding a load equal to 1 in the admissibility region depends on the definition of stability. We mainly refer to Definition 2 of Appendix A.1 in the chapter, also called rate stability, hence we include a load equal to 1 in the admissibility region.

Any multicast cell is characterized by its *fanout set*, i.e., by the set of switch output ports (destinations) to which the cell is directed. The cell *fanout* [41, 52] is defined as the number of different destinations of a multicast cell, i.e., the cardinality of the fanout set. We say that a cell has *fanout destination* j when output port j belongs to the fanout set of the cell. A unicast cell has fanout one, and its fanout destination is the only output port to which the cell is destined.

5.2 Scheduling disciplines

Since we consider switching fabrics with intrinsic multicast capabilities, the cost of the transfer of one multicast cell equals the cost of the transfer of one unicast cell. At each time slot, cells stored in input queues contend to access the switching fabric to reach output ports. The decision about which cells can be transferred is made by the switch scheduler, which implements a scheduling discipline. The fact that multicast cells have multiple destinations implies that some scheduling disciplines may elect to transfer in just one time slot the multicast cell to all destinations, while others may elect to transfer the cell in several time slots, reaching non-overlapping and exhaustive subsets of destinations. In the latter case, a *partial service* is adopted when a cell reaches a subset of its remaining destinations with its current transfer, whereas a *total service* is adopted when a cell reaches all its remaining destinations with its current transfer. In the case of partial service, the *residue* [41, 67] is defined as the set of fanout destinations that have not yet been reached after a multicast cell is transferred towards output ports. Note that, given the fanout set of multicast cells, the same overall residue cardinality is generated by any work-conserving scheduling discipline. Each scheduling discipline with partial service uses a specific way of distributing or concentrating the residue among all contending inputs.

The basic strategies normally used by the scheduling discipline are:

- *No fanout splitting* – Any multicast cell is transferred through the switching fabric once, only when all fanout destinations can be reached in the same time slot. If any of the fanout destinations cannot be reached because the multicast cell loses contention for an output port, the cell cannot be transferred, and it will contend again in a future time slot (normally the next one). This strategy is non-work-conserving, since it may happen that no cell is delivered to an available output because of contention. This discipline may favor cells with small fanout.

- *Fanout splitting* [20, 41, 67, 81] – Multicast cells may be delivered to output ports over a number of time slots. Only those fanout destinations that could not be reached in previous time slots are considered in the next time slot.

In [11] it was proved that the multicast scheduling problem is NP-hard, both with and without fanout splitting.

In the case of fanout splitting, every partial service causes an increase of the input load, leading to performance penalties. Indeed, when fanout splitting is considered, a cell is scheduled in an average number of time slots equal to α , with $\alpha \geq 1$. This fact increases the number of time slots necessary to schedule the cell transfers and thus either the input load must be lowered by the same factor α (α can be seen as a factor of bandwidth reduction) or a minimum internal speedup equal to α is required. This performance degradation due to excessive splitting, which generates additional load, was observed in [52]. If no fanout splitting is considered, the throughput can drop to very low values, since in this case another form of performance penalty is introduced, due to the non work-conserving service. A good scheduling discipline should therefore find the best compromise between these two major performance impairment.

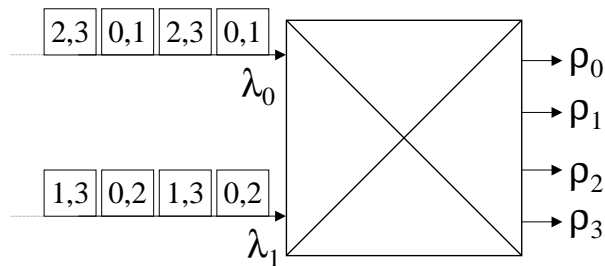


Figure 5.1. Traffic pattern leading to poor throughput with both fanout and no fanout splitting

The example shown in Figure 5.1 gives an indication of the problems that can arise with unfortunate but admissible traffic patterns. Inputs are fed with a sequence of back-to-back cells as shown in the figure, with alternating multicast destinations; numbers inside cells show their fanout destinations. This traffic is admissible; however, with multicast traffic, the constraints for traffic sustainability are well-defined only for OQ switch architectures: no input load and no output load must be larger than 1. In other words, any admissible traffic is sustainable. In the case of IQ switches, instead, while the best scheduling discipline for unicast traffic can sustain the same load of an OQ switch, it is easy to find a traffic pattern for which the best scheduling algorithm cannot reach the maximum throughput.

Using an OQ switch architecture, it is possible to sustain the considered traffic pattern since the load offered to each input and output is smaller than or equal to

one. Instead, with IQ switches and the best fanout splitting strategy, $3/2$ cells can be scheduled in each time slot in the whole switch. This means that, on average, for each input, $3/4$ cells are scheduled in each time slot, so that each cell requires $4/3$ time slots to be scheduled, and the bandwidth reduction factor is $\alpha \approx 1.33$. If a no fanout splitting discipline is considered, the throughput drops to $1/2$. Thus, with the setup shown in Figure 5.1, while an OQ switch can sustain the considered traffic pattern until $\lambda_0 \leq 1$ and $\lambda_1 \leq 1$, where λ_i is the average normalized load at input i , IQ switches must impose $\lambda_0 \leq 0.75$ and $\lambda_1 \leq 0.75$ when a fanout splitting discipline is considered and $\lambda_0 \leq 0.5$ and $\lambda_1 \leq 0.5$ for no fanout splitting disciplines. We consider in this chapter only scheduling disciplines with fanout splitting; indeed, scheduling disciplines with no fanout splitting imply that a total service is always adopted, whereas multicast cells can receive either total or partial service with fanout splitting scheduling discipline.

5.3 Optimal scheduling

In order to be able to define the optimal scheduling policy from the throughput point of view, we introduce a particular buffering scheme, which is an extension of the VOQ buffering architecture used for unicast traffic.

5.3.1 MC-VOQ queueing system

The choice of the queue structure obviously affects the scheduling discipline, since the cells examined in each time slot are always a (small) subset of all cells stored at input ports. On the other hand, the scheduling algorithm is tailored to the chosen queue architecture. Several different ways of organizing the input queue system can be envisaged; recall that under multicast traffic, buffer space may be used more efficiently by IQ switches than by OQ switches, in the sense that any multicast cell currently in the switch can be stored using only one buffer position.

In an IQ switch, when unicast cells are stored in just one FIFO queue per input port, the cell at the head of the queue can block the access to the switching fabric of subsequent cells, leading to the well-understood Head-of-the-Line (HoL) blocking effect [39], which limits the maximum throughput achievable by IQ switches. In the case of unicast traffic only, the usual approach to avoid HoL blocking consists of using at each input separate queues for each output (thus N queues per input, and N^2 queues overall); this queueing architecture is called *Virtual Output Queueing* (VOQ).

For multicast traffic, HoL blocking can be completely avoided using at each input separate FIFO queues for each one of the $(2^N - 1)$ possible fanout configurations. Note that each fanout configuration identifies what we call a multicast flow, the sequence of cells that arrive at one input port of the switch with identical fanout set. We

shall refer to this queue architecture with the name *MultiCast Virtual Output Queueing* (MC-VOQ). The (unicast and multicast) scheduling algorithm considers only the cells at the heads of the $2^N - 1$ FIFO input queues: only those cells may be scheduled for transfer to output ports. Hence, the considered scheduling algorithm is a *HoL scheduler*, i.e., it examines only cells at the head of each queue.

Note that our definition of HoL scheduler, although it assumes only FIFO queues, hence no queue look-ahead, does not prevent re-enqueueing an HoL packet to the tail of a different queue. Indeed, in the MC-VOQ architecture, when a multicast cell receives partial service, leaving a residue, we assume that that cell is dequeued from its current queue, and enqueued in the last position of the FIFO queue corresponding to the residue. For example, if a cell with fanout set $\{2,5,6\}$ can be transferred only to output 5 in the current time slot, it is dequeued from the queue corresponding to fanout set $\{2,5,6\}$ and queued to the FIFO storing packets for fanout set $\{2,6\}$. The above described approach can lead to out-of-sequence delivery of cells belonging to the same multicast flow.

Although the MC-VOQ architecture entails a very large number of queues, it is essential in the definition of the optimal multicast scheduling algorithm, since MC-VOQ allows the maximum possible throughput to be achieved. The use at each input of a set of FIFO queues is today standard in high-performance switches and routers: the per-flow and per-class queueing architectures are common examples of this type of queue architecture. The MC-VOQ architecture differs from these queue architectures only because of the need for $2^N - 1$ queues at each input; this may impair the switch feasibility, but the complexity of the queueing scheme is the price that has to be paid to completely eliminate HoL blocking. We consider also a per-flow queue architecture in this chapter, when defining a novel heuristic multicast scheduling algorithm, named Multicast Greedy, that provides good performance in several simulation scenarios.

5.3.2 Switch description

In this section we define the optimal scheduling discipline for an IQ switch with a MC-VOQ queueing architecture. Our methodology is based on the approach used in [9, 56, 62, 92]. We first introduce our notation and some useful relations. The queueing architecture is assumed to be MC-VOQ. Let I be the set of input ports, and O the set of output ports. Let N_Q be the total number of queues, $N_Q = N(2^N - 1)$.

We define X_n as the row vector of queue lengths (occupancies) at time n , i.e., $X_n = [X_{n,q_1}, X_{n,q_2}, \dots, X_{n,q_{N_Q}}]$, being $X_{n,q}$ the length of queue q .

Arrivals at switch inputs are described through a stochastic process A , which is a sequence of arrival vectors $A_n = [a_{n,q_1}, a_{n,q_2}, \dots, a_{n,q_{N_Q}}]$; $a_{n,q}$ is the number of cells arriving at queue q during time slot n . $D_n = [d_{n,q_1}, d_{n,q_2}, \dots, d_{n,q_{N_Q}}]$ is the departure

vector: $d_{n,q}$ is the number of cells leaving queue q during time slot n .

We assume $a_{n,q} \in \{0,1\}$ and $d_{n,q} \in \{-1,0,1\}$. The -1 value of $d_{n,q}$ is quite unusual, but it is necessary when, due to fanout splitting, only a part of the fanout set of a cell is switched, so that the residue must be buffered at the queue corresponding to the remaining fanout destinations, as discussed above. If a cell is partially transferred from queue q_i , and must be re-enqueued into queue q_j , then $d_{n,q_i} = 1$ and $d_{n,q_j} = -1$.

The queue length evolution is described by the relation

$$X_{n+1} = X_n + A_n - D_n \quad n \geq 0$$

We assume $X_0 = 0$.

Let w_i^I be a binary row vector of size $N(2^N - 1)$, i.e. $w_i^I = [w_{i,q_1}^I, \dots, w_{i,q_{N_Q}}^I]$, with $w_{i,q}^I = 1$ iff queue q stores cells from input i . Similarly to w_i^I , let $w_j^O = [w_{j,q_1}^O, \dots, w_{j,q_{N_Q}}^O]$, with $w_{j,q}^O = 1$ iff output j is included in the fanout set of the cells stored in queue q . Let $w_{ij}^{IO} = w_i^I \wedge w_j^O$, where \wedge denotes logical *and*; the element of the binary vector w_{ij}^{IO} corresponding to queue q is 1 iff q stores packets from input i and destined also to j . If V is a vector, let $u(V)$ be a vectorial operator which outputs a vector U (with the same size of V) whose k -th element u_k is equal to the step function applied to v_k , i.e., u_k is equal to 1 if $v_k > 0$, otherwise $u_k = 0$.

The technological constraints in the switching fabric are described by the following linear inequalities:

$$0 \leq w_i^I A_n^T \leq 1 \quad (5.1)$$

$$0 \leq w_i^I u(D_n^T) \leq 1 \quad (5.2)$$

$$0 \leq w_i^I u(-D_n^T) \leq 1 \quad (5.3)$$

$$0 \leq w_j^O u(D_n^T) \leq 1 \quad (5.4)$$

$$0 \leq w_{ij}^{IO} D_n^T \leq 1 \quad (5.5)$$

for $i \in I, j \in O$ and $n \geq 0$. Equation (5.1) means that at most one multicast cell can arrive at each input during a time slot; (5.2) means that at most one cell can be forwarded from each input; at most one cell can be re-enqueued into a queue, according to (5.3); (5.4) means that at most one cell is sent to an output at each time slot; by (5.5), each cell which receives a partial service is moved to a queue corresponding to a fanout set which is a subset of the initial one. Indeed, (5.5) states that three cases are possible: i) no multicast cell with destination j is transferred from input i , so that $w_{ij}^{IO} D_n^T = 0$; ii) one multicast cell is transferred from input i to output j , so that $w_{ij}^{IO} D_n^T = 1$; the cell transfer either reaches all destinations, or leaves a residue, but the latter does comprise output j ; iii) one multicast cell is transferred from input i to a subset of its destinations, which does not include output j , so that the cell must be re-enqueued at the input queue corresponding to its residue, and

$w_{ij}^{IO} D_n^T = 1 - 1 = 0$. By looking at all outputs of this cell, we see that re-enqueueing takes place only at a queue referring to outputs that were not reached, hence at a queue corresponding to a fanout set which is a subset of the initial one.

Table 5.1. Example of MC-VOQ queueing system in a switch with $N = 2$

Queue	Input	Fanout set
q_0	0	0
q_1	0	1
q_2	0	0,1
q_3	1	0
q_4	1	1
q_5	1	0,1

For example, in the case $N = 2$, with $I = O = \{0,1\}$ and the fanout set of the queues described by Table 5.1, we have:

$$\begin{aligned} w_1^I &= [1 \ 1 \ 1 \ 0 \ 0 \ 0], & w_2^I &= [0 \ 0 \ 0 \ 1 \ 1 \ 1], & w_1^O &= [1 \ 0 \ 1 \ 1 \ 0 \ 1], & w_2^O &= [0 \ 1 \ 1 \ 0 \ 1 \ 1] \\ w_{11}^{IO} &= [1 \ 0 \ 1 \ 0 \ 0 \ 0], & w_{12}^{IO} &= [0 \ 1 \ 1 \ 0 \ 0 \ 0], & w_{21}^{IO} &= [0 \ 0 \ 0 \ 1 \ 0 \ 1], & w_{22}^{IO} &= [0 \ 0 \ 0 \ 0 \ 1 \ 1]. \end{aligned}$$

5.3.3 Admissible region

The arrival vector at time n , A_n is a realization of a stochastic process characterized by the row vector Λ , whose elements are the average arrival rates at queues:

$$\Lambda \triangleq E[A_n]$$

We define input traffic to be admissible when neither input nor output ports are overloaded. With the notation that we introduced before, input ports are not overloaded if:

$$0 \leq w_i^I \Lambda^T \leq 1 \quad \forall i \in I \quad (5.6)$$

In a similar way, output ports are not overloaded if:

$$0 \leq w_j^O \Lambda^T \leq 1 \quad \forall j \in O \quad (5.7)$$

Note that (5.6) and (5.7) are necessary but not sufficient conditions for the rate stability of an IQ switch, while they guarantee the stability of an OQ switch.

The set of vectors Λ satisfying (5.6) and (5.7) forms the *admissible region* \mathcal{A} for input traffic.

5.3.4 Throughput definition

The switch throughput is usually measured at output ports. The instantaneous throughput $\rho_n(j)$ at output port j at time n , and the average throughput $\rho(j)$ at output port j are defined as follows:

$$\rho_n(j) \triangleq w_j^O u(D_n^T)$$

$$\rho(j) \triangleq E[\rho_n(j)]$$

When the fanout set of arriving cells is uniformly distributed over outputs, and traffic is admissible and uniformly distributed over input and output ports, if queue occupancies remain finite, the throughput can be computed as

$$\rho = \bar{F}\lambda$$

where \bar{F} is the average fanout of input cells, and λ is the cell arrival rate at each input port.

5.3.5 Optimal scheduling and capacity region

The scheduling of multicast traffic in an IQ switch can be formalized as a convex analysis problem.

Referring to the stochastic version of Lyapunov stability [9], the maximum throughput of the switch can be obtained, as demonstrated in Appendix A.1, by solving the following optimization problem at each time slot n :

$$D_n^* = \arg\{\max_{D_n}\{D_n X_n^T\}\} \quad (5.8)$$

subject to constraints (5.2), (5.3), (5.4) and (5.5). D_n^* defines the optimal scheduling choice at time slot n . We call *max-scalar discipline* the scheduling algorithm that selects D_n^* at each time slot n . The proof of the optimality of the max-scalar discipline is reported in Appendix A.1.

For a given X_n , the traffic patterns that satisfy the following relation guarantee that queue lengths remain bounded

$$\Lambda X_n^T \leq D_n^* X_n^T \quad (5.9)$$

Hence, such traffic patterns are sustainable. As shown in Appendix A.1, to satisfy (5.9), the arrival rate vector Λ must lie within the convex hull generated by all possible departure vectors $D^{(k)}$, i.e., the arrival rate vector Λ must be such that:

$$\Lambda = \sum_k \alpha_k D^{(k)} \quad (5.10)$$

$$\text{subject to } \begin{cases} \sum_k \alpha_k = 1 \\ 0 \leq \alpha_k \leq 1 \end{cases} \quad \forall k \quad (5.11)$$

The set of all arrival rate vectors Λ that satisfy the above expressions defines the *capacity region* \mathcal{C} for multicast traffic in an IQ switch. The capacity region contains all arrival rate vectors for which the queues of the IQ switch remain finite. The examples and the theorems in Section 5.5 prove that the capacity region is strictly internal to the admissible region. It is worth observing that for OQ switches the two regions coincide.

The implementation of the max-scalar scheduling discipline requires solving a linear programming problem at each time slot, but a numerical solution is extremely complex because of the huge number of possible departure vectors $D^{(k)}$, which corresponds to the number of possible switching configurations. In Appendix A.2 this number is computed, and results are reported in Table 5.2. Note the very fast growth of the problem complexity.

Table 5.2. Number of possible departure vectors in an IQ multicast switch

Number of ports	Number of switching configurations
2	19
4	$1.95 \cdot 10^5$
8	$3.42 \cdot 10^{25}$
16	$3.71 \cdot 10^{90}$

Starting from the definition of the admissible region \mathcal{A} and of the capacity region \mathcal{C} , we can formally define the minimum speedup to sustain all admissible traffic patterns in an IQ switch:

$$S_{min} = \arg \left\{ \inf_{1 \leq s \leq N} \left\{ \frac{1}{s} \Lambda \in \mathcal{C}, \forall \Lambda \in \mathcal{A}, \forall N \right\} \right\} \quad (5.12)$$

5.4 *K*-complex traffic definition

In this section we describe a multicast traffic pattern which proves to be critical for IQ switches; this traffic pattern will be used later in this chapter for the generation of numerical results characterizing the performance of IQ switches.

We introduce the following definitions:

- $O = \{o_k\}$ is the set of switch output ports; $|O| = N_o = N$
- $I = \{i_k\}$ is the set of switch input ports; $|I| = N_i = N$
- $A \subseteq I$ is the set of *active* switch input ports, i.e., input ports with cells waiting to be switched; $|A| = N_a$
- $R = \{r_l\}$ is the *request set* of (multicast) cells waiting to be switched at input queues
- $D_l = \{d_{lk}\}$ is the set of destinations of the l -th cell waiting to be switched
- $D_{lm} \subseteq D_l$ is the set of destinations of the l -th cell that are reached with the m -th transfer of the l -th cell through the switch (in the case of fanout splitting); $\cup_m D_{lm} = D_l$, and $D_{lm} \cap D_{ln} = \emptyset, \forall m \neq n$

We now describe a particular class of multicast traffic patterns, which is essential for our results.

Definition 14. A request set R is said **k -complex**, with¹ $k \in \mathbb{N}, k > 1$, if:

1. k cells are queued at each active input $a_l \in A$;
2. k cells are directed to each output $o_l \in O$;
3. for each sub-set of R comprising k cells, a destination exists to which all the cells in the subset are directed.

Table 5.3 reports an example of a 2-complex request set for a 6×6 switch, where only inputs i_0 and i_1 are active.

Table 5.3. Example of a 2-complex request set for a 6×6 switch with 2 active input ports

Input	Fanout sets	
i_0	$\{o_0, o_1, o_2\}$	$\{o_0, o_3, o_4\}$
i_1	$\{o_1, o_3, o_5\}$	$\{o_2, o_4, o_5\}$

In an $N \times N$ switch where N_a input ports are active, with $N = \binom{kN_a}{k}$ and $N_a \geq 2$, a k -complex request set R of size kN_a can be generated with the following algorithm:

¹In this chapter \mathbb{N} denotes the set of non negative integers, \mathbb{R} denotes the set of real numbers, and \mathbb{R}^+ denotes the set of non-negative real numbers.

- **Step 1.** Assign the first k cells in R to the first input, the second k cells to the second input, and so on, until the last k cells have been assigned to input N_a .
- **Step 2.** Form all the possible $N = \binom{kN_a}{k}$ different sub-sets of R whose size is k , and create an arbitrary injective correspondence from subsets of cells to destinations.
- **Step 3.** To each cell in R assign all the destinations that correspond to sets containing the cell itself.

For each cell, the resulting fanout is equal to $\binom{kN_a-1}{k-1} = \frac{N}{N_a}$.

Definition 15. A request set R' is called **generalized- k -complex**, with $k \in \mathbb{N}$, $k > 1$, if it contains kN_a cells independently extracted, possibly with repetitions, from a k -complex request set.

Note that a generalized- k -complex request set satisfies Condition 3 of Definition 14, but does not necessarily satisfy Conditions 1 and 2, since the number of cells queued at each active input and directed to each output can be different from k . Note also the conflicts between cells in a generalized- k -complex request set are not less than the conflicts in the corresponding k -complex request set.

Given a request set, we can define two types of traffic patterns:

- **stochastic** traffic patterns are obtained by offering at switch inputs requests randomly and equally likely extracted from the request set at a rate dependent on the desired load;
- **persistent** traffic patterns are obtained by periodically offering at switch inputs all the requests of the request set; the order at which requests are offered at the inputs must meet the following constraints:
 - at most one request is offered at any input in a time slot;
 - all the requests in the request set are offered at inputs in the minimum number of time slots, i.e., in a number of time slots equal to the maximum number of cells that arrive at any input or are destined to any output.

For example, if R is a k -complex request set, the traffic pattern obtained by repeating R at switch inputs every k time slots is called persistent k -complex traffic pattern.

It is important to observe that a persistent k -complex traffic pattern implies a load of input and output ports equal to 1.

5.5 Traffic sustainability conditions: preliminary results

In this section we present three original results concerning the sustainability region of an IQ switch with multicast traffic.

Theorem 9. *There exist admissible input multicast traffic patterns that lead to 100% throughput in OQ switches, under which IQ switches using a no fanout splitting policy provide a throughput that can drop to zero if the number of ports grows very large.*

Proof. We consider an (OQ or IQ) switch with N_a active input ports receiving a persistent 2-complex multicast traffic pattern.

An OQ switch architecture can transfer kN_a packets every $k = 2$ time slots, i.e., an average of N_a packets per time slot, and can thus sustain any admissible load; this is the case for the persistent traffic pattern that we are considering.

On the contrary, in an IQ switch using a no fanout splitting policy, under the same persistent 2-complex traffic pattern, at most one cell can be scheduled in each time slot. Thus, the throughput for IQ switches with no fanout splitting is $1/N_a$ of the throughput for OQ switches, and thus drops to zero when N_a grows indefinitely. \square

Theorem 10. *There exist admissible input multicast traffic patterns, under which the maximum sustainable throughput for an IQ switch using a fanout splitting policy is less than or equal to 0.5.*

Proof. To prove the theorem, we present a case in which the maximum load per input should be less than or equal to 0.5 for the IQ switch to be stable.

We consider a large size IQ switch loaded with a stochastic k -complex traffic pattern, in which only k input ports are active, and the offered load, measured as the average number of cells arriving at each input during each time slot, is λ . The effective service rate of each input queue, measured as the average number of packets transferred from each queue, is μ . It is necessary that $\lambda < \mu$ to guarantee the system stability, in the sense that all queue occupations are kept finite.

In each time slot, at most one multicast packet can be completely transferred, due to the properties of k -complex traffic (the complete transfer of two packets in the same time slot would lead to a conflict on the output port to which both packets are directed). As a consequence, the fanout of most packets must be split in at least two parts by any work-conserving scheduler. Thus, at each time slot, at most one packet can be completely transferred, and no more than $k - 1$ packets can be partially transferred. Hence, the maximum number of packets transferred in each time slot cannot exceed

$$T = 1 + \frac{k - 1}{2}$$

where the partial transfer of a packet whose fanout is split in two counts as half packet transfer, since two time slots are required to completely transfer the packet. The effective service rate per input port can then be written as:

$$\mu = T/k = \frac{1}{k} + \frac{k-1}{2k} = \frac{k+1}{2k}$$

Since $\mu > 0.5$ and $\lim_{k \rightarrow \infty} \mu = 0.5^+$, the stability condition requires that $\lambda \leq 0.5$. \square

Note that Theorem 10 can be referred also to switching architectures with internal speedup, i.e., to CIOQ architectures: the minimum speedup required to achieve 100% throughput under any admissible traffic pattern in CIOQ switches is not less than 2 for large switch size.

5.6 Traffic sustainability conditions: main results

In this section we state our main results, which concern the minimum speedup necessary to schedule a k -complex traffic pattern. We thus consider CIOQ switches, which provide an internal speedup, and first focus on the minimum number of slots required to transfer from the inputs to the outputs of a switch all the cells belonging to a request set. We call *time frame* a set of contiguous time slots. Theorem 16 of Appendix A.3 shows that, given a k -complex request set R with $|R| = kN_a$, for any finite integer $S \geq 2$, there exists an integer N_0 such that, $\forall N_a > N_0$, it is not possible to schedule all the cells in R with a frame length smaller than Sk .

This important result can be immediately applied to the general class of *frame-based* schedulers, but will also be later used to show properties of slot-by-slot schedulers.

In switches adopting a frame-based scheduler, a fixed number of time slots is grouped into a frame of fixed length, say L time slots, at both inputs and outputs. Assuming that the switch operates with speedup S (we assume S to be an integer, for the sake of simplicity), the scheduler works on an internal frame of length SL time slots (note however that, due to the internal speedup, the internal frame duration in time units coincides with the input and output frame duration). The scheduler allocates time slots of the internal frame for the transfer of multicast cells considering the state of input queues at the beginning of the frame. The scheduling constraints are that no more than one cell can be scheduled from any input and to any output during each time slot of the internal frame. The scheduling problem is in this case called *time slot assignment*, and it is formally defined in Appendix A.3. Frame-based schedulers can have interesting applications, since they may foster a simpler solution than slot-by-slot scheduler to scheduling traffic with strict QoS guarantees in IQ switches. It

can be shown that the set of all frame based schedulers is equivalent to the set of all slot-by-slot schedulers, provided that slot-by-slot schedulers introduce an additional delay equal to the frame length.

The following theorem is a rephrasing of Theorem 16 in Appendix A.3.

Theorem 11. *Consider a switch with N_a active inputs loaded by a persistent (or stochastic at load 1) k -complex traffic pattern, such that $|R| = kN_a$. For any finite S , there exists an integer N_0 such that $\forall N_a > N_0$ no frame-based scheduler can transfer all the cells at the input queues within one internal frame of size Sk .*

Theorem 11 is not a proof that a frame-based scheduler cannot achieve 100% throughput under a persistent k -complex traffic pattern in the case of finite frame size: the switch operates on a sequence of frames, and one may conjecture that a request that cannot be accommodated in a given frame can be transferred in subsequent frames at no extra cost, i.e., without increasing the duration of these subsequent frames. We conjecture that this is not the case, i.e., that 100% throughput cannot be achieved under a k -complex traffic pattern, but we were unable to obtain a formal proof for general schedulers. However, when k grows to infinity, Theorem 11 indeed proves that a frame-based scheduler cannot achieve 100% throughput under a k -complex traffic pattern. Moreover, in Theorem 12 we provide a proof of the above throughput limitation for a particular class of slot-by-slot schedulers.

Since the class of frame based schedulers is equivalent to the class of slot-by-slot schedulers (except possibly for delays), Theorem 11 also implies that IQ and CIOQ switches implementing a slot-by-slot HoL scheduling algorithm, when k grows to infinity, and, as a consequence, also the switch size grows to infinity, cannot achieve 100% throughput.

For finite size switches, Theorem 11 applies directly to periodic (CBR) traffic with bounded delay requirements: there exists a persistent traffic pattern that cannot be scheduled in large size IQ and CIOQ switches with any finite speedup, meeting the delay constraints. This means that the deployment of large size IQ or CIOQ switches can lead to performance degradations for real-time applications with QoS requirements.

Considering that a generalized- k -complex request set does not reduce conflicts between cells with respect to the corresponding k -complex request set, the following Corollary immediately follows from Theorem 11:

Corollary 6. *Consider a switch with N_a active inputs where a persistent generalized- k -complex traffic pattern is enqueued, such that $|R| = kN_a$. For any finite S , there exists an integer N_0 such that $\forall N_a > N_0$ it is not possible to schedule all the cells at input queues within an internal frame of size Sk .*

We can now go back to slot-by-slot HoL schedulers. Remember that we call multicast flow the sequence of cells that arrive at one input port of the switch with

identical fanout set. A *HoL flow-blocking scheduler* is an HoL slot-by-slot scheduler which does not allow interleaving of cells belonging to the same multicast flow; i.e., the (possibly partial) transfer of the i -th cell of flow f is enabled only after the transfer of all preceding $i - 1$ cells of the same flow f has been completed. This solution seems quite reasonable if in-sequence delivery of data must be guaranteed. Note that a FIFO HoL flow-blocking scheduler prevents cell re-queueing, which was instead assumed in our MC-VOQ architecture of Section 5.1.

We can now state our main result about slot-by-slot scheduling of multicast traffic.

Theorem 12. *Consider a switch with N_a active inputs loaded by a persistent k -complex traffic pattern, such that $|R| = kN_a$. For any finite speedup value S , there exists an integer N_0 such that $\forall N_a > N_0$ no HoL flow-blocking scheduler achieves 100% throughput.*

Proof. The theorem can be proved by contradiction; suppose that there exists a HoL flow-blocking scheduler that achieves 100% throughput with speedup S under a persistent k -complex traffic pattern. This implies that, on average, the transfer of $2kN_a$ cells is completed within two internal frames of total length $2kS$ time slots. Then, there must exist a pair of internal frames of length $2kS$ in which the transfer of at least $2kN_a$ cells is completed (see Lemma 11 in Appendix A.3). Since the scheduler is HoL flow-blocking, no more than kN_a cells may be simultaneously handled by the scheduler; thus, of the $2kN_a$ cells whose transfer is completed within a pair of internal frames of length $2kS$, at least kN_a have been completely transferred within the considered pair of internal frames (the first kN_a completions may refer to cells whose transfer may have started before the beginning of the considered pair of frames). However, since any set of kN_a cells belonging to a k -complex request set forms a generalized- k -complex request set, for Corollary 6, they cannot be completely scheduled in a frame whose length is less or equal to Sk . \square

While Theorem 11 provided only an asymptotic indication of the impossibility for slot-by-slot schedulers of achieving 100% throughput, Theorem 12 confirms this limitation for finite size switches on a restricted class of slot-by-slot schedulers. Note however that the restriction of a FIFO service of multicast cells belonging to the same flow seems quite reasonable for a wide range of applications, and in particular for real-time multimedia traffic streams, such as video, sound, and voice.

5.7 Greedy matching algorithm

In this section we propose a simple heuristic multicast scheduling algorithm, whose behavior approximates that of the optimal scheduling algorithm defined in Section 5.3, and we assess the performance of the heuristic algorithm by simulation.

The relevance of simple heuristic scheduling algorithms for multicast traffic is largely due to the fact that the optimal scheduler defined in Section 5.3 is way too complex for practical implementations, since it requires $(2^N - 1)$ queues at each input and a complex re-queueing algorithm of partially served (due to fanout splitting) multicast cells. Any partial service of a multicast cell implies the re-queueing of the same cell at the input queue corresponding to the cell residue. This does not guarantee in-sequence delivery of packets through the switch.

The proposed heuristic scheduling algorithm is based on a per-multicast-flow queueing architecture, where each queue is associated with a multicast flow, hence with one input port and a set of output ports, those toward which the multicast flow is directed. The proposed heuristic scheduling algorithm is an HoL flow-blocking scheduler, thus belongs to the class of schedulers considered in the previous section. When a multicast cell is partially served, it remains at the head of its queue, and will contend for the residual set of destinations in the next time slot, thus blocking the following cells of the same flow (HoL blocking thus exists within a multicast flow). In this case, in-sequence packet delivery through the switch is guaranteed.

5.7.1 Multicast scheduling

The heuristic multicast scheduling algorithm we propose is named *Multicast Greedy*. It uses as metric the number of cells in input queues.

The scheduler handles a list of multicast-flow queues, sorted by queue lengths, i.e., by the number of cells stored in each queue. At each time slot, all input and output ports are initially *unselected*. The scheduler orderly examines all queues of unselected input ports, starting from the longest queue. The cell at the head of the examined queue is scheduled for a (possibly partial) transfer from the corresponding unselected input port to all the unselected output ports belonging to its fanout set. The input port and output ports chosen in this step become *selected*. The algorithm iterates the above process, by orderly examining all queues of unselected input ports, until either all output ports are selected, or no more non-empty input port queues exist. All the scheduled cells can be transferred across the switching fabric to their destinations selected by the scheduler in one time slot, due to the intrinsic multicast capability of the switching fabric.

The scheduler tries to transfer each scheduled cell toward the largest possible number of output ports. If an output port is in the fanout set of the scheduled multicast cell and it is unmatched, it is selected as a matched destination in the current time slot, and it is subtracted from the fanout set of the considered cell.

The complexity of the algorithm depends on the maximum number of queued multicast flows, and on the efficiency in keeping an ordered structure among queues. Note that, at each time slot, at most $2N$ queues change their lengths by one unit (due to N arrivals and to N departures); this fact can be exploited to design efficient sorting

techniques. Moreover, at each time slot, the algorithm requires the inspection of only N^2 queues (the longest one for each input-output pair), independently of the number of active multicast flows.

5.8 Simulation study

5.8.1 Traffic patterns for simulation experiments

Table 5.4. Quasi-2-complex multicast request set for a 16×16 switch with $N_a = 4$

Input	Fanout sets	
i_0	$\{O_0, O_1, O_2, O_3\}$	$\{O_4, O_5, O_6, O_7\}$
i_0	$\{O_8, O_9, O_{10}, O_{11}\}$	$\{O_{12}, O_{13}, O_{14}, O_{15}\}$
i_1	$\{O_0, O_4, O_8, O_{12}\}$	$\{O_1, O_5, O_9, O_{13}\}$
i_1	$\{O_2, O_6, O_{10}, O_{14}\}$	$\{O_3, O_7, O_{11}, O_{15}\}$
i_2	$\{O_0, O_5, O_{10}, O_{15}\}$	$\{O_1, O_6, O_{11}, O_{12}\}$
i_2	$\{O_2, O_7, O_8, O_{13}\}$	$\{O_3, O_4, O_9, O_{14}\}$
i_3	$\{O_0, O_6, O_9, O_{12}\}$	$\{O_1, O_7, O_{10}, O_{13}\}$
i_3	$\{O_2, O_4, O_{11}, O_{14}\}$	$\{O_3, O_5, O_8, O_{15}\}$

We used stochastic traffic patterns in simulation experiments to assess the performance of the Multicast Greedy heuristic algorithm. These traffic patterns are described by a request set, comprising a list of multicast flows, each identified by an input port and a fanout set. Unicast flows are considered as a special case of multicast, with fanout equal to one. In general, only a subset A of the switch input ports can be active, i.e., originate at least a multicast flow, in a given traffic pattern.

We considered three request sets:

- *k-complex* request set, as described in Definition 14;
- *quasi-2-complex* request set: it refers to a 16×16 switch with $N_a = 4$, and the multicast flows are shown in Table 5.4; it is somehow similar to a 2-complex request set, but is adapted to a small switch size;
- *random* request set: the number of flows, and, for each flow, the input port, the fanout, and the fanout set are randomly and uniformly selected; more precisely, in our simulation experiments, the ratio between unicast and multicast traffic was randomly selected between 0.5 and 1.5, and the number of multicast cells at each active input was randomly selected between 0 and 4.

We used stochastic traffic patterns, hence, once the request set is defined, during a simulation experiment cells are randomly generated, choosing uniformly and independently among the flows comprised in the request set. The three considered stochastic traffic patterns are therefore:

- *k-complex* (\mathcal{Y}^k): this is one of the most critical traffic loads for the switch, and it can be defined only for very large switch sizes;
- *quasi-2-complex* (\mathcal{Y}^Q): this traffic pattern is similar to a *k-complex* traffic pattern with $k = 2$, but it refers to a small switch;
- *random* (\mathcal{Y}^R): contrary to the two previous traffic patterns, that were constructed so as to be difficult to schedule, this traffic pattern is considered as an example of a traffic that is much simpler to schedule.

During our simulation experiments, the load offered to the switch is chosen to be equal to the largest load which asymptotically avoids overloading the inputs: this load value was found with an heuristic searching process, running simulation experiments at different loads.

Simulation experiments were run until the confidence interval of the estimate of the average throughput for each destination reached with probability 0.95 a relative width smaller than 3%. The estimation of the confidence interval width is obtained with the batch means approach. Each queue is assumed to have finite capacity, equal to 1000 cells.

5.8.2 Simulation results

Table 5.5. Throughput for \mathcal{Y}^2 traffic

N_a	N	Max Throughput
2	6	0.679
4	28	0.588
8	120	0.521
16	496	0.479
32	2016	0.437
64	8128	0.405
128	32640	0.378
256	130816	0.355

Table 5.6. Throughput for \mathcal{Y}^k traffic and fixed $N_a = 8$

k	N	Max Throughput
2	120	0.521
3	2024	0.396
4	35960	0.334
5	658008	0.296

Table 5.5 shows throughput results for a 2-complex traffic pattern \mathcal{Y}^2 , for different switch sizes and numbers of active inputs N_a . The throughput can be observed to decrease as the switch size increases. For $N_a \geq 16$, throughput values are less than 0.5; this is an indication of the fact that, using our multicast greedy heuristic algorithm, a speedup larger than 2 is necessary to transfer a 2-complex multicast traffic pattern. Remember that OQ architectures can sustain any k -complex traffic pattern, but they require a speedup equal to the number of input/output ports N .

Table 5.6 reports throughput values for a fixed number of active input ports, $N_a = 8$, and variable k , for a k -complex traffic pattern \mathcal{Y}^k . For increasing k , the throughput can be seen to decrease considerably. This is again an indication of the fact that a significant speedup may be necessary to transfer k -complex traffic patterns.

With the quasi-2-complex traffic pattern \mathcal{Y}^Q , based upon the request set of Table 5.4, we obtained a throughput equal to 0.562, in the case $N = 16$, $N_a = 4$; this throughput value is lower than the one obtained for $N_a = 4$ and a 2-complex traffic pattern, as can be seen by Table 5.5. Although this result may seem strange, since the k -complex traffic pattern was defined as a worst-case scenario, an explanation is possible. Define an operator $\Gamma_d(\mathcal{Y})$ which gives the probability that d cells (randomly chosen from different inputs under traffic \mathcal{Y}) have at least one destination in common, i.e., that they are in conflict, and thus cannot be completely transferred in the same time slot. $\Gamma_d(\mathcal{Y})$ can be seen as a measure of the ‘difficulty’ of scheduling d cells in the same time slot, since when d cells are in conflict, their transfer must be split among at least d different time slots; $\Gamma_d(\mathcal{Y})$ is also a degree of ‘similarity’ to the d -complex traffic pattern. Note that, given the definition of a k -complex traffic pattern

$$\Gamma_d(\mathcal{Y}^k) = \begin{cases} 1 & \text{for } 1 < d \leq k \\ 0 & \text{for } d > k \end{cases}$$

For the quasi-2-complex traffic scenario, $\Gamma_2(\mathcal{Y}^Q) = 0.92$, $\Gamma_3(\mathcal{Y}^Q) = 0.25$ and $\Gamma_4(\mathcal{Y}^Q) = 0.063$. The fact that $\Gamma_2(\mathcal{Y}^Q) = 0.92$ would imply that \mathcal{Y}^Q is slightly less difficult to schedule than a 2-complex traffic pattern, but since $\Gamma_3(\mathcal{Y}^Q) = 0.25$, \mathcal{Y}^Q has a non negligible probability of behaving like a 3-complex traffic pattern, thus

being more difficult to schedule than \mathcal{Y}^2 . This is an indication of the fact that other multicast traffic patterns, different from the considered k -complex, exist, which are extremely difficult to schedule in IQ switches.

We also investigated by simulation the switch throughput for the traffic pattern \mathcal{Y}^R in a 16×16 switch, when all input ports are active, and the offered traffic is composed partly by unicast traffic and partly by multicast traffic; the ratio between the multicast and unicast offered load is a random variable uniformly distributed between 0.5 and 1.5. The unicast traffic is uniform across input and output ports; the multicast traffic is uniform across inputs, the fanout of every multicast cell is selected at random, and destinations are randomly chosen with uniform distribution. Over 200 random traffic patterns \mathcal{Y}^R were studied; for all of them, our heuristic greedy algorithm reaches at least 99% throughput at each output.

From the observation of simulation results it is possible to conclude that, for some multicast traffic patterns, IQ switches provide very good performance. This implies that the simple heuristic scheduling algorithm proposed in this chapter can be considered as a viable solution to schedule multicast traffic in IQ switches.

Note that, for the set of considered random traffic patterns, we obtained $0.0084 \leq I_2(\mathcal{Y}^r) \leq 0.0215$ and $0.0006 \leq I_3(\mathcal{Y}^r) \leq 0.0013$. The low degree of similarity to k -complex traffic shown by the randomly chosen traffic pattern is not a surprise and is an indication that the former of traffic is peculiar. However, the existence of traffic patterns that lead to very low throughput values in IQ switches remains a significant and important result.

Chapter 6

Scheduling variable-size packets

6.1 Input queued packet switches

The logical architecture for an IQ packet switch is shown in Fig. 6.1. At each input an Input Segmentation Module (ISM) segments the incoming packet into cells. PLS is the external packet line speed. ISMs operate in store-and-forward mode, are equipped with enough memory to store a maximum-size packet, and start the segmentation process only after the complete reception of the packet.

The cells resulting from the segmentation are transferred to the cell-switch input at a speed (called ILS) equal to the line speed PLS incremented to account for segmentation overheads. The capacity of each input queue at the cell-switch is limited to Q_{\max} , hence losses can occur. We assume that the entire packet is discarded if the input queue of the cell-switch does not have enough free space to store all the cells deriving from the segmentation of the packet *when the first of these cells hits the queue*. This is clearly a pessimistic assumption, but has the advantage of ease of implementation, and of avoiding the transmission of incomplete packet fractions through the switch.

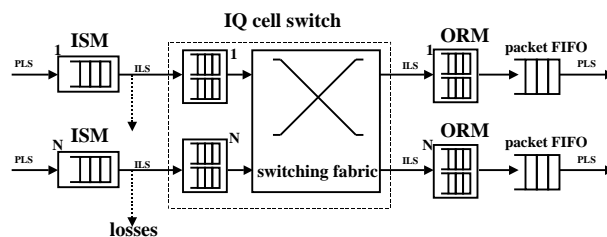


Figure 6.1. Logical architecture for an IQ packet switch

The cell-based switching fabric transfers cells from input to output queues, according to a scheduling algorithm. These cells are delivered to the Output Reassembly Module (ORM) at speed ILS. Here packets (i.e., IP datagrams) are reassembled. In general, cells belonging to different packets can be interleaved at the same output, hence more than one reassembly machine can be active in the same ORM. However, at most one cell reaches each ORM in a slot time, hence at most one packet is completed at each ORM in a slot.

Once a packet is complete, it is logically added to an output packet queue, called *packet FIFO* in the figure, from which packets are sequentially transmitted onto the output line. Note that the *packet FIFO* functionality is typically implemented by imposing a sequential transfer from the suitable ORM to the output line of all the cells belonging to the same reassembled packet. No internal speedup is required to support ISM and ORM, but for compensating internal overheads.

In the case of IQ packet switches using packet-mode scheduling, it is possible to further simplify the switch structure, and to improve its performance, by enforcing additional constraints on the scheduling algorithm. Indeed, cells belonging to the same packet are contiguous in the input queue of the internal IQ cell-switch. By using packet-mode scheduling (described in the following Section 6.1.1) cells belonging to the same packet are kept contiguous also in the output queue, and the ORM modules are no longer necessary (or at most one per output is used). In this case the logical architecture could be simplified by removing both the ORM module and the output packet FIFO from Fig. 6.1. Since each ORM operates in store-and-forward mode, hence introduces a delay equal to the packet size, the delays through the switch are reduced at least by a packet duration. We will not consider this possibility any further in the remainder of the chapter, since the removal of the packet FIFO can be achieved only if no format conversion is required to transmit the packet on the output link, a quite unrealistic hypothesis.

6.1.1 Packet-mode scheduling algorithms

Packet-mode scheduling algorithms introduce the additional constraint of keeping the cells belonging to the same packet contiguous also in output queues. To achieve this, the scheduling algorithm must enforce that, once the transfer through the switching fabric of the first cell of a packet has started towards the corresponding output port, no cells belonging to other packets can be transferred to that output, i.e., when an input is enabled to transmit the first cell of a packet comprising k cells, the input/output match must persist for the following $k - 1$ slots.

This is equivalent to having an infinite weight on the corresponding edge of graph G until all the cells belonging to the packet are transferred to the output port. Note that no conflicts can arise between infinitely-weighted connections, since no more than one cell can reach a given output in one slot, hence two connections directed to

the same output cannot simultaneously have an infinite weight.

We propose to extend the four considered scheduling algorithms to operate in packet-mode. The only complexity increase in the implementation is to add a boolean variable at each input to flag over-prioritized connections. Logically, maximal size matching algorithms must operate on ternary weights

$$w_{ij} = \begin{cases} 0 & \text{if } VOQ_{ij} \text{ is empty} \\ 1 & \text{if } VOQ_{ij} \text{ is non-empty} \\ \infty & \text{if a packet is being transferred from } VOQ_{ij} \end{cases}$$

6.1.2 Stability of packet-mode scheduling

Consider an IQ packet switch, and suppose that all input packet lengths are multiples of some unit length called UL (UL may correspond to a bit, a byte, or a cell). Consider the system of discrete-time queues comprising all input queues of the packet switch. The discrete time unit corresponds to a continuous time increment equivalent to UL. Let n be the discrete time variable.

We assume that customers correspond to cells to be transferred from input to output ports. Since we consider an IQ switch, each element d_n^i of the departure vector D_n , can only assume the values 0 and 1 $\forall i$ and $\forall n$. The arrival of a packet corresponds to the arrival of a group of customers, whose cardinality equals the packet length in UL units. Therefore, a_n^i can be larger than 1. However, if the traffic is admissible, $E[a_n^i] \leq 1, \forall i$.

Let $t_n \in \mathbb{N}^+$ be a non-defective sequence of regeneration instants (or stopping times) for the evolution of the system of queues, i.e., for any t_n , the evolution of the system following t_n is conditionally independent of the evolution of the system before t_n given the state $Y(t_n)$; moreover, $z_n = t_{n+1} - t_n$.

Definition 16. An IQ packet switch follows a **renewal MWM schedule** if at each stopping time t_n a new switching configuration is selected according to the outcome of a Maximum Weight Matching (MWM) algorithm whose weights are proportional to queue lengths, and the switching configuration is kept constant until t_{n+1} .

Definition 17. An IQ packet switch follows an **incremental MWM schedule** if at each stopping time t_n a new matching is selected according to the outcome of a MWM algorithm whose weights are proportional to queue lengths. Between two consecutive stopping times t_n and t_{n+1} , partial updates of the switching configuration are allowed. These reconfigurations are performed according to the outcome of a MWM algorithm whose weights are proportional to queue lengths, operating on a subset of input and output ports.

Lemma 5. An IQ packet switch following a renewal MWM schedule is stable under any admissible i.i.d. input traffic pattern A_n such that $E[A_n A_n^T] < \infty \forall n$.

Proof. The evolution of the system of discrete-time queues in the IQ packet switch is represented by a DTMC whose state is defined by the vector of queue lengths X_{t_n} ; between consecutive stopping times, the system evolution satisfies the following equation:

$$X_{t_{n+1}} = X_{t_n} + \sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i})$$

Note that all D_{t_n+i} , $i < z_n$ refer to the same matching; however, they need not be all equal, since some queue scheduled for transmission at time t_n may become empty before the next stopping time. If this happens, no packet can be transferred from empty queues.

By using the Lyapunov function $V(X_{t_n}) = X_{t_n} X_{t_n}^T$:

$$\begin{aligned} E[V(X_{t_{n+1}}) | X_{t_n}] - V(X_{t_n}) &= \\ &= E \left[2 \sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i}) X_{t_n}^T + \sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i}) \sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i})^T \right] \end{aligned}$$

Thus, under the assumption that $E[A_n A_n^T]$ is finite (which corresponds to assuming finite packet length variances), since also $E[D_{t_n+i} D_{t_n+i}^T]$ is finite:

$$\lim_{\|X_{t_n}\| \rightarrow \infty} \frac{E[V(X_{t_{n+1}}) | X_{t_n}] - V(X_{t_n})}{\|X_{t_n}\|} = \lim_{\|X_{t_n}\| \rightarrow \infty} \frac{2E[\sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i}) X_{t_n}^T]}{\|X_{t_n}\|}$$

Define now $D_\delta = \sum_{i=0}^{z_n-1} D_{t_n+i} - z_n D_{t_n}$; as noted before, this difference is due to the fact that some queues may become empty before changes in the switch configuration. Thus:

$$\frac{E[\sum_{i=0}^{z_n-1} (A_{t_n+i} - D_{t_n+i}) X_{t_n}^T]}{\|X_{t_n}\|} = \frac{E[\sum_{i=0}^{z_n-1} A_{t_n+i} X_{t_n}^T - z_n D_{t_n} X_{t_n}^T - D_\delta X_{t_n}^T]}{\|X_{t_n}\|}$$

Wald's Lemma [96, §2-13] can be applied, since t_n is a sequence of stopping times, therefore obtaining:

$$\frac{E[\sum_{i=0}^{z_n-1} A_{t_n+i} X_{t_n}^T - z_n D_{t_n} X_{t_n}^T - D_\delta X_{t_n}^T]}{\|X_{t_n}\|} = \frac{E[z_n](E[A_n] - D_{t_n}) X_{t_n}^T - E[D_\delta] X_{t_n}^T}{\|X_{t_n}\|}$$

Note that $E[D_\delta] X_{t_n}^T \geq -ME[z_n^2]$, since at most M components of D_δ can be non-null, no component of D_δ can exceed the value z_n , and, finally, a component of D_δ can be non-null only if the corresponding queue length at time t_n is smaller than

z_n . Moreover, for each admissible load and non-null queue length vector, $(E[A_n] - D_{t_n})X_{t_n}^T < 0$ as proved in [65]. Thus:

$$\begin{aligned} \lim_{\|X_n\| \rightarrow \infty} \frac{E[V(X_{t_{n+1}}) | X_{t_n}] - V(X_{t_n})}{\|X_{t_n}\|} &= \\ &= \lim_{\|X_n\| \rightarrow \infty} \frac{2E[z_n](E[A] - D_{t_n})X_{t_n}^T - 2E[D_\delta]X_{t_n}^T}{\|X_{t_n}\|} = \\ &= 2E[z_n] \lim_{\|X_n\| \rightarrow \infty} \frac{(E[A] - D_{t_n})X_{t_n}^T}{\|X_{t_n}\|} < -E[z_n]\epsilon \end{aligned}$$

□

Lemma 6. *An IQ packet switch following an incremental MWM schedule is stable under any admissible i.i.d. input traffic pattern A_n such that $E[A_n A_n^T] < \infty \forall n$.*

Proof. The proof can be easily obtained by applying the same Lyapunov function used in Lemma 5.

Consider an IQ packet switch with a given packet arrival process running an incremental MWM scheduler with stopping times $\{t_n\}$. A particular renewal MWM scheduler can be defined under the same arrival process and with the same set of stopping times. The latter scheduler is stable due to Lemma 5.

Since for the two schedulers we have the same set of stopping times, we get:

$$\sum_{i=0}^{z_n-1} D_{t_n+i}^I X_{t_n+i} \geq \sum_{i=0}^{z_n-1} D_{t_n+i} X_{t_n+i}$$

where $D_{t_n+i}^I$ is the departure vector at time t_n+i for the incremental MWM schedule, and D_{t_n+i} is the departure vector for the renewal MWM schedule. □

Definition 18. An IQ packet switch follows a **packet MWM schedule** if a new switching configuration is selected according to a MWM algorithm, whose weights are proportional to queue lengths, whenever either:

- all packet transmissions end at the same time, or
- all the queues selected for transfer become empty.

Definition 19. An IQ packet switch follows a **packet incremental MWM schedule** if:

- whenever either all packet transmissions end at the same time, or all the queues selected for transfer become empty, a new switching configuration is selected according to a MWM algorithm, whose weights are proportional to queue lengths, as in a packet MWM schedule;

- whenever some queues selected for transfer become idle (i.e., either they are empty, or packet transmissions end) a partial update of the switching configuration is allowed, according to an MWM algorithm among idle ports, whose weights are proportional to queue lengths.

Lemma 7. *Consider an IQ packet switch, following either a packet MWM schedule, or a packet incremental MWM schedule, whose input traffic is formed by variable length packets with i.i.d. random size. Packet sizes are expressed in integer multiples of UL. Assume that the average packet size is l and the packet size variance is σ^2 (both being finite). Assume that the transmission of packets from all queues selected by the MWM algorithm starts at the same time with exactly the same rate. Consider the sequence of instants t_n at which either the transmission of all the packets at the head of the selected queues ends at the same time, or all selected queues become empty. The sequence of stopping times t_n is non-defective, i.e., $z_n = t_{n+1} - t_n$ are such that $E[z_n] < \infty$ and $E[z_n^2] < \infty$.*

Proof. For simplicity, assume that the packet¹ length distributions at all queues are aperiodic, i.e., the maximum common divisor of all possible packet lengths expressed in UL is equal to 1. The proof can be easily extended to the case of periodicity. For simplicity we consider here a switch operating according to a packet MWM schedule, but the the proof can be easily extended for a switch operating according to a packet incremental MWM schedule.

We suppose that switch queues have infinite length, so that we neglect the probability that switch queues become empty near traffic saturation; thus, we obtain an overestimate of $E[z_n]$ and $E[z_n^2]$, since t_n are defined by only the sequence of instants in which transmission of all the packets at the head of the selected queues ends at the same time.

Each sequence of instants at which transmissions of packets end at queue k forms a discrete-time aperiodic renewal point process, thanks to the independence of packet lengths. Thus, for Blackwell's theorem [96, §2-19], the average number $E[f_n^k]$ of packets whose transmissions end at queue k at time n satisfies the following equation:

$$\lim_{n \rightarrow \infty} E[f_n^k] = \frac{1}{l} \quad (6.1)$$

However, no more than one packet transmission can end at each queue at each time (assuming no packet is of length zero); thus $E[f_n^k]$ equals the probability that a packet ends:

$$E[f_n^k] = \Pr\{\text{transmission ends at time } n \text{ and at queue } k\}$$

¹Note that the state definition in our analysis changes from number of cells to number of packets in the remainder of the chapter.

Limit (6.1) implies that, for any integer $m > 1$, there exists an instant n_k such that, $\forall n > n_k$:

$$\Pr\{\text{transmission ends at time } n \text{ and at queue } k\} > \frac{1}{ml} > 0$$

The probability that at instant n the transmission of packets at the head of all queues selected for transmission (be their number N_s) ends can be easily computed, since no correlation exists among queues behavior. Thus, given m , for $n > n_k, \forall k$:

$$\begin{aligned} \Pr\{\text{all tx end at time } n\} &= \prod_{k=1}^{N_s} \Pr\{\text{tx ends at time } n \text{ and at queue } k\} > \\ &> \prod_{k=1}^{N_s} \frac{1}{ml} = \frac{1}{(ml)^{N_s}} > 0 \end{aligned}$$

Consider now the sequence of instants t_n at which either all packet transmissions end, or selected queues become empty. The sequence t_n forms a renewal process; thus Blackwell's theorem applies:

$$\Pr\{\text{all tx end at time } n\} = E[f_n] = \frac{1}{E[z_n]}$$

where $E[f_n]$ is the average number of regenerations at time n ; since

$$\Pr\{\text{all tx end at time } n\} > 0$$

we obtain $E[z_n] < \infty$.

To prove that also $E[z_n^2] < \infty$, consider all packets transmitted from queue k between two subsequent regenerations; let W be the number of such packets, and L_j be their lengths expressed in UL. We can write:

$$\begin{aligned} E[z_n^2] - E^2[z_n] &= E \left[\left(\sum_{j=1}^W (L_j - E[L_j]) \right)^2 \right] = \\ &= E \left[\sum_{j=1}^W (L_j^2 - E^2[L_j]) \right] + E \left[\sum_{j=1}^W \sum_{\substack{i=1 \\ i \neq j}}^W (L_j L_i - E[L_j L_i]) \right] \end{aligned}$$

The second term in the sum can be easily shown to be null by conditioning on the value of W ; it can thus be eliminated. As a consequence:

$$E[z_n^2] - E^2[z_n] = E \left[\sum_{j=1}^W L_j^2 \right] - \sum_{j=1}^W E^2[L_j]$$

and by Wald's Lemma, since regeneration points are stopping times for the sequence L_j :

$$E[z_n^2] - E^2[z_n] = E[W]E[L^2] - E[W]E^2[L_j] = E[W]\sigma^2$$

Being $E[W]$ finite (otherwise $E[z_n]$ would be infinite), it results $E[z_n^2] < \infty$. \square

We can now state our main result.

Theorem 13. *Any IQ packet switch following either a packet MWM schedule or a packet incremental MWM schedule is strongly stable, provided that*

- *the input traffic is admissible*
- *the input traffic is formed by variable length packets with i.i.d. random size having finite average and variance*
- *the transmission of packets from all queues selected by the MWM algorithm starts at the same time with the same rate.*

Proof. The proof is quite straightforward from Lemma 5 (for packet MWM schedulers), or Lemma 6 (for packet incremental MWM schedulers), and Lemma 7, since the assumptions of Theorem 13 satisfy the conditions under which Lemma 7 holds. \square

Note that in Section 6.1.1 and in our simulation experiments we do not consider the packet MWM schedules of Theorem 13. We instead adopt a *constrained continuous heuristic matching*, i.e., in every slot, matching heuristics are used to approximate a MWM on the sub-graph of G in which busy inputs and outputs are removed; this is however quite similar to the packet incremental MWM schedule. Theorem 13 proves that the packet MWM algorithm guarantees stability, and shows at the same time that any matching algorithm that is not inferior to packet MWM also guarantees stability. While we could not prove that the considered constrained continuous heuristic matching is not inferior to packet MWM, the simulation results shown in Section 6.3 will not exhibit instabilities even for the constrained continuous heuristic matching approach, with the exception of iSLIP, which heuristically emulates MSM algorithms.

6.1.3 Delay of packet-mode scheduling

The relation between the average packet delay values in packet-mode and cell-mode schedulers can be understood with the help of a simplified queuing model.

We focus on one output port, and on packets directed from the different inputs to that output port, and consider only the packet delay component due to virtual output queuing (thus disregarding the effect of segmentation and reassembly modules). To estimate the packet delay with a queuing model, we also have to disregard the output conflicts in the switch; thus, our estimates will be reasonably accurate only for low to medium traffic loads. The transfer toward the output port corresponds to the packet service, and packets are modeled as customers requiring a variable amount of service. As a further simplification, we describe the packet arrival process at the switch ingress with a Poisson process, without taking care of overlapping ingress times.

This simplified setting corresponds to an M/G/1 queue, where cell-mode scheduling can be paralleled to processor-sharing (PS) or round-robin (RR) service, since all packets directed to the considered output are simultaneously served (again because of low traffic), and packet-mode scheduling can be paralleled to FIFO service, since each packet is served separately, with no interleaving of cells of different packets.

We know from queueing theory that the average delay $E[D_{PS}]$, in the case of PS service, is:

$$E[D_{PS}] = \frac{\rho E[S]}{1 - \rho}$$

where $E[S]$ is the average service time and ρ is the queue traffic (or utilization factor). Instead, for an M/G/1 queue with FIFO service, we know that:

$$E[D_{FIFO}] = \frac{\rho E[S]}{1 - \rho} \times \frac{1 + C_v^2}{2}$$

where C_v is the coefficient of variation of the service time.

Note that $E[D_{PS}]$ is equal to the average delay in an M/M/1 queue with FIFO service, so that $E[D_{PS}] = E[D_{FIFO}]$ for negative exponential distribution of packet lengths (that is for $C_v = 1$).

We define the *packet-mode gain*, denoted G , as the ratio between the average packet delay experienced with cell-mode scheduling and the average packet delay experienced with packet-mode scheduling. As noted before, in our simplified analysis these average packet delays refer only to the waiting time in input queues, not comprising the delays due to segmentation and reassembly. From the simplified queuing model, G can be estimated as:

$$G = \frac{2}{1 + C_v^2}$$

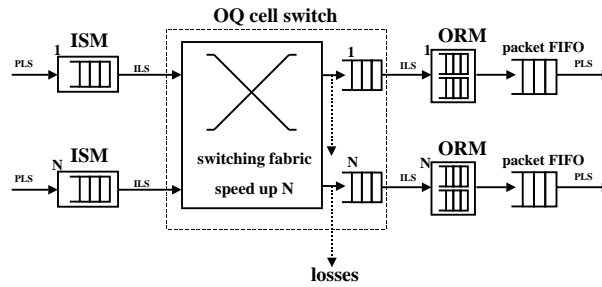


Figure 6.2. Logical architecture for a FIFO-OQ packet switch

Thus, the analytical model predicts packet delay gains for packet-mode scheduling in the case of packet length distributions with small variance ($C_v < 1$), whereas cell-mode scheduling is expected to provide lower packet delays when the packet length variance is large ($C_v > 1$).

Note that similar results could have been obtained by modeling cell-mode operation with group arrivals, and by interpreting individual customers as cells, and groups as packets. Packet delays would in this case be equivalent to group delays.

6.2 Output queued packet switches

Fig. 6.2 shows the logical architecture of a FIFO-OQ packet switch. Packets arrive at input ports where they are segmented by ISM modules, similarly to what happens in IQ routers. The cells obtained with the segmentation are sent to the cell-switch at speed ILS, and are immediately transferred to the output queues of the cell-switch, thanks to a speedup equal to N in the switching fabric. Losses may occur at output queues, whose capacity is limited to $N \times Q_{\max}$ for each queue.

From the output queues of the cell-switch, cells are delivered at speed ILS to an ORM module for reassembly. Once a packet is completed, it is queued to a packet FIFO queue similar to what was seen for the IQ case.

In a FIFO-OQ packet switch cells belonging to different packets can be interleaved in output queues. When a cell at the output of the switching fabric does not find room in the output queue of the cell-switch, it must be discarded. The other cells belonging to the same packet of the discarded cell may be in the output queue, or already in the ORM. We assume to be unable to identify and discard the other cells in the output queue: those cells will be discarded by the ORM module, which has knowledge of the existence of the packet. This means that cells belonging to packets that suffered partial losses unnecessarily use system resources.

The IQ architecture has the advantage that all the cells belonging to the same packet are contiguous in input queues, where losses occur. In the FIFO-OQ case, losses occur in queues where cells of different packets are interleaved.

To cope with this problem, although it would be possible to identify other discard policies, we consider an alternative architecture named VIQ-OQ. Referring to Fig. 6.2, a VIQ-OQ packet switch can be obtained by dropping the FIFO queues at the output of the OQ cell switch and by moving the ORM machines internally to the OQ cell switch. The ORM queues take the role of Virtual Input Queues: at each output, a different queue is available to store packets received from different inputs. This modification reduces the delay in the OQ architecture; moreover, cell losses are not spread through several packets as typically occurs in FIFO-OQ switch.

6.3 Simulation results

To evaluate the performance of IQ switching architectures by comparison with OQ architectures and to validate our analytical models, we conducted a number of simulation experiments. We only report here selected results for packet switches with equal numbers (N) of input/output interfaces, assuming that all input/output line rates are equal, and that only unicast traffic flows are present.

We assumed finite queue sizes in the simulation models. For IQ switches, each input queue VOQ_{ij} has finite length Q_{\max} ; when a cell directed to output j arrives at input i , and queue VOQ_{ij} is full, the cell is lost. No buffer sharing among queues at the same input port is allowed. For FIFO-OQ switches, each FIFO queue has finite length equal to $N \times Q_{\max}$, for VIQ-OQ switches each queue has finite length equal to Q_{\max} and no buffer sharing among queues at the same output port is allowed.

We are interested in analyzing the performance of packet-mode scheduling with respect to cell-mode scheduling in IQ architectures more than in comparing IQ with OQ architectures, which are considered and plotted as a reference.

6.3.1 Traffic scenarios

Our simulation models do not explicitly describe the arrival of IP datagrams at the packet-switch inputs. We instead model the arrival of cell bursts at the inputs of the internal cell-switch. These cell bursts originate from the segmentation of a packet.

The cell arrival process at input i , $A_k^{(i)}$, is characterized with a two-state ON-OFF model. When the input port is in the ON state, a packet is being received. The number of slots spent in the ON state, i.e., the size in cells of the packet, is a discrete random variable Φ_{ij} for packets directed from input i to output j . No cells are received in the OFF state. The number of slots spent in the OFF state is geometrically distributed

with average $E_{\text{OFF}} = (1 - p)/p$. The parameter p is set so as to achieve the desired input load.

We consider the following three traffic scenarios, which are described through their corresponding Γ (defined in Section 3.1) and Γ_P matrices. Γ_P derives from Γ , through all the Φ_{ij} random variables. Let us start to define the average *packet arrival rate*, denoted by λ_{ij}^P . If l_{ij} is the average packet size (in cells) for VOQ_{ij} , $\lambda_{ij} = l_{ij}\lambda_{ij}^P$. Similarly to γ_{ij} , it is normalized γ_{ij}^P and then defined the *normalized packet arrival rate matrix* $\Gamma_P = [\gamma_{ij}^P]$.

Uniform packet scenario. In this case $\gamma^{(ij)} = \gamma_P^{(ij)} = 1/N^2, \forall i, j$ (uniform traffic).

Packet lengths are uniformly distributed with lengths ranging between 1 and 192 cells. The maximum packet size comes from the Maximum Transmission Unit (MTU) of IP over ATM, which is equal to 9188 octets (9140 octets of user payload, and 20 + 20 octets of TCP and IPv4 headers, to which 8 octets are added for LLC/SNAP encapsulation). AAL5 then turns 9188 octets into 192 ATM cells. Numerical results for this scenario will be shown in the case $N = 16$.

Spotted packet scenario. In this case, an unbalanced load is generated towards different outputs, to emphasize the performance limitations of simpler scheduling algorithms. In addition, packet sizes are chosen according to a bimodal distribution, to highlight possible starvation effects in the service of long or short packets.

We assume $N = 8$, and set:

$$\Gamma = \Gamma_P = \frac{1}{40} \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Packet lengths can be either 3 or 100 cells, with identical probabilities.

Diagonal packet scenario. In this case $\gamma_P^{(ij)} = 1/N^2$ (the traffic is uniform at the packet level), but all packets with equal index of input and output ports ($i = j$) have a fixed size equal to 100 cells, whereas the packet size always is equal to 3 cells for packets directed to an output port with different index from their input port ($i \neq j$). Numerical results for this scenario will be shown in the case $N = 16$. The reason for considering this scenario is to highlight the starvation effect in the service of short packets due to the transfer of long packets.

6.3.2 Performance indices

Results are presented with graphs where the following performance indices are plotted versus the switch normalized traffic load. The latter is defined as the ratio between the input traffic load and the total capacity of input/output lines, both of them computed at the cell level (hence, the normalized load varies between 0 and 1).

- **Cell delay.** (defined in Section 3.3.3).
- **Packet delay.** (defined in Section 3.3.3).
- **Matching persistence.** This is the probability that an active input port transmits a cell to the same output port that was addressed in the last time slot when the input port was active. The reason for defining and observing this performance index lies in the intuition that, under high loads, most inputs are active transferring packets to different outputs. When the transfer of a packet ends, some other input/output pairs remain busy (due to the variable size of packets), so that flexibility may be lost in choosing which packet can be scheduled next. In particular, if all other inputs and outputs are busy, nothing but another packet from the same input to the same output can be selected (thus possibly jeopardizing performance).

Simulation runs were executed until the estimate of the average cell delay reached with probability 0.95 a relative width of the confidence interval equal to 2%.

6.3.3 Uniform packet scenario

Fig. 6.3 shows, for the four considered scheduling algorithms operating in *cell-mode*, curves of the average packet delay. Note that IQ switches always yield longer delays than FIFO-OQ and VIQ-OQ switches, but differences are limited within a factor 2.5, for load smaller than 0.95. No losses were experienced with queue lengths equal to 30,000 cells.

For loads less than 0.9, MUCS and RPA generate longer delays than iSLIP and iOCF. This is mainly due to the QL and ML metrics, which tend to equalize queue lengths: when a large packet arrives at a particular queue, the queues (at other inputs) that store small packets directed to the same output suffer a temporary starvation. iOCF, instead, provides the lowest delays among weight-aware algorithms.

Performance results are significantly different if packet-mode scheduling proposed in this chapter is considered. Curves of the average packet delays are shown in Fig. 6.4 where a significant reduction in packet delays with respect to cell-mode scheduling can be observed. Differences between the four algorithms are limited, and

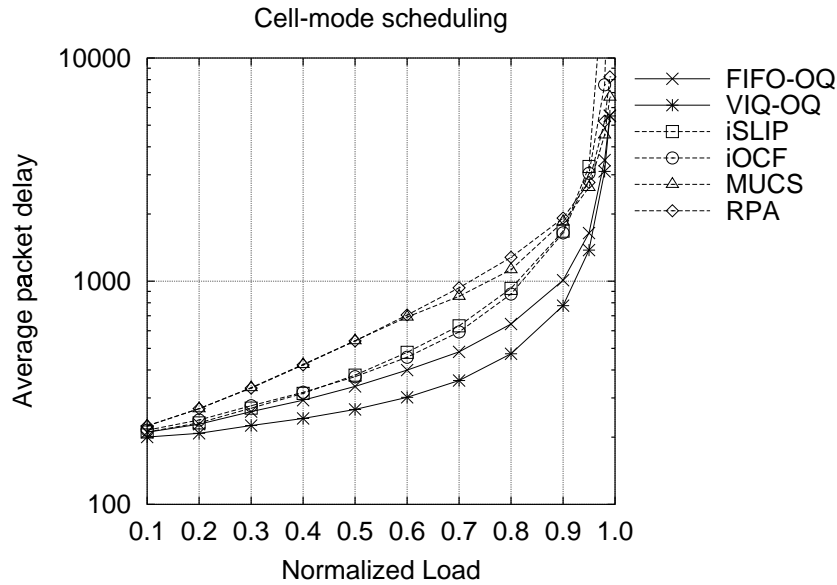


Figure 6.3. Average packet delay for cell-mode scheduling in the uniform traffic scenario

we observe (small) gains over FIFO-OQ switches for loads up to around 0.6. IQ exhibits the largest delay advantage over FIFO-OQ at loads around 0.4. Cell delays in this same scenario, not shown here, are larger for all IQ architectures with respect to both OQ architectures at all loads. The VIQ-OQ yields the best performance for all loads.

The reductions in packet delays are interesting, specially if we consider the negligible additional cost of implementing packet-mode schedulers. The simple queueing model of Section 6.1.3 justifies this performance improvement, which is not very intuitive (in this case $C_v = 0.287$). Note that more complex scheduling algorithms, better tailored to the statistics of packet traffic, could probably provide even larger gains.

6.3.4 Spotted packet scenario

This scenario aims to emphasize the performance limitations and the possible starvation of simpler algorithms under unbalanced loads.

Figs. 6.5 and 6.6 show, respectively, the average packet delays when cell-mode and packet-mode scheduling is adopted. With queue lengths equal to 10,000 cells, iSLIP experienced losses for traffic loads greater than 0.9 in cell-mode, and greater

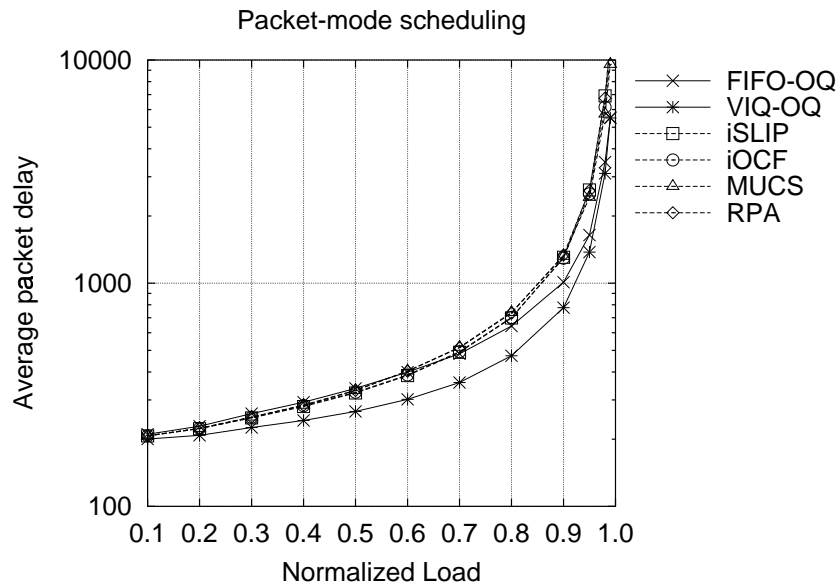


Figure 6.4. Average packet delay for packet-mode scheduling in the uniform traffic scenario

than 0.95 in packet-mode. No packet losses were suffered by other scheduling algorithms.

Packet-mode improves the performance for high input traffic because of the following reasons:

- when iSLIP operates in packet-mode, the QO metric gains sensibility about the packet length, since the matching is blocked until the packet transmission is completed. In this way the transient high load in a queue due to a packet arrival is “perceived” by the algorithm, which gives priority to large packets;
- when a packet transmission is completed, packet-mode algorithms compute the new matching only for all those inputs and outputs that are not currently involved in packet transfers. While the outcome of the scheduling algorithm is generally not a maximal matching, by reducing the number of inputs available the achieved matching is more likely maximal. For example, iSLIP and iOCF always find a maximal matching when $\log N$ inputs are available for the matching.

The matching persistence (not shown) is always greater than 0.98 for algorithms operating in packet-mode. Fig. 6.7 shows the average matching persistence when

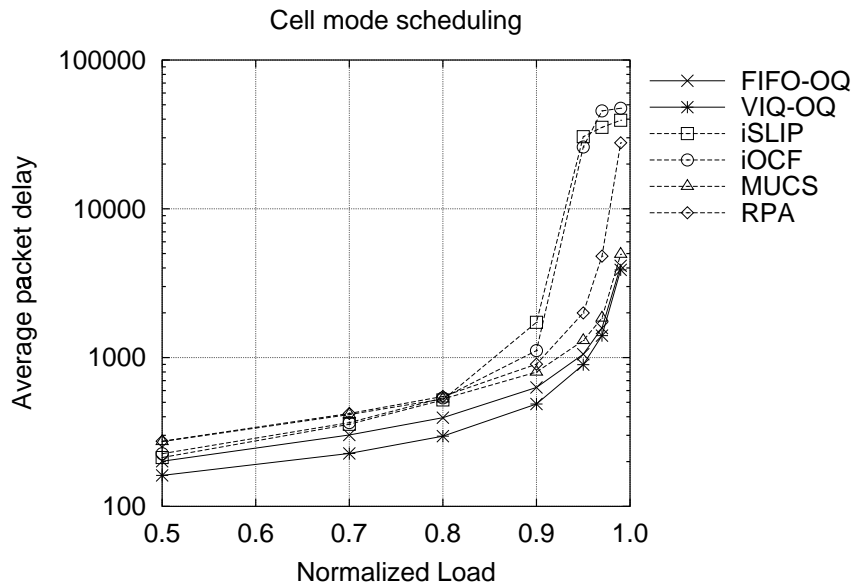


Figure 6.5. Average packet delay for cell-mode scheduling in the spotted traffic scenario

cell-mode scheduling is adopted. With cell-mode scheduling, for low traffic load, few packets enter the switch, and the matching changes only when a new packet arrives, so that the matching persistence is high. For high traffic load, when a QO metric is adopted (i.e., with iSLIP) the persistence tends to zero because the matching is forced to change by the adoption of the round-robin contention resolution. On the contrary, when metrics depend on queue lengths (even indirectly, like in iOCF), for high load, a large packet entering the switch increases significantly the input queue weight, as well as its priority, so that the matching persistence grows. However, note that the large matching persistence of packet-mode scheduling does not negatively affect throughput and delay.

6.3.5 Diagonal packet scenario

Recall that this scenario is used to highlight possible starvation in the service of short packets due to the transfer of large packets. Figs. 6.8 and 6.9 show the average packet delay curves for large packets when, respectively, cell-mode and packet-mode are adopted with the diagonal traffic scenario.

Only iSLIP experienced losses in cell-mode with queue lengths equal to 10,000 cells. When packet-mode is adopted, the total packet delay is always smaller than for

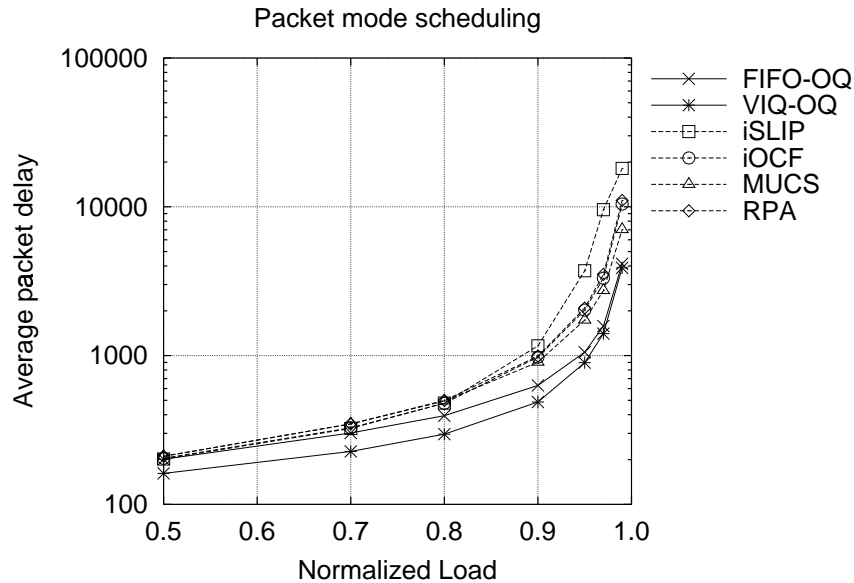


Figure 6.6. Average packet delay for packet-mode scheduling in the spotted traffic scenario

cell-mode (except when losses were experienced).

Packet-mode scheduling obtains almost always packet delays smaller than FIFO-OQ, whereas this is not true for cell-mode scheduling. This fact is due to the chosen packet length distribution, as foreseen by the model presented in Section 6.1.3. The delay improvement obtained with the use of packet-mode scheduling in the transfer of long packets is paid by shorter packets, as can be observed in Fig. 6.10. Note that the delay increase for short packets can be detrimental to some applications (e.g., real-time applications using short packets).

The overall packet delay (averaged over large and short packets) is always smaller for VIQ-OQ architectures.

6.3.6 Packet-mode gains

Next, we show some simulation results concerning the actual packet-mode gain values under different packet lengths distributions. Fig. 6.11 shows G for uniform distribution of packet lengths (with $C_v = 0.287$). Fig. 6.12 shows G for a bimodal distribution of packet lengths (with $C_v = 4.975$). Fig. 6.13 shows G for negative exponential distribution of packet lengths.

Table 6.1 compares the theoretical values with the experienced packet-mode gains

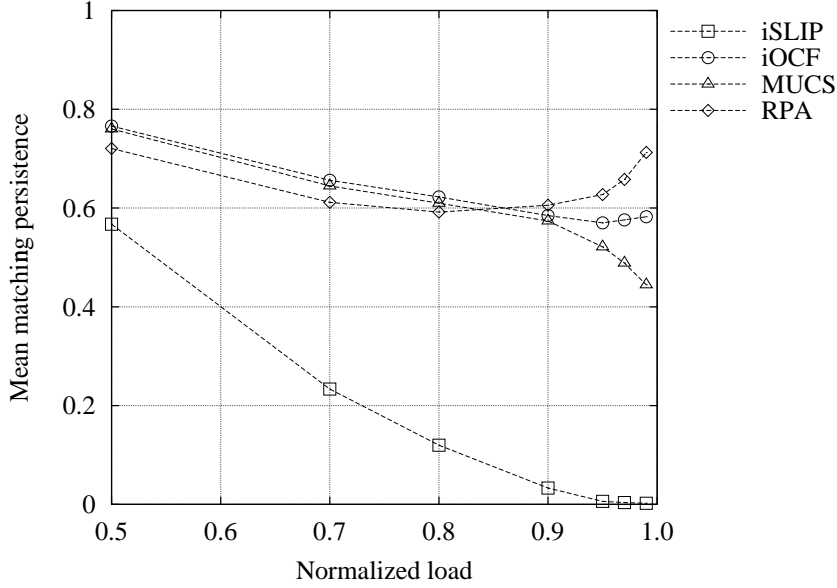


Figure 6.7. Average matching persistence for cell-mode scheduling in the spotted traffic scenario

C_v	G			
	by theory	by simulation for loads:		
		0.1	0.3	0.5
0.287	1.84	1.49	1.43	1.35
1.00	1.00	0.98	0.94	0.93
4.975	0.077	0.052	0.072	0.11

Table 6.1. Packet-mode gains from theoretical and simulation analysis for iSLIP

for different scenarios, in the case in which iSLIP is adopted. We observe that the packet-mode gains obtained by simulation can be well estimated by the theoretical values, especially at low load. The proposed queueing model is thus capable of capturing the key aspects of the behavior of cell- and packet-mode iSLIP at low load.

Note also that iOCF in cell-mode shows very poor performance under bimodal packet length distribution for input load equal to 0.95. This is due mainly to the matching used in iOCF, which is not maximal; this drawback can be overcome by iterating the algorithm N times, as was experienced by simulation. This is another example of the fact that a reduction of the matching variability through the adoption of packet-mode scheduling increases the performance of non-maximal matching heuristics.

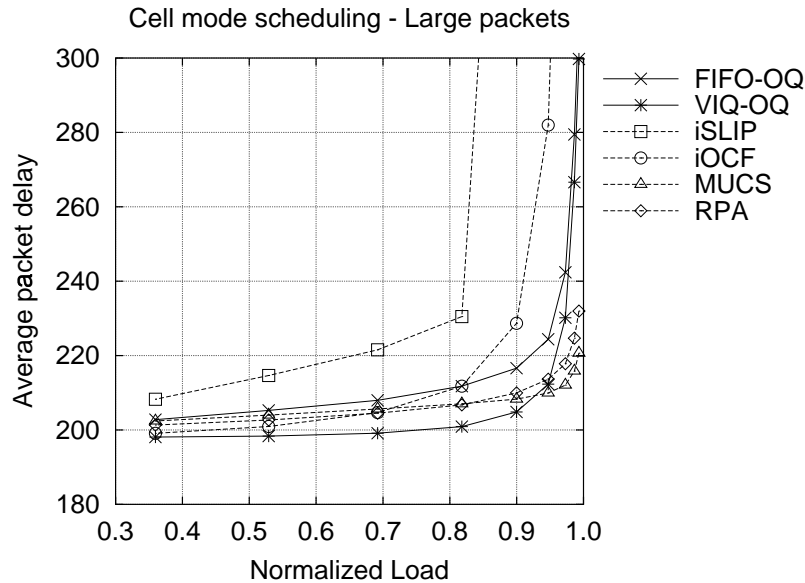


Figure 6.8. Packet delay for large packets in the diagonal traffic scenario with cell-mode scheduling

6.4 Combined input-output queued packet switches

CIOQ packet switches are built around a CIOQ cell switch. For the sake of conciseness, we do not show the logical architecture in this case, but it can be easily obtained by combining Figs. 6.1 and 6.2. The only difference is that we do not consider VOQ in the CIOQ case, hence only one FIFO queue is available at each input (and at each output) of the cell switch. Losses may occur at both input and output interfaces.

6.4.1 Performance of combined input-output queued packet switches

This section presents simulation results for CIOQ switches using the FIFO-2 scheduling algorithm described in Section 2.2.4. A speedup factor equal to 2, according to the $S^{(2)}$ speedup definition, is supposed to be available in the internal cell-switch. We consider the uniform traffic scenario only. The capacity of all queues is taken to be unlimited for the results presented in this section, in order to be able to observe the unbalancement of queue lengths between input and output queues.

Fig. 6.14 plots curves of the average packet delay normalized to OQ values. The curves in the figure refer to the following switch setups, which differ in the values

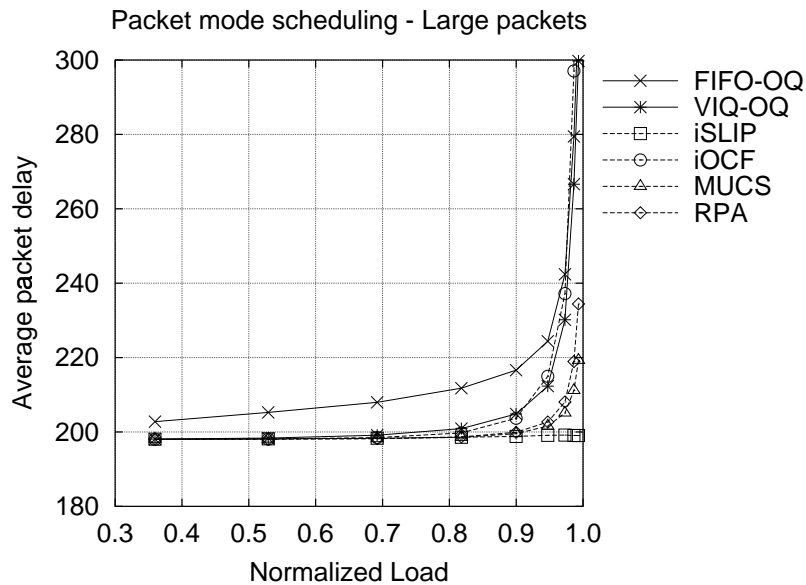


Figure 6.9. Packet delay for large packets in the diagonal traffic scenario with packet-mode scheduling

taken by the SPEEDUP-IP-IN and SPEEDUP-IP-OUT parameters (see Fig. 2.3 in Section 2.2.5), and in the operation mode (cell mode vs. packet mode) of the FIFO-2 scheduling algorithm (the extension of FIFO-2 to packet-mode operation is straightforward).

- **FF-121:** basic FIFO-2, $\text{SPEEDUP-IP-IN}=\text{SPEEDUP-IP-OUT}=1$. The cells belonging to IP packets are fed to the cell-switch at the same speed of input and output lines.
- **FF-221:** basic FIFO-2, $\text{SPEEDUP-IP-IN}=2$, $\text{SPEEDUP-IP-OUT}=1$. The cells belonging to IP packets are fed to the cell-switch at twice the speed of input and output lines.
- **FF-PM121:** packet-mode FIFO-2, $\text{SPEEDUP-IP-IN}=\text{SPEEDUP-IP-OUT}=1$. The cells belonging to the same packet are transferred to their output in contiguous slots. Note that up to two packets (received at different inputs) can be interleaved in output cell queues in this setup. An ORM module is therefore required at each output.
- **FF-PM221:** packet-mode FIFO-2, $\text{SPEEDUP-IP-IN}=2$, $\text{SPEEDUP-IP-OUT}=1$.

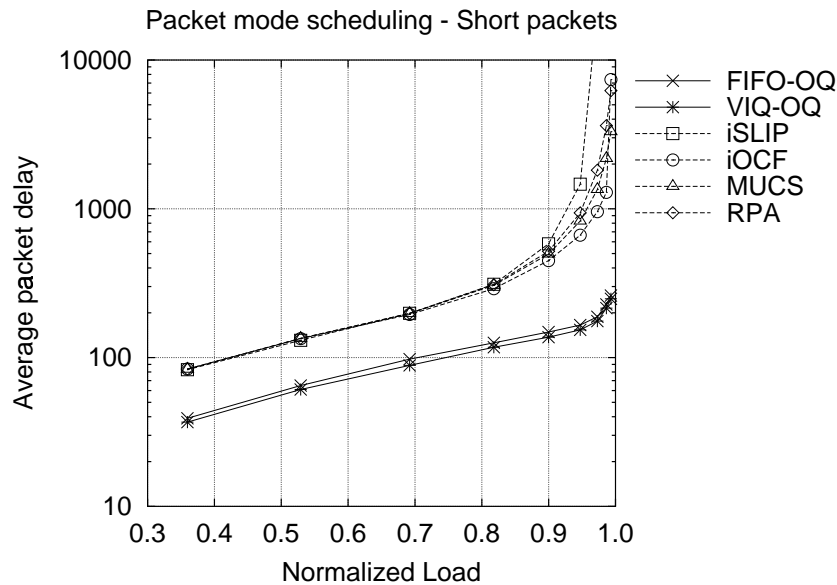


Figure 6.10. Packet delay for short packets in the diagonal traffic scenario with packet-mode scheduling

1. The packet-mode FIFO-2 scheduling algorithm has in this case the additional constraint that packets cannot be interleaved in output cell queues: this becomes possible without performance degradation because of $\text{SPEEDUP-IP-IN} = 2$, since two cells of the same packet can be switched to their output in one slot time. ORM modules are therefore not strictly necessary in this case: they are kept for a fair comparison with the other setups.

- **FF-PM222**: packet-mode FIFO-2, $\text{SPEEDUP-IP-IN} = 2$, $\text{SPEEDUP-IP-OUT} = 2$. The same constraints of FF-PM221 apply to the FIFO-2 algorithm: packets are not interleaved in output queues.

In the notation above, “FF” stands for FIFO, “PM” stands for packet-mode (recall the different constraints on the scheduling algorithm when SPEEDUP-IP-IN equals 1 or 2), and the three digits refer to the values taken by SPEEDUP-IP-IN , $S^{(2)}$, and SPEEDUP-IP-OUT , respectively.

As expected, average packet delays at low loads (not plotted here) were observed to be around twice the average packet duration for the architectures with $\text{SPEEDUP-IP-OUT} = 1$ (where one ISM and one ORM module must be traversed by each packet), and to be around 1.5 times the average packet duration for FF-PM222 (for which the ORM can be traversed by two packets in one packet time, i.e., at double speed).

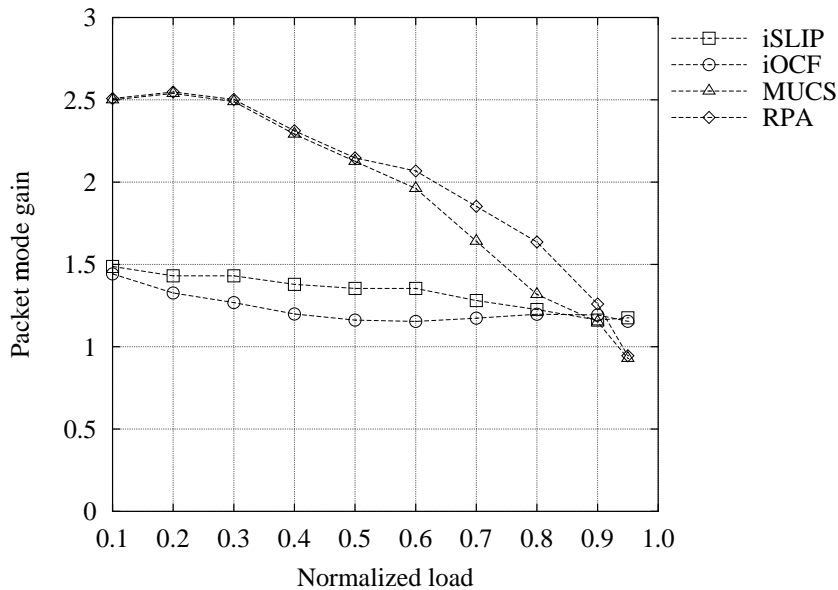


Figure 6.11. Packet-mode gain for uniform packet length distribution

The curves in Fig. 6.14 show that all the considered architectures operating in packet-mode have better performance than OQ packet switches. As previously observed, IQ and CIOQ switches provide extra advantages over OQ switches when variable-size packets are considered.

Note that the FIFO-2 algorithm in packet-mode can be more efficient than output queuing also when no speedup exists between the IP line interfaces and the internal cell-switch (i.e., for FF-PM121). The best-performing setups are FF-PM221 and FF-PM222 which avoid output packet interleaving. Delay ratios can be as small as 0.64 (i.e., 1.5 times in favor of CIOQ) when the ORM modules are present, and lower values can be achieved for the optimized architecture.

FF-121 exhibits delays slightly larger than OQ. Note that FF-121 can be considered as a simplified version of OQ, in which at most 2 (instead of N) cells can reach an output in a slot time.

It is also interesting to observe that FF-221 behaves better than FF-121, mainly thanks to the fact that the cells belonging to the same packet are kept closer to each other in the transfer through the switch.

Simulation results not presented here show that very similar behaviors were observed for CIOQ architectures also in the case of hot-spot traffic.

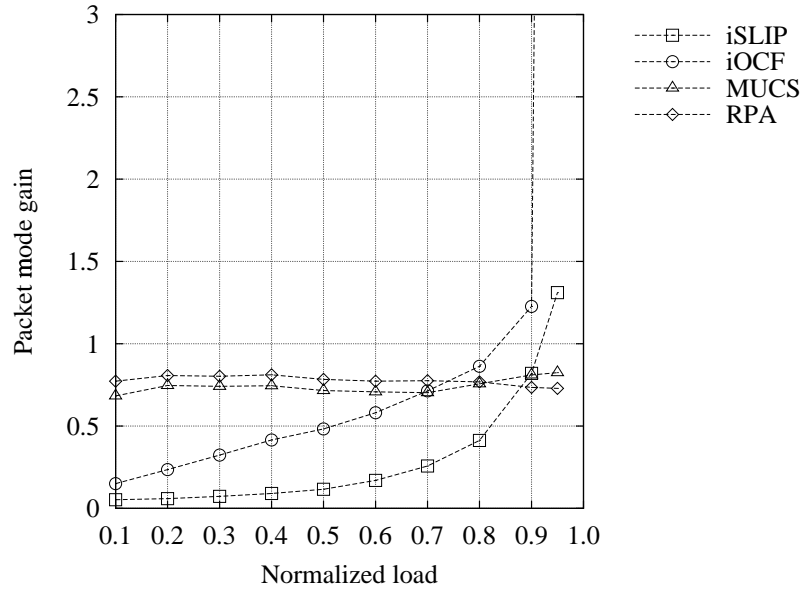


Figure 6.12. Packet-mode gain for bimodal packet length distribution

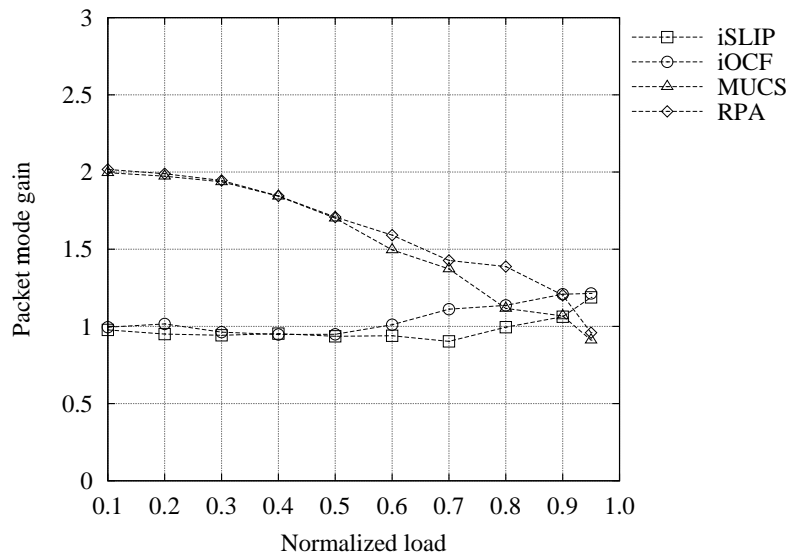


Figure 6.13. Packet-mode gain for exponential packet length distribution

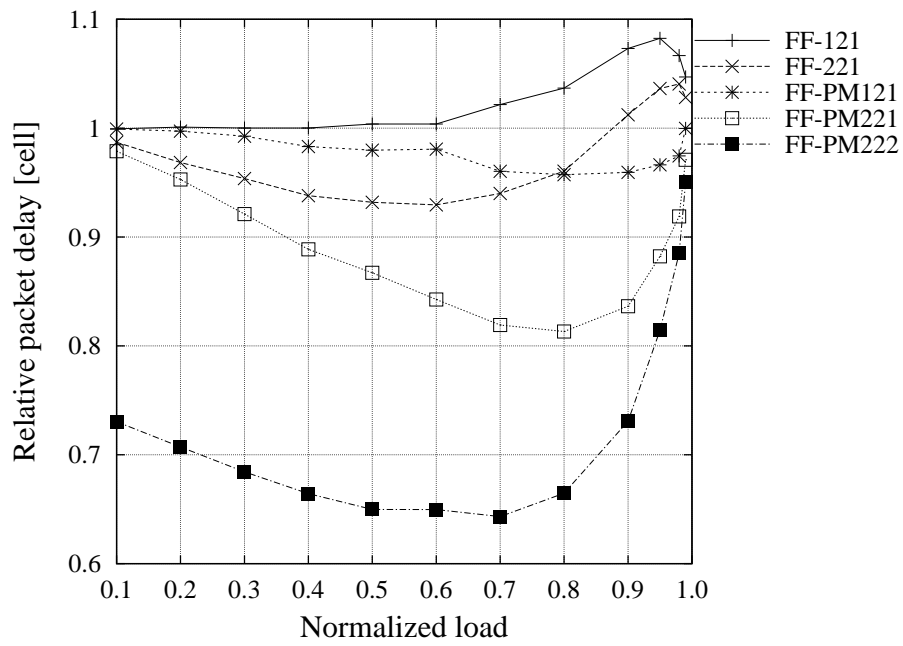


Figure 6.14. Relative average packet delay in CIOQ architectures for in the uniform traffic scenario

Chapter 7

Conclusions

In this Ph.D. dissertation we have discussed the design of scheduling algorithms for high speed switching architectures. We have focused our study mainly on input queued (IQ) switches, since they are the most promising architectures in terms of scalability with the speed and the number of ports.

We have described the main issues involved in the design of high performance routers and have motivated the problem of scheduling for IQ switched. After defining the problem of scheduling, as a problem of finding a matching, we have introduced the “tools” of our study: simulation and stochastic modeling to evaluate the performance of the algorithms.

So far, the scheduling algorithms known in literature have been either simple to implement but with poor performance or too complex but with good performance. Our first significant contribute has been the proposal of a suite of scheduling algorithms, based on the learning approach, with very low complexity and very good performance in terms of delays and throughput. The main ideas we exploited have been the memory of the state of the queues from the past, the randomization, the information about arrivals and the parallel search.

Our second significant contribute has been the design of scheduling algorithms for multicast traffic. We have proposed a new buffering scheme and defined the optimal policy for scheduling multicast traffic in an IQ switch. Thanks to that, we were able to find out some intrinsic throughput limitations in an IQ switch, fed by multicast traffic. We have proposed also an implementable greedy scheduler, whose performance analysis showed its good behavior for generic multicast traffic.

Usually, in an high speed router, the incoming packets are of variable size, whereas the internal switch fabric (implemented in hardware) works with cell of fixed size. Our third contribute has been the design of cell-based scheduling algorithms integrated within the router and optimized to transfer variable size packets. We have shown some counterintuitive properties of throughput and delays about these optimized algorithms.

We wish to make the last point. We think that the approaches of our three contributions are compatible one with each other. This means that all of them could be integrated in a practical scheduling algorithm to design an high performance router. It is a promising idea, but not trivial to be studied. So we consider this idea as a possible and promising topic of our future research work.

Bibliography

- [1] Ajmone Marsan M., Bianco A., Giaccone P., Leonardi E., Neri F., “Router architectures exploiting input-queued cell-based switching fabrics”, International Workshop on QoS in Multiservice IP Networks (QoS-IP 2001), Rome, Italy, Jan. 2001, pp. 223-238
- [2] Ajmone Marsan M., Bianco A., Leonardi E., Milia L., “RPA: a flexible scheduling algorithm for input buffered switches”, *IEEE Transactions on Communications*, vol. 47, n. 12, Dec. 1999, pp. 1921-1933
- [3] Ajmone Marsan M., Bianco A., Giaccone P., Leonardi E., Neri F., “Scheduling in input-queued cell-based packet switches”, *IEEE GLOBECOM'99*, Rio de Janeiro, Brazil, Dec. 1999
- [4] Ajmone Marsan M., Bianco A., Giaccone P., Leonardi E., Neri F., “Packet scheduling in input-queued cell-based switches”, *IEEE INFOCOM'01*, Anchorage, AK, Apr. 2001
- [5] Ajmone Marsan M., Bianco A., Giaccone P., Leonardi E., Neri F., “On the throughput of input-queued cell-based switches with multicast traffic”, *IEEE INFOCOM'01*, Anchorage, AK, Apr. 2001
- [6] Ajmone Marsan M., Bianco A., Giaccone P., Leonardi E., Neri F., “Optimal multicast scheduling in input-queued switches”, *IEEE ICC'01*, Helsinki, Finland, June 2001
- [7] Ajmone Marsan M., Bianco A., Filippi E., Giaccone P., Leonardi E., Neri F., “On the behavior of input queuing switch architectures”, *European Transactions on Telecommunications*, vol. 10, n. 2, Mar. 1999, pp. 111-124
- [8] Ajmone Marsan M., Leonardi E., Mellia M., Neri F., “On the stability of input-buffer cell switches with speed-up”, *IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000

- [9] Ajmone Marsan M., Leonardi E., Mellia M., Neri F., "On the stability of input-queued switches with speedup", *IEEE/ACM Transactions on Networking*, vol. 9, n. 1, Feb. 2001
- [10] Anderson T., Owicki S., Saxe J., Thacker C., "High speed switch scheduling for local area networks", *ACM Transactions on Computer Systems*, vol. 11, n. 4, Nov. 1993, pp. 319-351
- [11] Andrews M., Khanna S., Kumaran K., "Integrated scheduling of unicast and multicast traffic in an input-queued switch", *IEEE INFOCOM'99*, vol. 3, New York, NY, 1999, pp. 1144-1151
- [12] Atkinson R., "RFC 1626: Default IP MTU for use over ATM AAL5", May 1994
- [13] Awdeh Ra'ed Y., Mouftah H.T., "Survey of ATM switch architectures", *Computer Networks & ISDN Systems*, vol. 27, n. 12, Nov. 1995, pp. 1567-1613
- [14] Azar Y., Broder A., Karlin A., Upfal E., "Balanced allocations", *ACM STOC*, 1994, pp. 593-602
- [15] Bennett J.C.R., Zhang H., "WF²Q: Worst-case fair weighted fair queuing", *IEEE INFOCOM'96*, San Francisco, CA, Mar. 1996, pp. 120-128
- [16] Bux W., Denzel W.E., Engbersen T., Herkersdorf A., Luijten R.P., "Technologies and building blocks for fast packet forwarding", *IEEE Communications Magazine*, Jan. 2001, pp. 70-77
- [17] Chang C.S., Lee D.S., Jou Y.S., "Load balanced Birkhoff-von Neumann switches", *2001 IEEE Workshop on High Performance Switching and Routing*, 2001, pp. 276-280.
- [18] Chen J.S., Stern T.E., "Throughput analysis, optimal buffer allocation, and traffic imbalance study of a generic nonblocking packet switch", *IEEE Journal on Selected Areas in Communications*, vol. 9, n. 3, Apr. 1991, pp. 439-449
- [19] Chen W., Chang Y., Hwang W., "A high performance cell scheduling algorithm in broadband multicast switching systems", *IEEE GLOBECOM'97*, vol. 1, New York, NY, 1997, pp. 170-174
- [20] Chen X., Lambadaris I., Hayes J., "A general unified model for performance analysis of multicast switching", *IEEE GLOBECOM'92*, vol. 3, New York, NY, 1992, pp. 1498-1502

- [21] Chiussi F.M., Xia Y., Kumar V.P., "Backpressure in shared-memory-based ATM switches under multiplexed bursty sources", *IEEE INFOCOM'96*, San Francisco, CA, Mar. 1996, pp. 830-843
- [22] Chuang S.T., Goel A., McKeown N., Prabhakar B., "Matching output queueing with a combined input/output-queued switch", *IEEE Journal on Selected Areas in Communications*, vol. 17, n. 6, Jun. 1999, pp. 1030-39
- [23] "Cisco 12000 gigabit switch router", Product Overview, www.cisco.com, Apr. 2000
- [24] Dagermark M., Brodnik A., Carlsson S., Pink S., "Small forwarding tables for fast routing lookups", *ACM SIGCOMM'97*, Cannes, France, Sep. 1997, pp. 3-14
- [25] Dai J., Prabhakar B., "The throughput of data switches with and without speedup", *IEEE INFOCOM 2000*, vol. 2, Tel Aviv, Israel, Mar. 2000, pp. 556-564
- [26] Demers A., Keshav S., Shenker S., "Analysis and simulation of a fair queueing algorithm", *ACM SIGCOMM'89*, Austin, TX, Sept. 1989, pp. 3-12
- [27] Duan H., Lockwood J.W., Kang S.M., Will J.D., "A high performance OC12/OC48 queue design prototype for input buffered ATM switches", *IEEE INFOCOM'97*, vol. 1, Kobe, Japan, 1997, pp. 20-28.
- [28] Even S., Kariv O., "An $O(n^{2.5})$ algorithm for maximum matching in general graphs", *16th Symposium on Foundations of Computer Science*, University of California at Berkeley, Oct. 1975, pp. 100-112
- [29] "GFR MultiGigabit Routers", Product Overview, www.lucent.com, Apr. 2000
- [30] Giaccone P., "Tecniche di accodamento e trasferimento in architetture di commutazione a larga banda", Laurea Thesis, Politecnico di Torino, Italy, May 1998, *in Italian*
- [31] Giaccone P., Prabhakar B., Shah D., "Towards simple, high-performance schedulers for high-aggregate bandwidth switches", *IEEE INFOCOM'02*, New York, NY, Jun. 2002
- [32] Giaccone P., Shah D., Prabhakar B., "An implementable parallel scheduler for input-queued switches", *Hot Interconnects 9*, Stanford, CA, Aug. 2001, pp. 9-14
- [33] Golestani S., "A self-clocked fair queueing scheme for broadband applications", *IEEE INFOCOM'94*, Toronto, Canada, June 1994, pp. 636-646

- [34] Goudreau M.W., Kolliopoulos S.G., Rao S.B., "Scheduling algorithms for input-queued switches: randomized techniques and experimental evaluation", *IEEE INFOCOM 2000*, vol. 3, Tel-Aviv, Israel, Mar. 2000, pp. 1634-1643
- [35] Guo M., Chang R., "Multicast ATM switches: survey and performance evaluation", *Computer Communication Review*, vol. 28, n. 2, Apr. 1998, pp. 98-131
- [36] Gupta A.K., Georganas N.D., "Analysis of a packet switch with input and output buffers and speed constraints", *IEEE INFOCOM'91*, Bal Harbour, FL, Apr. 1991, pp. 694-700
- [37] Gupta P., Lin S., McKeown N., "Routing lookups in hardware at memory access speed", *IEEE INFOCOM'98*, San Francisco, CA, Apr. 1998, pp. 1240-1247
- [38] Hayes J.F., Breault R., Mehmet-Ali M.K., "Performance analysis of a multicast switch", *IEEE Transactions on Communications*, vol. 39, n. 4, Apr. 1991, pp. 581-587
- [39] Hluchyj M.G., Karol M.J., Morgan S., "Input versus output queueing on a space division switch", *IEEE Transactions on Communications*, vol. 35, n. 12, Dec. 1987, pp. 1347-1356
- [40] Hopcroft J.E., Karp R.M., "An $n^{2.5}$ algorithm for maximum matching in bipartite graphs", *Society for Industrial and Applied Mathematics Journal on Computing*, vol. 2, n. 4, Dec. 1973, pp. 225-231
- [41] Hui J., Renner T., "Queueing strategies for multicast packet switching", *IEEE GLOBECOM'90*, 1990, pp. 1431-1437
- [42] Hui J.Y., "Switching and traffic theory for integrated broadband networks", *Kluwer Academic Publishers*, Jan. 1990
- [43] Hung A., Kesidis G., McKeown N., "ATM input-buffered switches with guaranteed-rate property", *IEEE ISCC'98*, Athens, Greece, Jul. 1998, pp. 331-335
- [44] Iliadis I., Denzel W.E., "Performance of packet switches with input and output queueing", *IEEE ICC'90*, Atlanta, GA, Apr. 1990, pp. 747-53
- [45] Iyer S., McKeown N., "Making parallel packet switches practical", *IEEE INFOCOM'01*, Anchorage, AK, Mar. 2001
- [46] Iyer S., Kompella R.R., McKeown N., "Techniques for fast packet buffers", *Gigabit Networking Workshop*, Anchorage, AK, Apr. 2001

- [47] Kam A., Siu K.Y., “Linear-complexity algorithms for QoS support in input-queued switches with no speedup”, *IEEE JSAC*, vol. 17, n. 6, Jun. 1999, pp. 1040-1056
- [48] Karol M., Eng K., Obara H., “Improving the performance of input-queued ATM packets switches”, *IEEE INFOCOM’92*, Firenze, Italy, May 1992, pp. 110-115
- [49] Karol M., Hluchyj M., Morgan S., “Input versus output queuing on a space division switch”, *IEEE Transactions on Communications*, vol. 35, n. 12, Dec. 1987, pp. 1347-1356
- [50] Keshav S., Sharma R., “Issues and trends in router design”, *IEEE Communications Magazine*, vol. 36, n. 5, May 1998, pp. 144-151
- [51] Kim C.K., Lee T.T., “Call scheduling algorithm in multicast switching systems”, *IEEE Transactions on Communications*, vol. 40, n. 3, Mar. 1992, pp. 625-635
- [52] Kim C.K., Lee T.T., “Performance of call splitting algorithms for multicast traffic”, *IEEE INFOCOM’90*, 1990, pp. 348-356
- [53] Kleinrock L., *Queueing systems – Volume 1: Computer applications*, John Wiley and Sons, 1975
- [54] Kleinrock L., *Queueing systems – Volume 2: Computer applications*, John Wiley and Sons, 1976
- [55] Krishna P., Patel N.S., Charny A., Simcoe R.J., “ On the speedup required for work-conserving crossbar switches”, *IEEE Journal on Selected Areas in Communications*, vol. 17, n. 6, Jun. 1999, pp. 1057-1066
- [56] Kumar P.R., Meyn S. P., “Stability of queueing networks and scheduling policies”, *IEEE Transactions on Automatic Control*, vol. 40, n. 2, Febr. 1995, pp. 251-260
- [57] Leonardi E., Mellia M., Neri F., Ajmone Marsan M., “Bounds on average delays and queue size averages and variances in input queued cell-based switches”, *IEEE INFOCOM’01*, Anchorage, AK, vol. 3, Apr. 2001, pp.1095-1103
- [58] van Lint J. H., Wilson R. M., “A course in combinatorics”, *Cambridge University Press*, 1992
- [59] Liu Z., Righter R., “Scheduling multicast input-queued switches”, *Journal of Scheduling*, John Wiley & Sons, May 1999

- [60] McKeown N., "iSLIP: a scheduling algorithm for input-queued switches", *IEEE Transactions on Networking*, vol. 7, n. 2, Apr. 1999, pp. 188-201
- [61] McKeown N., "Scheduling algorithms for input-queued cell switches", *Ph.D. Thesis*, University of California at Berkeley, 1995
- [62] McKeown N., Anantharam V., Walrand J., "Achieving 100% throughput in an input-queued switch", *IEEE INFOCOM'96*, vol. 1, San Francisco, CA, Mar. 1996, pp. 296-302
- [63] McKeown N., Anderson T.E., "A quantitative comparison of scheduling algorithms for input-queued switches", *Computer Networks and ISDN Systems*, vol. 30, n. 24, Dec. 1998, pp. 2309-2326
- [64] McKeown N., Izzard M., Mekkittikul A., Ellesick B., Horowitz M., "The Tiny Tera: a packet switch core", *IEEE Micro Magazine*, vol. 17, Feb. 1997, pp. 27-40
- [65] McKeown N., Mekkittikul A., Anantharam V., Walrand J., "Achieving 100% throughput in an input-queued switch", *IEEE Transactions on Communications*, vol. 47, n. 8, Aug. 1999, pp. 1260-1267
- [66] McKeown N., Mekkittikul A., "A starvation free algorithm for achieving 100% throughput in an input queued switch ", *ICCCN'96*, Rockville, MD, Oct. 1996, pp. 694-700
- [67] McKeown N., Prabhakar B., "Scheduling multicast cells in an input-queued switch", *INFOCOM'96*, vol. 1, San Francisco, CA, Mar. 1996, pp. 261-278
- [68] Mekkittikul A., McKeown N., "A practical scheduling algorithm to achieve 100% throughput in input-queued switches", *IEEE INFOCOM'98*, vol. 2, New York, NY, Apr. 1998, pp. 792-799
- [69] Minkenbergh C., Engbersen T., "A combined input and output queued packet-switched system based on PRIZMA switch on-a-chip technology", *IEEE Communications Magazine*, vol. 38, n. 12, Dec. 2000, pp. 70-77
- [70] Mitzenmacher M., "The power of two choices in randomized load balancing", *PhD thesis*, University of California, Berkeley, 1996
- [71] Motwani R., Raghavan P., "Randomized algorithms", *Cambridge University Press*, 1995
- [72] Newman P., Minshall G., Lyon T., Huston L., "IP switching and gigabit routers", *IEEE Communications Magazine*, Jan. 1997, pp. 64-69

- [73] Nijenhuis A., Wilf H., "Combinatorial algorithms: for computers and calculators", 2nd Edition, Academic Press, chap. 7, New York, 1978, p. 56
- [74] Nong G., Hamdi M., "On the provision of integrated QoS guarantees of unicast and multicast traffic in input-queued switches", *IEEE GLOBECOM'99*, vol. 3, 1999, pp. 1742-1746
- [75] Oie Y., Murata M., Kubota K., Miyahara H., "Performance analysis of non blocking packet switch with input and output buffers", *IEEE Transactions on Communications*, vol. 40, n. 8, Aug. 1992, pp. 1294-1297
- [76] Oie Y., Suda T., Murata M., Miyahara H., "Survey of the performance of non blocking switches with FIFO input buffers", *IEEE ICC'90*, Atlanta, GA, Apr. 1990, pp. 737-741
- [77] Parekh A.K., Gallager R.G., "A generalized processor sharing approach to flow control in integrated services networks - the single node case", *IEEE/ACM Transactions on Networking*, vol. 1, n. 3, Jun. 1993, pp. 344-357
- [78] Partridge C., et al., "A 50-Gb/s IP router", *IEEE Transactions on Networking*, vol. 6, n. 3, Jun. 1998, pp. 237-248
- [79] Pattavina A., "Switching theory: architectures and performance in broadband ATM networks", Wiley, John & Sons, Dec. 1997
- [80] Pawlikowski K., "Steady-state simulation of queueing processes: a survey of problems and solutions", *ACM Computing Surveys*, vol.22, n.2, June 1990, pp. 123-163
- [81] Prabhakar B., McKeown N., Ahuja R., "Multicast scheduling for input-queued switches", *IEEE Journal on Selected Areas in Communications*, vol. 15, n. 5, Jun. 1997, pp. 855-866
- [82] Prabhakar B., McKeown N., "On the speedup required for combined input and output queued switching", *Automatica*, vol 35, n. 12, 1999, pp. 1909-1920
- [83] Psounis K., Prabhakar B., "A randomized web-cache replacement scheme", *IEEE INFOCOM'01*, Anchorage, AK, Apr. 22-26, 2001
- [84] Schoenen R., Post G., Sander G., "Weighted arbitration algorithms with priorities for input-queued switches with 100% throughput", *BSS'99, 3rd IEEE International Workshop on Broadband Switching Systems*, Kingston, Canada, Jun. 1999

- [85] Shah D., Giaccone P., Prabhakar B., "An efficient randomized algorithm for input-queued switch scheduling", Hot Interconnects 9, Stanford, CA, Aug. 2001, pp. 3-8
- [86] Stiliadis D., Varma A., "Providing bandwidth guarantees in an input buffered crossbar switch", *IEEE INFOCOM'95*, vol. 3, Los Alamitos, CA, 1995, pp. 960-968
- [87] Stoica I., Zhang H., "Exact emulation of an output queueing switch by a combined input output queueing switch", *IWQoS'98, 6th International Workshop on Quality of Service*, Napa, CA, May 1998, pp. 218-224
- [88] Tamir Y., Chi H.-C., "Symmetric crossbar arbiters for VLSI communication switches", *IEEE Transaction on Parallel and Distributed Systems*, vol. 4, n. 1, Jan. 1993, pp. 13-27
- [89] Tamir Y., Frazier G., "High performance multi-queue buffers for VLSI communication switches", *15th Ann. Symp. on Comp. Arch.*, Washington, DC, Jun. 1988, pp. 343-354
- [90] Tarjan R.E., *Data structures and network algorithms*, Society for Industrial and Applied Mathematics, Pennsylvania, Nov. 1983
- [91] Tassiulas L., "Linear complexity algorithms for maximum throughput in radio networks and input queued switches", *IEEE INFOCOM'98*, vol. 2, New York, NY, 1998, pp. 533-539
- [92] Tassiulas L., Ephremides A., "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks", *IEEE Transactions on Automatic Control*, vol. 37, n. 12, Dec. 1992, pp. 1936-1948
- [93] Tweedie R.L., "The existence of moments of stationary markov chains", *Journal of Applied Probability*, n. 20, 1983, pp. 191-196
- [94] Waldvogel M., Varghese G., Turner J., Plattner B., "Scalable high-speed IP routing lookups", *ACM SIGCOMM'97*, Cannes, France, Sep. 1997, pp. 25-36
- [95] Wolf T., Turner J.S., "Design issues for high-performance active routers", *IEEE Journal on Selected Areas in Communications*, vol. 19, n. 3, Mar. 2001, pp. 404-409
- [96] Wolff R.W., *Stochastic modeling and the theory of queues*, Prentice-Hall, NJ, 1989

BIBLIOGRAPHY

- [97] Yeh Y., Hluchyj M.G., Acampora A.S., “The Knockout switch: a simple, modular architecture for high performance packet switching”, *IEEE Journal on Selected Areas in Communications*, vol. 5, Oct. 1987, pp. 1274-1283
- [98] Zhang H., “Service disciplines for guaranteed performance service in packet-switching networks”, *Proceedings of the IEEE*, vol. 83, n. 10, Oct. 1995, pp. 1374-1396

BIBLIOGRAPHY

Appendix A

Proofs about scheduling multicast traffic

A.1 Optimality of the max-scalar policy for multicast traffic

Definition 20. Let $CH[\{D^{(k)}\}_k]$ represent the *convex hull* generated by all possible admissible departure vectors $D^{(k)}$.

Theorem 14. *A necessary condition to achieve 100% throughput in an IQ switch is that the input traffic pattern, described by a sequence of random arrival vectors A_n and satisfying the strong law of large numbers, hence stationary and ergodic, satisfies the following condition:*

$$E[A_n] \in CH[\{D^{(k)}\}_k]$$

Proof. Suppose that the switch achieves 100% throughput, i.e.:

$$\lim_{n \rightarrow \infty} \frac{X_n}{n} = 0 \quad \text{with probability 1}$$

i.e., the length of the queues remains finite, or at most grows to infinity with sub-linear rate. As a consequence:

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{X_n}{n} &= \lim_{n \rightarrow \infty} \frac{\sum_{i=0}^{n-1} (A_i - D_i)}{n} = \\ &= \lim_{n \rightarrow \infty} \left[\frac{\sum_{i=0}^{n-1} A_i}{n} - \frac{\sum_{i=0}^{n-1} D_i}{n} \right] = \\ &= E[A] - \lim_{n \rightarrow \infty} \frac{\sum_{i=0}^{n-1} D_i}{n} = 0 \quad \text{with probability 1} \end{aligned}$$

A is the stationary and ergodic arrival process, and A_n the random arrival vector at time n ; due to stationarity, $E[A] = E[A_n]$ holds.

The above equality implies that there exists a sequence of departure vectors such that

$$E[A] = \lim_{n \rightarrow \infty} \frac{\sum_{i=0}^{n-1} D_i}{n} \quad \text{with probability 1}$$

This means that, for some instances of D_i , $\forall \epsilon > 0$, there exists a n_0 such that, for all $n > n_0$,

$$\left\| E[A] - \frac{\sum_{i=0}^{n-1} D_i}{n} \right\| < \epsilon$$

Note that $\frac{\sum_{i=0}^{n-1} D_i}{n}$, for each n , is in $CH[\{D^{(k)}\}_k]$ by definition. As a consequence $E[A]$ is the limit of a sequence of vectors in $CH[\{D^{(k)}\}_k]$, and thus it is in $CH[\{D^{(k)}\}_k]$, since $CH[\{D^{(k)}\}_k]$ is a closed set. \square

Theorem 15. *An IQ switch implementing the max-scalar discipline is strongly stable for each traffic pattern such that $E[A_n]$ is an internal point of $CH[\{D^{(k)}\}_k]$.*

Proof. The assertion can be proved by using the Lyapunov function methodology. Since $E[A_n] \in CH[\{D^{(k)}\}_k]$ it immediately follows that, for all $X_n \neq 0$,

$$\frac{E[A_n]X_n^T}{\|X_n\|} < \frac{\hat{A}_n X_n^T}{\|X_n\|} = \frac{E[A_n]X_n^T}{\|X_n\|} \alpha \leq \frac{\max_{D_n} D_n X_n^T}{\|X_n\|}$$

where \hat{A}_n is the vector parallel to $E[A_n]$ on the border of $CH[\{D^{(k)}\}_k]$, and $\alpha = \frac{\|\hat{A}_n\|}{\|E[A_n]\|} > 1$. Denoting $\arg\{\max_{D_n} \{D_n X_n^T\}\}$ by D_n^* as in (5.8), we can thus write

$$\frac{(\hat{A}_n - D_n^*)X_n^T}{\|X_n\|} \leq 0 \tag{A.1}$$

Since $\hat{A}_n = \alpha E[A_n] = E[A_n] - (1 - \alpha)E[A_n]$, we can substitute in (A.1) and write:

$$\frac{(E[A_n] - (1 - \alpha)E[A_n] - D_n^*)X_n^T}{\|X_n\|} \leq 0$$

Hence, if r_{\min} is the minimum non-null component of vector $E[A_n]$,

$$\lim_{\|X_n\| \rightarrow \infty} \frac{(E[A_n] - D_n^*)X_n^T}{\|X_n\|} \leq (1 - \alpha) \frac{E[A_n]X_n^T}{\|X_n\|} \leq (1 - \alpha)r_{\min} < 0$$

(note that $(1 - \alpha)$ is a negative non-null quantity). By using the quadratic Lyapunov function associated with the identity matrix (see [9] and [62]), i.e., $V(X_n) = X_n X_n^T$, and assuming the max-scalar scheduling discipline, we can write

$$\begin{aligned} E[V(X_{n+1}) - V(X_n) \mid X_n] &= E[X_{n+1} X_{n+1}^T - X_n X_n^T \mid X_n] = \\ &= E[(X_n + A_n - D_n^*)(X_n + A_n - D_n^*)^T - X_n X_n^T] = \\ &= E[2(A_n - D_n^*) X_n^T + (A_n - D_n^*)(A_n - D_n^*)^T] \end{aligned}$$

Being $(A_n - D_n^*)(A_n - D_n^*)^T$ negligible for $\|X_n\| \rightarrow \infty$, if we choose $\epsilon = -(1 - \alpha)r_{\min}/2$, we observe that there exists $B > 0$ and $\epsilon > 0$ such that, for each X_n : $\|X_n\| > B$,

$$E[V(X_{n+1}) - V(X_n) \mid X_n] < -\epsilon \|X_n\|$$

If in addition $E[A_n A_n^T] < \infty$, then $E[V(X_{n+1}) \mid X_n] < \infty, \forall X_n$; since the conditions of Theorem 2 hold, the system is strongly stable. \square

Since the strong stability of a system also implies 100% throughput, by combining the previous results it follows that:

Corollary 7. *Under any i.i.d. sequence A_n of arrivals such that $E[A_n A_n^T] < \infty$, the max-scalar scheduling discipline maximizes the stability region of the switch, i.e., the switch cannot be stable under any other scheduling discipline if it is unstable under the max-scalar scheduling discipline.*

A.2 Multicast departure vectors

Let us define $F(i, j, k)$ as the number of possible departure vectors in a IQ switch with i inputs, j total outputs and k outputs still available, i.e., outputs that have not been fully scheduled yet.

The number of possible $D^{(k)}$ in an $L \times N$ switch is given by $F(L, N, N)$, which can be computed with the following recursive formula:

$$F(i, j, k) = F(i - 1, j, k) + \sum_{s=1}^k \binom{k}{s} 2^{j-s} F(i - 1, j, k - s) \quad (\text{A.2})$$

when $1 < i, j \leq N$ and $1 \leq k \leq j$. Furthermore:

$$F(i, j, k) = 1$$

when $i = 0$ or $j = 0$ or $k = 0$, since we include the all-zeros departure vector.

Let us consider Equation (A.2). $F(i-1, j, k)$ is the number of permutation matrices when the i -th input is not scheduled. The fanout of the considered permutation for the i -th input is s . The binomial factor is the number of fanout sets which can be generated by s outputs included in the k outputs still available. 2^{j-s} is the number of possible output configurations that can include the considered fanout set (of s elements). With a fanout set of s elements, $(i-1)$ remaining inputs, j total outputs and $(k-s)$ outputs not yet scheduled, $\binom{k}{s} 2^{j-s} F(i-1, j, k-s)$ is the total number of permutation matrices that have to be added to the permutation matrix in which the i -th input is not scheduled.

A.3 On the minimum frame lengths for complete multicast schedules

This appendix is referred in the previous Sections 5.3.5 and 5.6.

We first modify and extend the definitions introduced at the beginning of Section 5.4 in the following way, referring to scheduling a request set in a time frame:

- $R = \{r_k\}$ is the request set of (multicast) cells waiting to be switched at input queues at the beginning of a time frame
- $F = \{f_k\}$ is the set of time slots in a switching frame; $|F| = L$; L is the frame length, whose minimum value necessary for the transfer of all cells in R will be discussed in this appendix; note that at most NL cells can be transferred in a frame of length L

Each (multicast) cell $r_l \in R$ must be transferred from its input port to all its destination output ports. Thus, each $r_l \in R$ is associated with a pair (i_l, D_l) , where $i_l \in I, D_l \subseteq O$.

Definition 21. Two multicast cells r_l and r_k are said to be in conflict if they either are associated with the same input port ($i_l = i_k$), or have at least one destination in common ($D_l \cap D_k \neq \emptyset$).

In the switch architecture we consider, a subset of the cells in R can be transferred from input ports to output ports in the same time slot, provided that no cells in the subset are in conflict. Conflicts may force the transfer of a cell r_l to disjoint subsets of D_l in different time slots, since we accept fanout splitting.

Definition 22. A Scheduling $\mathcal{S}[R]$ is defined as a function whose domain is R and whose image is¹ $U = 2^{I \times 2^O \times F}$. U is the set of transfer units, whose elements u_{lk} associate with each multicast cell $r_l \in R$ a triple (i_l, D_{lk}, f_k) , where $i_l \in I$ is the cell

¹Given a set A , 2^A denotes the power set of A .

input port, $D_{lk} \subseteq D_l$ is a non-empty subset of the cell destinations, and $f_k \in F$ is a time slot in the frame.

Informally, \mathcal{S} assigns to each multicast cell $r_l \in R$ a set of transfer units $u_{lk} = (i_l, D_{lk}, f_k) \in U$; each transfer unit allows a (possibly partial) transfer of the multicast cell, i.e., a transfer toward a subset of the cell destinations.

Definition 23. A Scheduling $\mathcal{S}[R]$ is *admissible* if, given any two elements $(i_1, D_{1k}, f_k) \in U$ and $(i_2, D_{2n}, f_n) \in U$, such that $f_k = f_n$, then

- $i_1 \neq i_2$, and
- $D_{1k} \cap D_{2n} = \emptyset$

i.e., they are non conflicting.

In other words, no (possibly partial) transfer of two different multicast cells can occur in the same time slot of the frame if conflicts arise.

Definition 24. An Admissible Scheduling $\mathcal{AS}[R]$ is *complete* if $\cup_k D_{lk} = D_l$, $\forall r_l \in R$, i.e., if it achieves a complete transfer of all cells belonging to R , using a subset of time slots belonging to F .

Definition 25. A Time Slot Assignment $\mathcal{TSA}[R]$ is defined as a function whose domain is R and whose image is the power set of F ($\mathcal{TSA}[R] : R \rightarrow 2^F$).

Informally, a \mathcal{TSA} defines a correspondence between each cell $r_l \in R$ and the set of time slots in the frame in which copies of the cell are transferred.

A more complex, but more useful definition of a $\mathcal{TSA}[R]$ will allow us to relate a Time Slot Assignment $\mathcal{TSA}[R]$ to an Admissible Scheduling $\mathcal{AS}[R]$.

Definition 26. A Timing function $\mathcal{T}[U] : U \rightarrow 2^F$ is a function that, given a set $\{(i_l, D_{lk}, f_k)\}_k$, returns a set containing all the third components f_k of each element.

Informally, the timing function \mathcal{T} extracts the time slot information related to (possibly partial) multicast cell transfers.

Definition 27. A Time Slot Assignment $\mathcal{TSA}[R] = \mathcal{T}(\mathcal{AS}[R])$ is a composition of two functions, a Timing function \mathcal{T} applied to the image of an Admissible Scheduling function \mathcal{AS} .

Although a time slot assignment $\mathcal{TSA}[R]$ does not completely specify the scheduling of multicast cells to be transferred from sources to destinations (it only defines the set of used time slots, with no information about sources and destinations), we will concentrate our attention only on $\mathcal{TSA}[R]$.

Definition 28. A time slot assignment $\mathcal{TSA}[R]$ is said to be *complete* if there exists a complete Admissible Scheduling \mathcal{AS} such that $\mathcal{TSA}[R] = \mathcal{T}(\mathcal{AS}[R])$.

Definition 29. For each sub-set $A \subseteq R$, let $\mathcal{I}_{\mathcal{TS}\mathcal{A}}(A)$ be the image of A through $\mathcal{TS}\mathcal{A}[R]$, i.e., the set comprising all time slots in which some element of A is transferred.

Lemma 8. Let R be a k -complex request set. Consider a complete $\mathcal{TS}\mathcal{A}[R]$ and any set $D \subseteq R$ such that $|D| = k$; then, $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(D)| \geq k$.

Proof. Suppose that there exists a subset $D \subseteq R$, with $|D| = k$, such that $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(D)| < k$; since the request set is k -complex, there exists a destination o^* to which all cells in D are directed. Copies of the cells in D directed to o^* must be transferred in different time slots to avoid conflicts. Thus, at least k time slots are necessary to transfer all the cells in D . As a consequence, the schedule $\mathcal{TS}\mathcal{A}[R]$ is not complete. \square

As a trivial consequence:

Lemma 9. Let R be a k -complex request set. Consider a $\mathcal{TS}\mathcal{A}[R]$ on F ; if there exists a set D s.t. $D \subseteq R$, $|D| = k$ and $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(D)| < k$, then $\mathcal{TS}\mathcal{A}[R]$ is not complete.

We now introduce a lemma that will be used in subsequent proofs.

Lemma 10. For a strictly positive integer-valued random variable X whose average value $E[X] = \mu$ does not exceed m , the $\frac{100}{m}$ -th percentile does not exceed m .

Proof. Since $\mu \leq m$, by definition:

$$\begin{aligned} E[X] = \mu &= \sum_{i=1}^{\infty} i \Pr\{X = i\} \\ &= \sum_{i=1}^m i \Pr\{X = i\} + \sum_{i=m+1}^{\infty} i \Pr\{X = i\} \\ &\geq \sum_{i=1}^m \Pr\{X = i\} + (m+1) \sum_{i=m+1}^{\infty} \Pr\{X = i\} \\ &= \Pr\{X \leq m\} + (m+1)(1 - \Pr\{X \leq m\}) \end{aligned}$$

But, since $\mu \leq m$:

$$\Pr\{X \leq m\} + (m+1)(1 - \Pr\{X \leq m\}) \leq m$$

and finally, solving for $\Pr\{X \leq m\}$,

$$\Pr\{X \leq m\} \geq \frac{1}{m}$$

\square

We also need the following result in subsequent proofs, which is an extension of the well-known combinatorics “pigeonhole” principle.

Lemma 11. *If u transfer units (objects) are transferred in t time slots (containers), it is possible to find a subset C of containers, such that $|C| = c$ in which at least $\lceil \frac{uc}{t} \rceil$ objects are contained for each integer positive number $c \leq t$.*

The main analytical result on frame based IQ switches is in Theorem 11 of Section 5.6, whose statement is rephrased below:

Theorem 16. *Let R be a k -complex request set, such that $|R| = kN_a$. For any finite integer S , $S \geq 2$, there exists an integer N_0 such that $\forall N_a > N_0$ no complete $\mathcal{TSA}[R]$ exists with frame length $L \leq Sk$.*

Note that S is equivalent to an internal switch speedup, since the k -complex request set has k cells per output and k cells per active input, hence it can be transferred on the output lines in k time slots by an OQ switch. Thus, the theorem states that, if the number of active input ports is sufficiently large, no speedup value is sufficient to obtain a complete \mathcal{TSA} , hence to achieve 100% throughput. This result will be separately proved in the cases $S = 2$, $N_0 = 32$, and $S = 3$, $N_0 = 2187$; the proofs can be extended to larger values of S .

Proof. [Case $S = 2$]

Since we consider in this proof the case $S = 2$, we must show that no complete \mathcal{TSA} exists if $L \leq 2k$. Obviously, if no complete \mathcal{TSA} exists for a frame length $L = 2k$, no complete \mathcal{TSA} can be found for shorter frame lengths. Thus, we consider the case $L = 2k$.

We will prove that, for any possible $\mathcal{TSA}[R]$, a subset G of R can be found, such that $|G| = k$ and $|\mathcal{I}_{\mathcal{TSA}}(G)| < k$. Then, for Lemma 9, we can state that the considered $\mathcal{TSA}[R]$ is not complete.

The identification of set G will require a number of steps, defining a chain of subsets of R : $H \subseteq G \subseteq C \subseteq B \subseteq R$, as well as a subset W of the time slots in the frame F .

- Set B comprises all cells whose transfer is scheduled either once or twice in the frame (not a larger number of times).
- Set W comprises a number of time slots smaller than $k/2$, chosen so as to maximize the number of cells in B transferred (possibly partially) within W .
- Set C comprises all cells whose transfer is scheduled at least once within W (but not more than twice, being $C \subseteq B$).
- Set H comprises all cells whose transfer is scheduled only within W (either once or twice, being $H \subseteq B$).

- Set G is obtained by adding cells to H (if necessary, see below) to obtain $|G| = k$.

It is possible to visualize these sets by considering a scheduling as a $L \times N$ matrix \mathbf{S} , where rows correspond to time slots in the frame F , and columns correspond to switch input ports in I . The elements of \mathbf{S} are $s_{nl} = 0$ if no cell from i_l is scheduled for transfer in time slot f_n , and $s_{nl} = m$ if in time slot f_n the transfer of cell r_m is scheduled from i_l to a subset of its destinations D_{mk} . By so doing, the definition of the above sets becomes:

- B comprises all cells whose identifier appears either once or twice in \mathbf{S} (but no more than twice)
- W comprises the rows of \mathbf{S} with the largest numbers of elements of B (selecting at most $k/2 - 1$ rows)
- C comprises all cells of B whose identifier appears in the rows of \mathbf{S} within W (the same identifier may appear also once outside W)
- H comprises all cells of B whose identifier appears only in the rows of \mathbf{S} within W (either once or twice)

To obtain a complete $\mathcal{TS}\mathcal{A}[R]$ it is necessary that $\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_l\}) \neq \emptyset, \forall r_l \in R$; thus, we will consider only the class of $\mathcal{TS}\mathcal{A}[R]$ for which $\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_l\}) \neq \emptyset, \forall r_l \in R$.

If $L = 2k$, then the average size of the image of individual cells in R through $\mathcal{TS}\mathcal{A}[R]$ cannot be greater than 2 (because the copies of kN_a cells must fit into $2kN_a$ time slots; considering matrix \mathbf{S} , at most all its $2kN_a$ elements can be nonzero, thus scheduling on the average twice the transfer of each one of the kN_a cells), i.e.: $E_{r_l} [|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_l\})|] \leq 2$. As a consequence, from Lemma 10, $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_l\})| \leq 2$ for at least half of the cells in R .

Let $B \subset R$ be the set of cells for which $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_l\})| \leq 2, \forall r_l \in B$; for Lemma 10, $|B| \geq |R|/2 = kN_a/2$. Thus, the average number of cells in B that are scheduled in each time slot of the frame is never less than $\frac{|B|}{L} \geq \frac{kN_a}{2} \frac{1}{2k} = \frac{N_a}{4}$.

For Lemma 11, there exists a set W of time slots with $|W| < k/2$ in which a set C of cells belonging to B (i.e., $C \subseteq B$) is scheduled, and $|C| \geq |W|N_a/4$.

Let us now restrict the attention to the case $|C| \geq 4k$, which implies that $N_a > 32$, and that k must be sufficiently large to generate enough cells to significantly fill the frame with transfers.

Let H be the set of cells $r_l \in C$ for which $\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_l\}) \subseteq W$. Note that H comprises all the cells in C for which $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_l\})| = 1$, and that H may be a null set.

Now we consider three cases:

- $|H| \geq k$. We can set $G = H$, obtaining $|G| \geq k$ and $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(G)| < k$.
- $k/2 \leq |H| < k$. Let G be a set of cells such that $H \subseteq G \subset C$, $|G| = k$. We define a set of time slots $X = \mathcal{I}_{\mathcal{TS}\mathcal{A}}(G \setminus H) \cap W$. Since $\mathcal{I}_{\mathcal{TS}\mathcal{A}}(H) \subseteq W$, by construction:

$$\begin{aligned}\mathcal{I}_{\mathcal{TS}\mathcal{A}}(G) &\subseteq W \cup [\mathcal{I}_{\mathcal{TS}\mathcal{A}}(G \setminus H) \setminus X] \\ |\mathcal{I}_{\mathcal{TS}\mathcal{A}}(G)| &\leq |W| + |\mathcal{I}_{\mathcal{TS}\mathcal{A}}(G \setminus H) \setminus X|\end{aligned}$$

Since $\mathcal{I}_{\mathcal{TS}\mathcal{A}}(r_l), \forall r_l \in (G \setminus H)$ has at most one element in $[\mathcal{I}_{\mathcal{TS}\mathcal{A}}(G \setminus H) \setminus X]$:

$$\begin{aligned}|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(G \setminus H) \setminus X| &= \left| \bigcup_l \mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_l \mid r_l \in (G \setminus H)\}) \setminus X \right| \leq \\ &\leq \sum |\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_l \mid r_l \in (G \setminus H)\}) \setminus X| \leq k/2\end{aligned}$$

Since $|W| < k/2$, we have $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(G)| < k/2 + k/2 = k$.

- $0 \leq |H| < k/2$. Define $h = k - |H|$; obviously, $k/2 < h \leq k$. Each cell in $C \setminus H$ (at least $3k+h$) will be scheduled once in $F \setminus W$ (referring to matrix \mathbf{S} , the cells in $C \setminus H$ appear once in the rows within W and once in rows not belonging to W). On the average, at least $\frac{3k+h}{|F \setminus W|}$ cells belonging to $C \setminus H$ will be scheduled in each time slot in $F \setminus W$ (since we have set $|C| \geq 4k$, we have $|H| = k - h$, thus $|C \setminus H| \geq 4k - (k - h)$). Note that $\frac{3k+h}{|F \setminus W|} > \frac{h}{k/2}$, because $3k + h \geq 4h$ and $|F \setminus W| < |F| = 2k$. Thus, for Lemma 11, it is possible to find a set of cells H' , s.t. $H' \subseteq (C \setminus H)$, $|H'| = h$, and $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(H') \setminus (\mathcal{I}_{\mathcal{TS}\mathcal{A}}(H') \cap W)| \leq \frac{k}{2}$ (referring to matrix \mathbf{S} , H' comprises the cells that belong to the $k/2$ rows not in W containing the largest numbers of cells in $C \setminus H$).

Finally, let $G = H \cup H'$. By construction $|G| = k$ and

$$|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(G)| = |\mathcal{I}_{\mathcal{TS}\mathcal{A}}(H)| + |\mathcal{I}_{\mathcal{TS}\mathcal{A}}(H')| < k$$

□

When the size of the switch grows, it is possible to generalize the proof for frame sizes which are integer multiples of k . We briefly sketch the proof for $L = 3k$.

Proof. [Case $S = 3$]

Also in this case we will prove that, for each $\mathcal{TS}\mathcal{A}[R]$, a subset A of R can be found, such that $|A| = k$ and $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(A)| < k$. Then, for Lemma 9 we conclude that the considered $\mathcal{TS}\mathcal{A}[R]$ is not complete.

If $L = 3k$, the average size of the image of individual cells in R through $\mathcal{TS}\mathcal{A}[R]$ cannot be larger than 3, i.e.: $E_{r_i}[|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_i\})|] \leq 3$. As a consequence, from Lemma 10, $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_i\})| \leq 3$ for at least one third of the cells in R .

Let $B \subset R$ be the set of cells for which $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_i\})| \leq 3$, $\forall r_i \in B$; for Lemma 10, $|B| \geq |R|/3 = kN_a/3$. Thus, the average number of cells in B that are scheduled in each time slot of the frame is never less than $\frac{|B|}{L} \geq \frac{kN_a}{3} \frac{1}{3k} = \frac{N_a}{9}$.

For Lemma 11, there exists a set W of time slots with $|W| < k/3$ in which a set C of cells belonging to B (i.e., $C \subseteq B$) is scheduled, and $|C| \geq |W|N_a/9$.

Let us now restrict the attention to the case $|C| \geq 81k$, which implies that $N_a > 2187$, and that k must be sufficiently large to generate enough cells to significantly fill the frame with transfers.

Let H be the set of cells $r_i \in C$ for which $\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_i\}) \subseteq W$. Note that H comprises all the cells in C for which $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{c_i\})| = 1$, and that H may be a null set.

For simplicity, we assume that $H = \emptyset$. The proof, however can be extended to the more general case. In this case each one of the $|C|$ cells in C will be scheduled once or twice in the residual time slots belonging to $F \setminus W$.

On the average, $\frac{|C|}{|F \setminus W|}$ cells belonging to C will be scheduled in each time slot belonging to $F \setminus W$. Under the assumption $|C| \geq 81k$, $\frac{|C|}{|F \setminus W|} > 27$. Thus, for Lemma 11, there exists a set of W' time slots, with $|W'| = k/3$, in which at least $9k$ cells in C are scheduled. Let Y the set comprising these cells.

Let H' be the set of cells $r_i \in Y$ for which $\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_i\}) \subseteq W \cup W'$. Note that H' comprises all the cells in Y for which $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(\{r_i\})| \leq 2$. Again, we suppose that $H' = \emptyset$.

In this case each one of the $|Y|$ cells in Y will be scheduled once in the residual time slots belonging to $F \setminus (W \cup W')$. On average $\frac{|Y|}{|F \setminus (W \cup W')|}$ cells belonging to Y will be scheduled in each time slot belonging to $F \setminus (W \cup W')$. Note that $\frac{|Y|}{|F \setminus (W \cup W')|} > 3$. Thus, for Lemma 11, there exists a set of W'' time slots, with $|W''| = k/3$, in which at least k cells in Y are scheduled. Let Y' be the set comprising these cells. Since $|Y'| = k$ and $\mathcal{I}_{\mathcal{TS}\mathcal{A}}(Y') \subseteq (W \cup W' \cup W'')$, we get $|\mathcal{I}_{\mathcal{TS}\mathcal{A}}(Y')| < k$. \square

Appendix B

Theoretical performance of an output queued switch

This appendix was referred in Section 4.12. We want to estimate the mean input-queue length and mean delay experienced by a packet in an OQ switch. We want to extend the theoretical results presented in [39] to consider also diagonal traffic.

Consider one of the N output queues in OQ switch fed by balanced i.i.d. Bernoulli traffic, i.e. all the inputs and the outputs receive the same traffic load (note that uniform, diagonal, logdiagonal and sparse traffic are balanced). The packet delay and average queue length can be estimated by modeling the output queue like an $G/D/1$ queue, where the service is deterministic with rate $\mu = 1$. The arrival process is a batch arrival with N possible arrivals if the traffic is uniform (hence, the queueing model is $Geo^N/D/1$) and with 2 possible arrivals if the traffic is diagonal (hence, the queueing model is $Geo^2/D/1$).

If x_t is the output queue occupation at time t , a_t is the number of arrivals at time t and d_t is the departure vector at time t : $d_t = 1$ if $x_t > 0$. The fundamental equation of the queue is:

$$x_{t+1} = x_t - d_t + a_t \quad (\text{B.1})$$

following the conventions in Kleinrock [53]. Let us define the z -transform:

$$X(z) = E[z^{x_t}] = \sum_{k=0}^{\infty} z^k P(x_t = k) \quad (\text{B.2})$$

$$A(z) = E[z^{a_t}] = \sum_{k=0}^{\infty} z^k P(a_t = k) \quad (\text{B.3})$$

Consider the following arrival processes:

- *single Bernoulli arrivals:*

$$P(a_t = 1) = p, \quad P(a_t = 0) = 1 - p \Rightarrow A(z) = 1 - p + pz \quad (\text{B.4})$$

- *N-batch Bernoulli arrivals:*

$$P(a_t = k) = \binom{N}{k} p^k (1-p)^{N-k} \quad 0 \leq k \leq N \quad (\text{B.5})$$

$$\Rightarrow A(z) = (1 - p/N + zp)^N \quad (\text{B.6})$$

- *2-batch Bernoulli arrivals:*

$$P(a_t = 1) = 2p(1 - p/3)/3 + (1 - 2p/3)p/3, \quad P(a_t = 2) = 2p^2/9, \quad (\text{B.7})$$

$$P(a_t = 0) = (1 - 2p/3)(1 - p/3) \quad (\text{B.8})$$

$$\Rightarrow A(z) = (1 - p + 2p^2/9) + zp(1 - 4p)/9 + z^2 2p^2/9 \quad (\text{B.9})$$

For all these arrival processes:

$$A'(1) = \left. \frac{d}{dz} A(z) \right|_{z=1} = 1 \quad A''(1) = \left. \frac{d^2}{dz^2} A(z) \right|_{z=1} = p \quad (\text{B.10})$$

It can be shown [53, Sec.5.6, Eq.5.85] that the stationary solution of the queueing system is given by:

$$X(z) = A(z) \frac{(1-p)(z-1)}{z - A(z)} \quad (\text{B.11})$$

We are interested in computing the average queue length:

$$E[x] = \lim_{z \rightarrow 1} Q'(z) \quad \text{where} \quad Q'(z) = \frac{d}{dz} Q(z) \quad (\text{B.12})$$

Note that the average delay can be computed using Little's result:

$$E[W] = E[x]/p \quad (\text{B.13})$$

We start to evaluate $Q'(z)$:

$$Q'(z) = (1-p) \frac{[A'(z)(1-z) - A(z)][A(z) - z] - A(z)(1-z)[A'(z) - 1]}{[A(z) - z]^2} \quad (\text{B.14})$$

To evaluate (B.12), we obtain an indeterminate form and so we are required to use l'Hospital's rule:

$$A(z) = 1 + p(z - 1) + A''(1)/2(z - 1)^2 + o(z - 1)^2 \quad (\text{B.15})$$

$$A'(z) = p + A''(1)(z - 1) + o(z - 1) \quad (\text{B.16})$$

After some laborious steps, we find:

$$E[x] = p + \frac{A''(1)}{2(1 - p)} \quad (\text{B.17})$$

For a $M/D/1$ queue, the arrival process is single Bernoulli, hence $A''(1) = 0$ and $E[x] = p$, which is coherent with the well-know result (note that we are assuming that a new arrival, finding the queue empty, is served at once and its delay is null).

For uniform traffic, the arrival process is an N -batch Bernoulli, hence:

$$A''(1) = p^2(N - 1)/N \quad (\text{B.18})$$

$$E[x] = p + \frac{N - 1}{N} \frac{p^2}{2(1 - p)} \quad (\text{B.19})$$

which is coherent with what was found in [39] (the term p of difference depends on the fact the we are computing the number of packet in the queue and in service).

For diagonal traffic, the arrival process is 2-batch Bernoulli, hence:

$$A''(1) = \frac{4}{9}p^2 \quad (\text{B.20})$$

$$E[x] = p + \frac{2p^2}{9(1 - p)} \quad (\text{B.21})$$