

UNIVERSITÀ DELLA CALABRIA



Dipartimento di ELETTRONICA,  
INFORMATICA E SISTEMISTICA

UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,  
Informatica e Sistemistica

Dottorato di Ricerca in  
Ingegneria dei Sistemi e Informatica  
XX ciclo

*Tesi di Dottorato*

# Scheduling Techniques In High-Speed Packet Switches

Alessandra Scicchitano





UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,  
Informatica e Sistemistica


Dottorato di Ricerca in  
Ingegneria dei Sistemi e Informatica  
XX ciclo

*Tesi di Dottorato*

Scheduling Techniques  
in High – Speed Packet Switches

Alessandra Scicchitano  
*Alessandra Scicchitano*

Coordinatore  
Prof. Domenico Talia



Supervisor  
Prof. Andrea Bianco  
*Andrea Bianco*  
Prof. Antonella Molinaro  
*Antonella Molinaro*

DEIS

DEIS – DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA  
Novembre 2007

Settore Scientifico Disciplinare: ING-INF/05

Ad Andrea ed alla mia famiglia



---

## Prefazione

Packet switches are at the heart of modern communication networks. Initially deployed for local- and wide-area computer networking, they are now being used in different contexts, such as interconnection networks for High-Performance Computing (HPC), Storage Area Networks (SANs) and Systems-on-Chip (SoC) communication. Each application domain, however, has particular requirements in terms of bandwidth, latency, scalability and delivery guarantee. These requirements must be carefully taken into account and have a major impact on the design of the switch.

We propose novel scheduling techniques that enable the construction of distributed (multi-chip) schedulers for large crossbars, develop a scheme for integrated scheduling of unicast and multicast traffic and study flow-control mechanisms that allow switches to achieve lossless behavior while providing fine-grained control of active flows. We also present a comparison between synchronous and asynchronous systems showing that asynchronous switching seems a competitive solution with respect to the more traditional synchronous approach.

November 2007

Rende,  
*Alessandra Scicchitano*





---

# Indice

<b>1</b>	<b>Introduction</b>	9
1.1	Background	9
1.2	Contributions	10
1.3	Outline of the Thesis	10
<b>2</b>	<b>Packet Switch</b>	13
2.1	General Architecture	13
2.1.1	Switching Fabric	14
2.2	Traffic conditions for a packet switch	16
2.3	Buffering Strategies	17
2.3.1	Output-queued (OQ)	17
2.3.2	Input-queued (IQ)	18
2.3.3	IQ switches with Virtual Output Queueing (VOQ)	19
2.3.4	Combined Input-Output-Queued (CIOQ) Switches	21
2.4	Scheduling Unicast Traffic in IQ Switches	22
2.4.1	Optimal Scheduling Algorithm	22
2.4.2	Parallel Iterative Matching Algorithms	22
2.4.3	Sequential Matching Algorithms	25
2.5	Scheduling Multicast Traffic in IQ Switches	26
2.5.1	Queueing	26
2.5.2	Scheduling	27

---

## Parte I A Switching Architecture for Synchronous IQ Switch

---

<b>3</b>	<b>Distributed Implementation of Crossbar Schedulers</b>	31
3.1	Introduction	31
3.2	Two- vs. three-phase algorithms	31
3.3	Round Trip Time Latencies	32
3.4	Multi-chip implementation	33

<b>4</b>	<b>Distributed Scheduler under Unicast Traffic Conditions</b> . . . .	35
4.1	Scheduling algorithms with RTT latencies . . . . .	35
4.2	Synchronous Round Robin . . . . .	36
4.3	Performance results . . . . .	38
<b>5</b>	<b>Distributed Scheduler under Multicast Traffic Conditions</b> . .	41
5.1	Multicast scheduling algorithms . . . . .	41
5.2	Improved Multicast Round Robin . . . . .	42
5.3	Performance results . . . . .	43
<b>6</b>	<b>Distributed Integration of Unicast and Multicast Scheduling</b>	49
6.1	Introduction . . . . .	49
6.2	Frame Integration . . . . .	50
6.3	Description of Scheduler . . . . .	50
6.4	Dynamic Frame . . . . .	50
6.5	Simulation Result . . . . .	51

---

## Parte II A Switching Architecture for Asynchronous SANs

---

<b>7</b>	<b>Introduction to Storage Area Networks</b> . . . . .	63
7.1	Storage Area Networks . . . . .	63
7.2	Networking Technologies for SANs . . . . .	65
7.2.1	Fibre Channel . . . . .	65
7.2.2	Credit-based flow control . . . . .	66
<b>8</b>	<b>The Switching Architecture</b> . . . . .	69
8.1	System Overview . . . . .	69
8.2	Line-cards . . . . .	70
8.3	Switching fabric . . . . .	70
8.3.1	Improvement of multicast scheduling . . . . .	71
8.4	Control mechanisms for lossless delivery . . . . .	73
<b>9</b>	<b>Performance Results</b> . . . . .	75
9.1	Performance results under multicast traffic conditions . . . . .	75

---

## Parte III Synchronous vs Asynchronous Switching

---

<b>10</b>	<b>Synchronous versus Asynchronous under Multicast Traffic</b> .	83
10.1	Introduction . . . . .	83
10.2	System Model . . . . .	84
10.3	Performance results . . . . .	86

---

**Parte IV Conclusion**

---

**11 Conclusion** ..... 93

**Riferimenti bibliografici** ..... 95



## Introduction

### 1.1 Background

The history of packet-switched networks dates back to the '60s, when deployment of the ARPANET, ancestor of the Internet, was initiated. In the '90s the Internet became a global and ubiquitous networking infrastructure, used for business, entertainment and scientific purposes. Since then, the bandwidth demand of the Internet community has been steadily increasing at exponential rates. To satisfy it, researchers and engineers have studied extensively the design of high-performance switching fabrics, that are at the heart of Internet routers. Today's commercial Internet routers offer aggregate bandwidths on the order of terabits per second and employ sophisticated algorithms for packet buffering, processing and scheduling.

The success of this technology has led researchers to investigate its usage in other domains, where the communication subsystem has become a primary performance bottleneck. Packet switching is being used to build interconnection networks for High-Performance Computing (HPC) systems, where a large number of computing nodes and memory banks must be interconnected. It is replacing the traditional bus-based interconnection between servers and storage devices, giving birth to Storage Area Networks (SANs).

While the benefits of using packet switching in these domains have long been recognized, it is important to remember that each of them has its specific set of requirements, significantly different from those typical of computer networks. Table 1.1 summarizes the requirements for packet switches used in computer networks as well as in the two other domains we are considering. The most significant differences are in terms of latency, aggregate bandwidth and delivery guarantee.

Moreover, current technology trends are playing a significant role in the design of packet switches. Issues such as power consumption, chip I/O bandwidth limitations and packaging constraints are becoming primary concerns for designers.

	<b>IP Routers</b>	<b>Fibre Channel SAN Switches</b>	<b>HPC Interconnects</b>
<b>Throughput</b>	<i>Very Important</i>	<i>Very Important</i>	Moderately Important
<b>Latency</b>	Not Important	Moderately Important	<i>Very Important</i>
<b>Delivery Guarantee</b>	Can Tolerate Small Losses	<i>Losses not Acceptable</i>	<i>Losses not Acceptable</i>
<b>Line Rate/ Port Count</b>	< 10 Gb/s ~ 100 ports	< 10 Gb/s ~ 100 ports	$\geq 40$ Gb/s ~ 1000 ports

**Tabella 1.1.** Requirements of domain-specific interconnection networks.

## 1.2 Contributions

In this thesis we present new scheduling algorithms, aimed at distributed switch implementation and Storage Area Networks respectively. We discuss how the specific requirements of the respective domains and current technology trends have influenced the design. More importantly, we present some of the architectural innovations that allow them to satisfy the demanding needs of their operating environments.

In Part I we study the distributed implementation of a crossbar scheduler, considering the effect of big latencies on its performance. Referring to this, we present new scheduling algorithms, both for unicast and multicast, that achieve a level of performance that is close to that of a single-chip implementation.

In Part II we discuss the performance of a Storage Area Network under multicast traffic conditions. The architecture presents a number of important features, such as an asynchronous design and the presence of a central arbiter that allow the switch to achieve lossless behavior and isolate congesting flows.

Finally in Part III, asynchronous and synchronous switch architectures are compared under multicast traffic.

## 1.3 Outline of the Thesis

The thesis is organized as follows:

**Chapter 2** introduces basic concepts and the terminology used in the rest of the thesis. It provides an overview of switching architectures and a brief survey of scheduling algorithms.

**Chapter 3** introduces the problem of big latencies for interconnections in multichip implementation.

**Chapter 4** is devoted to the specific problem in unicast scenario: how to build schedulers for large crossbars using multiple chips and overcoming the delay and I/O bandwidth limitations caused by distribution.

**Chapter 5** described how to solve the same problem but in a multicast scenario.

**Chapter 6** addresses the problem of scheduling unicast and multicast traffic concurrently over a single fabric, achieving high overall performance and providing fairness guarantees.

**Chapter 7** opens Part II of the thesis, describing the evolution of the server-storage interface and illustrating how Storage Area Networks improve the organization of storage resources.

**Chapter 8** introduces the switching architecture for SANs, focusing in particular on the mechanisms used to achieve loss-free operation and isolate congesting flows.

**Chapter 9** contains a simulation-based study of system performance under multicast traffic, analyzing the effects of system parameters (buffer sizes, fabric and link speed-up) and traffic characteristics (uniformity, packet size distribution).

**Chapter 10** studies differences between synchronous and asynchronous performance under multicast traffic.

**Chapter 11** draws conclusions of the thesis.





## Packet Switch

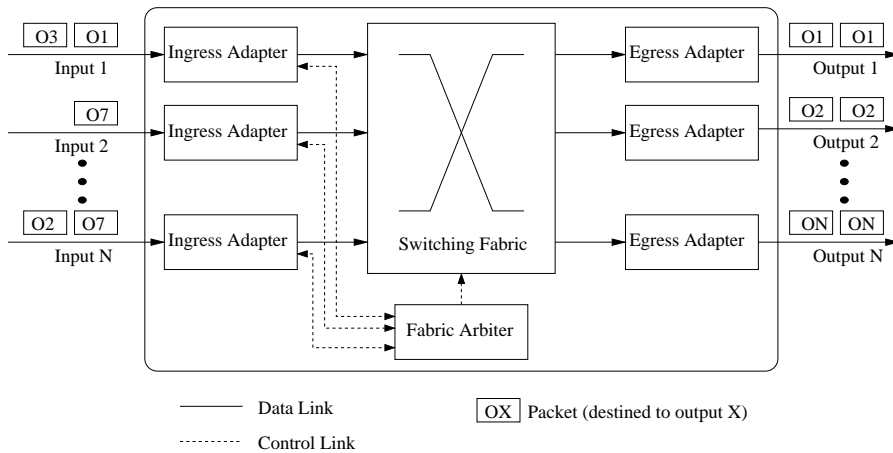
In this chapter we introduce the basic concepts and the terminology used in the rest of the thesis. We first present the general architecture of a packet switch and discuss the main distinguishing feature: buffer placement. After an overview of output-queued (OQ), input-queued (IQ) and combined input-output-queued (CIOQ) switches, we focus on the problem of scheduling unicast and multicast traffic in IQ switches. We provide a survey of the most popular scheduling algorithms and discuss their characteristics in terms of performance and complexity.

Packet switching is a broad field, which has been studied extensively for decades. A comprehensive treatment of the topic can be found in [1], [2] and [3].

### 2.1 General Architecture

A *packet switch* is a network device that receives packets on *input ports* and forwards them on the appropriate *output ports*. Figure 2.1 shows the architecture of a packet switch with  $N$  input/output ports. Packets are received on an input port and enter an *ingress adapter*, where they are stored (if necessary) and processed. Processing may include look-up of the destination port, recalculation of header fields (TTL, CRC, etc.) and filtering. Packets are then transmitted through the *switching fabric* and reach the *egress adapters*, where they are stored (if necessary) and prepared for transmission on the output links. If the switching fabric operates only on fixed-size data units, variable-size packets have to be segmented on the ingress adapter and reassembled on the egress adapter. Usually an ingress adapter is coupled to an egress adapter and they physically reside on a single board called *linecard* that can host multiple bi-directional ports.

A switch is *synchronous* if the linecards and the fabric are coordinated by mean of global clock signal and all ingress adapters start cell transmission at the same time. If the switch is *asynchronous*, on the contrary, the linecards



**Figure 2.1.** General architecture of a packet switch.

and the fabric work on independent clock domains and transmission from different ingress adapters is not coordinated. In general synchronous switches internally operate on fixed-size cells, whereas asynchronous switches may natively support variable-size packets. Synchronous architectures are more popular because synchronicity simplifies many aspects of the design and implementation of the device. However, asynchronous switches have advantages as well, so they are being actively researched [4, 5, 6]. In the rest of this chapter we will implicitly refer to synchronous, cell-based switches.

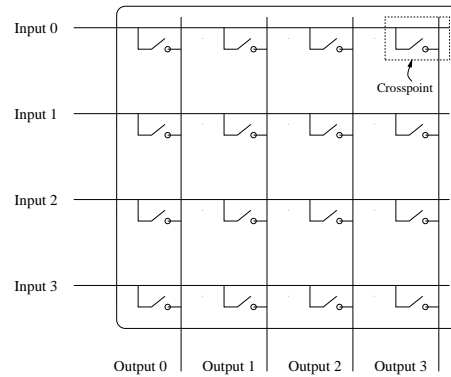
### 2.1.1 Switching Fabric

The switching fabric sets up connections between ingress and egress adapters. It is *non-blocking* if a connection between an idle input and an idle output can always be set-up, regardless of which other connections have already been established. This is a very desirable property, because it helps the switch in forwarding multiple packets concurrently, thus increasing throughput and reducing latency.

The fabric may run at a higher data rate than the linecards; in this case the ratio between the data rate of the fabric ports and that of the switch ports is called *speed-up*. For example, in a synchronous switch with speed-up two, at every time slot ingress/egress adapters can transmit/receive two cells to/from the fabric. When speed-up is used, the egress adapters can receive cells from the fabric at a higher rate than they can transmit on the output links, so they need buffers to temporarily store cells in excess. The term speed-up generally refers to the case in which both input and output fabric ports run faster than the switch ports; however, it is possible to have output speed-up only, i.e.

to have only fabric output ports run at a higher data rate. Speed-up on the fabric inputs only is possible but has no practical use.

### Crossbar



**Figure 2.2.** A  $4 \times 4$  crossbar.

The crossbar is a very simple fabric that directly connects  $n$  inputs to  $m$  outputs, without intermediate stages. From a conceptual point of view, it is composed by  $n + m$  lines, one for each input and one for each output, and  $n \times m$  *crosspoints*, arranged as depicted in Figure 2.2. Input  $i$  is connected to output  $j$  if crosspoint  $(i, j)$  is *closed*.

Every output can be connected to only one input at a time, i.e. at most one crosspoint can be closed on a column. However, one input can be connected to multiple outputs at the same time by closing the corresponding crosspoints on the input row. In this case the signal at the input port is replicated to all the outputs for which the crosspoint is closed. The fabric has intrinsic support for *multicast* (one-to-many) communication. The crossbar is obviously non-blocking: an idle input (output) has all crosspoints its row (column) open, thus it is enough to close the crosspoint at the intersection to connect them.

The simplicity of the crossbar and its non-blocking property make it a very popular choice for packet switches. The main drawback is its intrinsic quadratic complexity, due to the presence of  $n \times m$  crosspoints. Crossbars implemented on a single chip may also be limited by the amount of I/O signals that must be mapped to chip pins. However, it is possible to build a large multi-chip crossbar by properly interconnecting smaller single-chip ones [7]. The complexity in terms of gates remains quadratic.

## 2.2 Traffic conditions for a packet switch

The arrival of packets at the switch inputs can be modeled with a discrete-time stochastic process. At every timeslot at most one fixed-size data unit, called *cell* can arrive on each input. Variable-size packets can be considered as “bursts” of cells received at the same input in subsequent timeslots and directed to the same output.

We denote with  $\lambda_{ij}$  the average arrival rate on input  $i$  of cells directed to output  $j$ , normalized to the input/output link speed. The *offered load from input  $i$*  is the (normalized) rate at which cells enter the switch on input  $i$  and is represented by the term  $\sum_{j=1}^N \lambda_{ij}$ , where  $N$  is the number of input/output ports. Conversely, the *offered load to output  $j$*  is the (normalized) rate at which cells destined to output  $j$  enter the switch and is equal to the sum  $\sum_{i=1}^N \lambda_{ij}$ .

Traffic is *admissible* if no input/output links are overloaded, i.e. if the arrival rate at the inputs and the offered load to the outputs are less than or equal to the capacity of the input/output links. Formally, the admissibility conditions can be stated as:

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

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

In these conditions it is theoretically possible for the switch to forward to the outputs all the cells it receives on the inputs in finite time.

Traffic is *uniform* if a cell entering the switch can be directed to any output with equal probability:

$$\lambda_{ij} = 1/N \quad \forall i, j$$

It is *independent and identically distributed (i.i.d.)*, also called *Bernoulli*, if the probability that a cell arrives at an input in a certain timeslot:

- is identical to and independent from the probability that a cell arrives at the same input in a different timeslot AND
- is independent from the probability that a cell arrives at another input.

The performance of a packet switch is mainly measured in terms of *throughput* and *latency*. Throughput is the (normalized) rate at which the device forwards packets to the outputs, latency is the time taken by a packet to traverse the switch. A switch achieves 100% throughput if it is able to sustain an offered load to all outputs equal to 1, under the hypothesis that traffic is admissible.

### Multicast traffic

Under unicast traffic conditions, every packet received on input port has direct to just one output port. In multicast traffic, conditions are different.

Traffic generated by a single source and directed to multiple destinations is called *multicast*. The set of outputs a multicast cell is destined to is called the *fanout set* and its cardinality the *fanout*<sup>1</sup>. For clarity, we distinguish between the *input cell* that is transmitted to the switching fabric and the *output cells* that are generated by the replication process.

A scheduling discipline is termed *fanout splitting* if it allows partial service of an input cell, i.e. if the associated set of output cells can be transferred to the outputs over multiple timeslots. *No fanout splitting* disciplines, on the contrary, require all the output cells associated to an input cell to be delivered at the outputs in the same timeslot. Fanout splitting offers a clear advantage because it allows the fabric to deliver in every timeslot as many cells as possible to the outputs, at the price of a small increase of implementation complexity.

The *residue* is the set of all output cells that lose contention for output ports in a timeslot and have to be transmitted in subsequent timeslots.

## 2.3 Buffering Strategies

Due to traffic independence, the switch may receive in the same time slot multiple cells directed to the same output. In this case there is a *conflict* between inputs caused by *output contention*. It is not possible to forward one of the contending cells and discard all the other, because the drop rate would be unacceptable for any practical application. Therefore, the switch is endowed with internal buffers to store cells that cannot be transmitted immediately on the output link. The buffering strategy, mainly if the cells are buffered before being transferred through the switching fabric or after, is a major architectural trait and strongly influences performance, scalability and cost of a switch [8].

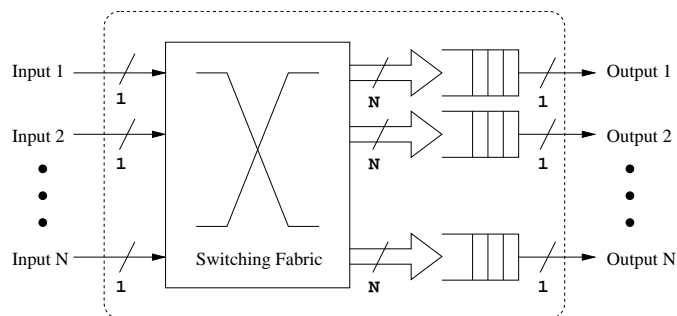
### 2.3.1 Output-queued (OQ)

In OQ switches all cells arriving at the fabric inputs are immediately transferred through the switching fabric and stored at the outputs. At every timeslot up to  $N$  cells directed to the same output can arrive, so the fabric must operate with speed-up  $S = N$  and the memory bandwidth at each egress adapter must be equal to  $N$  times the line rate of the switch ports<sup>2</sup> (Figure 2.3).

If multiple cells are buffered at an egress adapter, it is necessary to decide in which order they will be transmitted on the output link. This choice allows the switch to prioritize different flows but does not have an impact on throughput. The OQ switch offers ideal performance, i.e. it achieves 100% throughput under any traffic pattern.

<sup>1</sup> The term “fanout” is often used to refer also to the set itself.

<sup>2</sup> For simplicity we only consider memory *write* bandwidth.

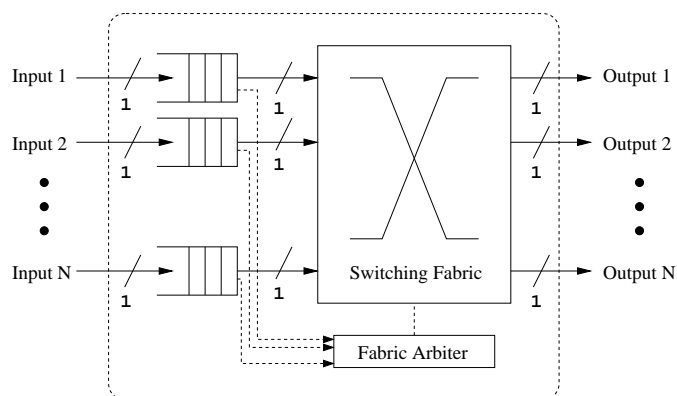


**Figure 2.3.** An Output-queued switch.

The problem with OQ switches is scalability: fabric speed-up and, above all, egress adapters memory bandwidth, grow linearly with  $N$ . As the bandwidth offered by commercial memories is on the same order of link rates, the OQ architecture is a suitable choice only for systems with a small number of ports or low link rates.

### 2.3.2 Input-queued (IQ)

In IQ switches the fabric transfers to the egress adapters only cells that can be transmitted immediately on the output links. Those that are blocked because of output contention are buffered on the ingress adapters (Figure 2.4).



**Figure 2.4.** Input-queued switch.

This strategy has the following crucial consequences:

- buffers are not needed on the egress adapters, because at every timeslot the cell received from the switching fabric can be transmitted immediately on the output link<sup>3</sup>;
- the switching fabric does not need speed-up, because it must be able to deliver at most one cell per timeslot to each egress adapter;
- the memory bandwidth of the buffers on the ingress adapters is equal to the switch ports line rate, irrespective of  $N$ , because at most one cell per timeslot arrives at each input;
- a scheduler is required to decide which among multiple cells contending for the same output will be transferred; the fabric must be configured accordingly.

In the simplest case, arriving cells are stored in FIFO queues and each ingress adapter can only transmit the cell that is at the head of its queue. This constraint leads to a phenomenon called “Head-of-the-line (HOL) Blocking”: a cell that is at the head of its input queue and cannot be transferred because of output contention blocks all the other cells in the same queue. Blocked cells may be destined to outputs for which no other input is contending, so the opportunity to transfer a cell is lost. HOL-blocking can severely degrade performance: for large values of  $N$  it limits switch throughput to about 58% under uniform i.i.d. traffic [8].

This level of performance is not acceptable, so in the past there have been many attempts to overcome the problem, in general by relaxing the FIFO constraint and allowing the scheduler to consider multiple cells from the same queue. In recent years increased CMOS densities have made feasible a new queueing architecture, called Virtual Output Queueing, that completely eliminates HOL blocking and allows IQ switches to achieve high performance.

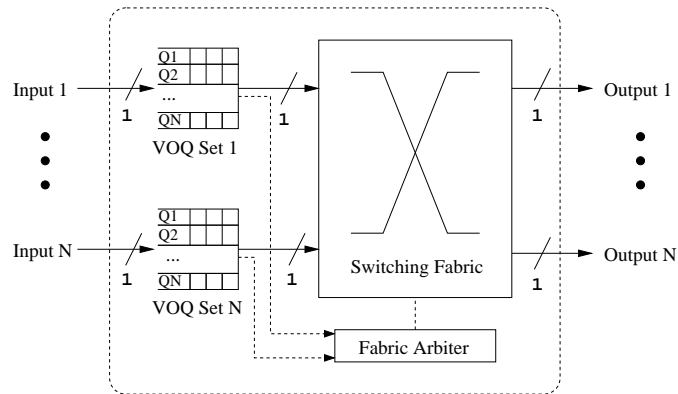
### 2.3.3 IQ switches with Virtual Output Queueing (VOQ)

Virtual Output Queues (VOQs) are sets of independent FIFO queues, each of which is associated to a specific output [9]. In an IQ switch it is possible to avoid HOL-blocking by deploying a set of  $N$  VOQs on each ingress adapter (Figure 2.5). With VOQs, cells destined to different outputs can be served in any order and do not interfere with each other; cells destined to the same output, on the contrary, are served with a FIFO policy to preserve the ordering of packets belonging to the same flow.

At every timeslot the scheduler must decide which cells to transfer through the switching fabric, subject to the constraints that at most one cell can depart from an ingress adapter and at most one cell can be delivered to an egress adapter. The problem is equivalent to calculating a matching on a bipartite graph, as illustrated in Figure 2.6. Nodes on the left and right side represent

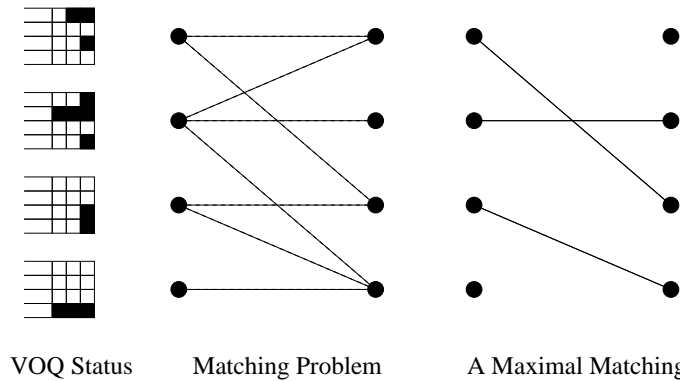
---

<sup>3</sup> We neglect flow-control issues and assume that a cell can always be transmitted on an idle output link.



**Figure 2.5.** Input-queued switch with Virtual Output Queues.

fabric inputs and outputs respectively; dashed lines (edges) represent non-empty VOQs, i.e. cells that can be chosen for transfer. A *matching* is a set of edges such that each input is connected to at most one output and each output to at most one input.



**Figure 2.6.** A Bipartite Graph Matching (BGM) problem.

A matching is *maximum size* if it contains the highest number of edges among all valid matchings; it is *maximal* if it is not possible to add new edges without removing previously inserted ones. For instance, the matching shown in Figure 2.6 is maximal but not maximum: no edges can be added, but it is easy to verify that there exists valid matchings with four edges. Edges can be assigned weights, such as the number of cells enqueued in the corresponding VOQ, or the time the cell at the head-of-the-line has been waiting for service.



If weights are defined, the *Maximum Weight Matching (MWM)* is the one that maximizes the sum of the weights associated to the edges it contains.

IQ switches with VOQs can achieve 100% throughput under any i.i.d. traffic pattern, but only if very sophisticated scheduling algorithms are employed [10]. These algorithms are in general difficult to implement in fast hardware and too complex to be executed in a single timeslot. However, as we will discuss in Section 2.4, a number of heuristic matching algorithms that achieve satisfactory performance with reasonable complexity have been devised. Therefore input-queueing with VOQs is today the preferred architecture for the construction of large, high-performance packet switches. From this point on, when discussing IQ switches we will implicitly assume that VOQs are present.

### 2.3.4 Combined Input-Output-Queued (CIOQ) Switches

OQ and IQ switches represent two diametrically opposing points in the trade-off between speed-up and scheduling complexity. The former employ maximum speed-up but require no scheduling, the latter run without speed-up but need complex schedulers. CIOQ switches (with VOQs) represent an intermediate point: they buffer packets both at the inputs and at the outputs, employ moderate speed-up  $S$  ( $1 \leq S \leq N$ ) and use simpler schedulers.

Early simulation studies of CIOQ switches showed that, under a variety of switch sizes and traffic patterns, a small speed-up (between two and five) leads to performance levels close to those offered by OQ switches. These hints led a number of researchers to analytically investigate the maximum performance achievable by CIOQ switches. Among the many results that were published, these are particularly significant:

- With a speed-up  $S = 2$  and proper scheduling algorithms, a CIOQ switch can exactly *emulate* an OQ switch, for any switch size and under any traffic pattern [11, 12]. “Emulating” means producing exactly the same cell departure process at the outputs given the same cell arrival process at the inputs.
- A CIOQ switch employing any maximal matching algorithm with a speed-up of two achieves 100% throughput under any traffic pattern, under the restriction that no input or output is oversubscribed and that the arrival process satisfies the strong law of large numbers [13].

These results prove that with moderate speed-up the performance of an IQ switch can be dramatically improved and that it can even reach the performance of an OQ switch if proper scheduling is used. A small fractional speed-up ( $S < 2$ ) is also typically used to compensate for various forms of overhead, such as additional headers that must be internally prepended to cells and padding imposed by segmentation [14].

## 2.4 Scheduling Unicast Traffic in IQ Switches

### 2.4.1 Optimal Scheduling Algorithm

The optimal scheduling algorithms for an IQ switch, i.e. the one that maximizes throughput, is the Maximum Weight Matching (MWM), when queue lengths are used as weights [15]. McKeown et al. noted that, with this choice of the weights, specific traffic patterns can lead to permanent starvation of some queues [10]. However, they also proved that 100% throughput is still achieved for any i.i.d. traffic pattern if the ages of HOL cells are used as weights; in this case starvation cannot happen. The most efficient known algorithm for calculating the MWM of a bipartite graph converges in  $O(N^3 \log N)$  time [16]. Despite polynomial complexity, this algorithm is not practical for high-performance packet switches, because it is difficult to implement in fast hardware and cannot be executed in the short duration of a timeslot. For this reason, a number of heuristic algorithms have been developed.

### 2.4.2 Parallel Iterative Matching Algorithms

Parallel iterative matching algorithms are the most popular class of heuristic matching algorithms. All inputs in parallel try to match themselves to one output by using a request-grant protocol. VOQ selection at the inputs and contention resolution at the outputs are performed by *arbiters* (also called *selectors*) using round-robin or random criteria. The process is iterated multiple times, until a maximal matching is obtained or the maximum number of iterations is reached. On average these algorithms converge in  $\log_2 N$  iterations, but in the worst case they can take  $N$ .

#### PIM

PIM [17] (Parallel Iterative Matching) is one of the first parallel iterative matching algorithms that have been proposed. In every timeslot the following three *phases* are executed and possibly repeated multiple times:

1. *Request*: every unmatched input sends a request to every unmatched output for which it has a queued cell.
2. *Grant*: every output that has been requested by at least one input *randomly* selects one to grant.
3. *Accept*: if an input receives more than one grant, it selects *randomly* one to accept.

The main disadvantage of PIM is that it does not perform well, as it achieves only 63% throughput with a single iteration under uniform i.i.d. traffic. Moreover, it employs random selection, which is difficult and expensive to perform at high speed and can cause unfairness under specific traffic patterns [18].

## RRM

RRM (Round-Robin Matching) [18] addresses some of the drawbacks of PIM by replacing random selection with round-robin. The selection logic at each input and output is composed by a round-robin selector and a pointer. Pointers at the outputs are named *grant pointers*, whereas those at the inputs *accept pointers*.

Every iteration of RRM entails the following three phases:

1. *Request*: every unmatched input sends a request to every output for which it has a queued cell.
2. *Grant*: every output that has been requested by at least one input selects one to grant in round-robin order, starting from the position indicated by the grant pointer. The pointer is advanced (modulo  $N$ ) to one input beyond the one just granted.
3. *Accept*: if an input receives more than one grant it selects one to accept in round-robin order, starting from the position indicated by the accept pointer. The pointer is advanced (modulo  $N$ ) to one output beyond the one just accepted.

The performance of RRM is very close to that of PIM, so still quite poor.

## *i*-SLIP

*i*-SLIP [19] is an improvement of RRM that, with an apparently minor modification, achieves much higher performance. The three phases are modified as follows:

1. *Request*: same as RRM.
2. *Grant*: every output that has been requested by at least one input selects one to grant in round-robin order, starting from the position indicated by the pointer. The pointer is advanced (modulo  $N$ ) to one input beyond the one just granted *if and only if the released grant is accepted in the accept phase*.
3. *Accept*: same as RRM.

Moreover, the grant and accept pointers are updated only in the first iteration; a detail that is crucial to prevent starvation of any VOQ under any traffic pattern.

*i*-SLIP performs extremely well: under uniform i.i.d. traffic it achieves 100% throughput with a single iteration, because it guarantees *desynchronization* of the grant pointers. When the switch is loaded at 100% and traffic is uniform i.i.d., all VOQs are backlogged. Assume that the grant pointers at multiple outputs point to the same input, i.e. they are *synchronized*. The input receives multiple grants, accepts one and moves the accept pointer. Thanks to the modification of the grant phase, only one of the grant pointers (the one corresponding to the grant that has been accepted) is moved and leaves the

group. For the same reason, at most one new grant pointer can join the group. It is possible to prove that, after a transient period, all grant pointers point to different inputs, regardless of their initial position. A *maximum* matching is produced at every timeslot and 100% throughput is achieved. Desynchronization is preserved as long as all VOQs are non-empty, because all the released grants are accepted and so all the grant pointers move “in lockstep”.

Another important feature of *i*-SLIP is that it is fair and starvation free, i.e. it does not favor some flows over others and guarantees that a cell at the head of a VOQ will be served within finite time.

## DRRM

DRRM [20] (Dual Round-Robin Matching) is a further variant of *i*-SLIP that achieves similar performance with one less phase and less information exchange between the input and the outputs.

The two phases performed in each iteration are:

1. *Request*: every unmatched input selects *one* unmatched output to request in round-robin order, starting from the position indicated by a *request pointer*. In the first iteration, the pointer is updated to one position beyond the input just requested (modulo  $N$ ) if and only if a grant is received in the *grant* phase.
2. *Grant*: each output that has been requested by at least one input selects one to grant in round-robin order, starting from the position indicated by a *grant pointer*. In the first iteration the pointer is updated to one position past the input just granted (modulo  $N$ ).

A grant phase is not required because each input requests only one output, so it can receive at most one grant, which is automatically accepted.

DRRM achieves 100% throughput under uniform i.i.d. traffic because in this situation request pointers (moved only if a grant is received) desynchronize.

Figure 2.7 shows the operation of the DRRM algorithm for a  $4 \times 4$  switch. At the end of the first iteration all pointers (except R4 and G1) are moved forward by one position. As the matching is maximal, it is not necessary to perform additional iterations.

## FIRM

FIRM [21] is an improvement of *i*-SLIP that achieves lower average latency by favoring FCFS order of arriving cells. It does so by introducing a minor modification in the pointer update rule of the grant phase of *i*-SLIP: *in the first iteration, if a grant is not accepted, the grant pointer is moved to the granted input*. The authors also show that this modification reduces the maximum waiting time for any request from  $(N - 1)^2 + N^2$  to  $N^2$ .

A similar modification has been proposed for DRRM in [22].

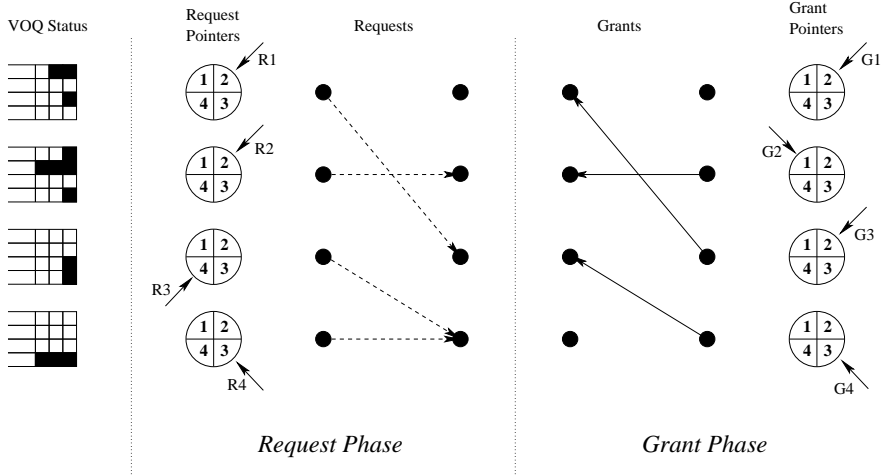


Figure 2.7. The behavior of the DRRM algorithm in a sample scenario.

### Weighted Algorithms

As an attempt to approximate the behavior of MWM and improve performance under non-uniform traffic, heuristic iterative weighted algorithms have been developed. Among these are *i*-OCF (Oldest Cell First), *i*-LQF (Longest Queue First) and *i*-LPF (Longest Port First), proposed by Mekittikul and McKeown [23].

#### 2.4.3 Sequential Matching Algorithms

Sequential scheduling algorithms produce a maximal matching by letting each input add an edge at a time to an initially empty matching.

RPA [24] (Reservation with Pre-emption and Acknowledgement) and RRGs [25] (Round Robin Greedy Scheduler) are examples of sequential matching algorithms. An input receives a partial matching, adds an edge by selecting a free output and passes it on to the next input. Inputs considered first are favored, because they find most outputs still available. To avoid unfairness, the order in which inputs are considered is rotated at every timeslot. These algorithms always produce a maximal matching, are fair and can be pipelined to improve the matching rate. However, they require strong interaction among the inputs and introduce latency at low load when pipelined.

The Wavefront Arbiter [26] (WFA) is another popular sequential arbiter. The status of all the  $N^2$  VOQs of the system is represented in a  $N \times N$  request matrix  $R$ :  $R_{i,j} = 1$  if input  $i$  has a cell destined to output  $j$ , 0 otherwise. Sets of VOQs that are positioned on a diagonal of the matrix are conflict-free, because

they correspond to cells enqueued at different inputs and destined to different outputs. Hence it is possible to produce a matching by sequentially “sweeping” all the diagonals of the request matrix, excluding input and outputs that have already been matched. WFA is fast, simple and offers good performance; however, it suffers from some minor fairness and implementation issues [7].

## 2.5 Scheduling Multicast Traffic in IQ Switches

In an IQ switch replication can be achieved simply by transmitting cells through the switching fabric multiple times, one for every egress adapter that must be reached. However, the crossbar has intrinsic multicasting capabilities and can replicate a cell to multiple outputs in a single timeslot. A scheduler that takes advantage of this feature can reduce the latency experienced by cells and the load on the fabric input ports, which are occupied for only one timeslot.

In this section we briefly introduce the problem of scheduling multicast traffic and present some of the most popular scheduling algorithms.

### 2.5.1 Queueing

A multicast cell can be destined to any subset of the  $N$  outputs, so the number of possible fanout sets is  $2^N - 1$ . Even for moderate values of  $N$  it is not practically feasible to provide a dedicated queue to cells with the same fanout set, therefore HOL-blocking cannot be completely eliminated. Indeed, most architectures store cell arriving on an ingress adapter in a single queue and serve them in FIFO order.

To alleviate HOL-blocking, in [27] the authors propose a windowing scheme that allows the scheduler to access any cell in the first  $L$  positions of the queue. This scheme offers throughput improvements, but requires random-access queues, which are complex to implement. Moreover, it is clearly not effective under bursty traffic.

In [28] and [29] the benefits that can be gained by using a small number of FIFO queues at each ingress adapter are investigated. When multiple queues are present, it is necessary to define a queueing policy. Static queueing policies always enqueue cells with a given fanout in the same queue, whereas dynamic policies may enqueue them in different ones, depending on status parameters such as queue occupancy. Static policies lose effectiveness when few flows are active, because most of the available queues may remain empty, whereas dynamic policies lead to out-of-order delivery.

In [32] maximum switch performance is analyzed, under the hypothesis that a queue is provided for every possible fanout set. The results of this work have great theoretical interest, because they show that an IQ switch is not able to achieve 100% throughput under arbitrary traffic patterns, even if it employs this ideal queueing architecture and the optimal scheduling discipline.

### 2.5.2 Scheduling

The problem of scheduling multicast traffic in an input-queued switch has been addressed by a number of theoretical studies. In [33] and [34] the performance of various scheduling disciplines (such as random or oldest-cell-first) is analyzed under different assumptions. Work in [35] studies the optimal scheduling policy, obtaining it for switches of limited size (up to three inputs) and deriving some of its properties in the general case.

In [36] the authors take a more practical approach: they specifically target the design of efficient and implementable scheduling algorithms when FIFO queueing is used and fanout splitting allowed. They provide important insight on the problem and propose various solutions with different degrees of performance and complexity. An important observation is that at any timeslot, given a set of requests, all *non-idling* policies (those that serve as many outputs as possible) transmits cells to the same outputs and leave the same residue. What differentiates one policy from the other is residue distribution, i.e. the criteria with which the set of output cells that have lost contention is partitioned among the inputs. A *concentrating* policy assigns the residue to as few inputs as possible. Policies exhibiting this property serve in each timeslot as many HOL cells as possible, helping new cells to advance to the head of the queue. As new cells may be destined to idle outputs, throughput is increased. Actually a proof is given that for a  $2 \times N$  switch a concentrating policy is optimal, but it cannot be extended to switches of arbitrary size.

The first proposed algorithm, called “Concentrate” implements a purely concentrating policy. However, the authors note that the algorithm suffers from fairness issues, as it can permanently starve queues, so they proceed with the design of TATRA, a concentrating algorithm with fairness guarantees. As TATRA is difficult to implement in hardware, they further propose the Weight Based Algorithm (WBA). WBA is a heuristic algorithm that approximates concentrating behavior by favoring cells with small fanout and guarantees fairness by giving priority to older cells. The algorithm works as follows:

1. At the beginning of every cell time each input calculates the *weight* of the cell at the head of its queue, based on the age of the cell (the older, the heavier) and the fanout (the larger, the lighter).
2. Each input submits a weighted request to all the outputs that it wishes to access.
3. Each output independently grants the input with the highest weight; ties are broken randomly.

In the specific implementation shown in the paper, the weight is calculated as  $W = \alpha A - \phi F$ , where  $A$  is the age (expressed in number of timeslots),  $F$  is the fanout and  $\alpha$  and  $\phi$  are multiplication factors that allow tuning of the scheduler for performance or fairness. Large  $\alpha$  implies that older cells are strongly favored, improving fairness, while large  $\phi$  penalizes cells with large fanout, exalting the concentrating property and thus improving performance.

Calculations show that a cell has to wait at the head of the queue for no longer than  $(N(\phi/\alpha + 1) - 1)$  timeslots. WBA can be easily implemented in hardware, as reported in the paper.



**A Switching Architecture for Synchronous IQ  
Switch**



## Distributed Implementation of Crossbar Schedulers

### 3.1 Introduction

IQ switches are suited for several application domains, such as traditional routers/switches, SANs (Storage Area Networks), and HPC (High-Performance Computing) interconnects; in most of these application domains, a large number of ports and high line rates are dominant and exceed single-chip implementation limits. Multichip implementation is limited by power density, gate count, pin count, I/O bandwidth and wiring, due to the high degree of connectivity between the input and output selectors [14]. In this thesis we present new scheduling algorithms enabling the construction of schedulers for large switches, while achieving a level of performance that is close to that of a single-chip implementation.

### 3.2 Two- vs. three-phase algorithms

A very popular solution to determine a heuristic matching is to devise parallel, iterative matching algorithms based on a three-phase (request-grant-accept) [19, 21] or two-phase (request-grant) [44, 36, 20] scheme. Iterative matching algorithms can be classified into two- and three-phase algorithms according to the number of steps per iteration. In three-phase algorithms, there are *request*, *grant*, and *accept* steps in every iteration. In the request phase, every input sends a request to *every* output it has at least one cell for. In the grant phase, every output independently selects one request to grant. As these decisions are independent, multiple outputs may grant the same input. Therefore, in the third phase, every input selects one grant to accept. Two-phase algorithms, on the other hand, comprise only a request and a grant phase. In the request phase, every input sends a request to *one* output for which it has at least one cell. In the grant phase, every output independently selects a request to grant. Because every input can receive one grant at maximum, there is no need for an accept phase, i.e., every grant is automatically accepted.

Input and output selection is based on a prioritized round-robin mechanism, i.e., the input (output) selector chooses the first eligible output (input) starting from the position indicated by a *pointer*. The pointer update policy is a crucial characteristic of each algorithm and must be chosen carefully to guarantee performance and fairness. The update policies employed by these algorithms share a common trait: once a connection (corresponding to a VOQ) becomes highest priority, it will be given precedence over the other competing ones until it is established. In *i*-SLIP this is achieved by having an output grant the same input (in the first iteration) until the grant is accepted. In DRRM, on the contrary, an input will keep requesting the same output (in the first iteration) until it receives a grant. This feature guarantees fairness and leads to *pointer desynchronization* [18], i.e., it assures that under heavy traffic (when all the VOQs are nonempty) each output grants a different input (*i*-SLIP) or each input requests a different output (DRRM). When this happens, there are no conflicts and a maximum-size matching is produced in every timeslot, leading to 100% throughput.

### 3.3 Round Trip Time Latencies

Recently, several researchers have addressed the problem of scheduling algorithms in input queued switches when considering round trip time latencies in the scheduling process. In [38] round trip latencies are introduced by the need of addressing multi-rack implementation in very large switches. Multi-rack implementation implies that the physical distance between line-cards and the switching fabric is non negligible with respect to the time slot. As such, performance of a centralized scheduler based on the classical iterative three-phase (request-grant-accept) scheme are shown to degrade for large physical distances; a solution to cope with this problem is proposed, being based on a differential signalling scheme and on a slight increase of the scheduler complexity, which is assumed to keep track of VOQs state.

The centralized single-chip implementation of scheduling algorithms is largely dominant; however, scalability problems may arise for very large high-speed switches [39]. When looking at multi-chip implementations, device separation implies that decision taken by input/output selectors belonging to different devices are known only after an inter-chip latency, named RTT (Round Trip Time). As such, as shown in Fig. 3.1, information critical i) to determine the matching, ii) to update the pointers and iii) to issue new requests, is delayed by the inter-chip latency, causing performance degradation. RTTs may be significant with respect to the time slot. Indeed, at high speed the time slot is rather short (13ns at 40Gbit/s for a 64bytes packet) if compared with inter-chip communication latencies which include propagation delays, data serialization and pin sharing which may be required to overcome the I/O pin count limit.

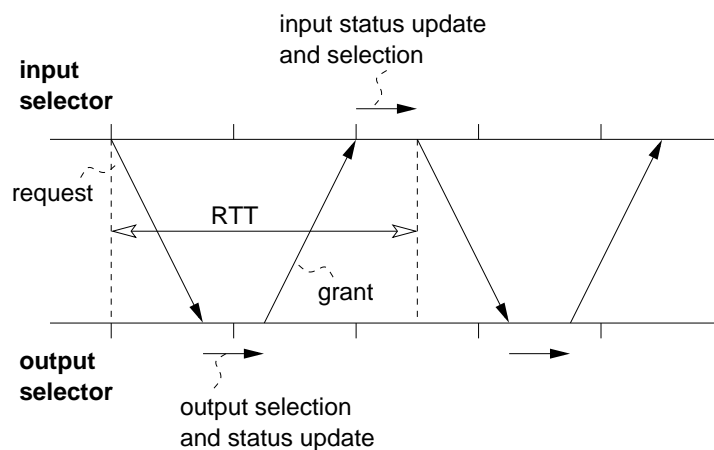
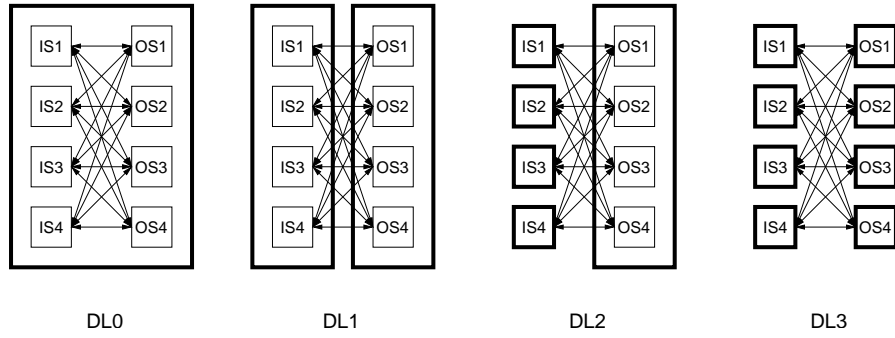


Figure 3.1. Round trip time between input and output selectors.

### 3.4 Multi-chip implementation

Let us focus on iterative schedulers, based on three-phase or two-phase information exchange among inputs and outputs. These schedulers adopt OSs (Output Selectors) to choose among multiple requests received by inputs and ISs (Input Selectors) to select a proper request to issue in a given time slot and to choose among multiple grants received by OSs. Scheduler distribution over several chips entails partitioning the selectors used to determine a heuristic matching over physically separated devices. In a single-chip implementation, all selectors are tightly coupled and decisions taken at inputs (outputs) are immediately available to outputs (inputs). When dealing with multi-chip implementations, the communication latency between devices implies that algorithms devised to run under the hypothesis of having all the scheduling state information available may not be optimal. Indeed, information needed to update the pointer status or to issue new requests may be known with a delay of some tens of cell time. Performance degradation and loss of fairness were already shown to be a possible problem in this multi-chip scenario.

Different levels of selectors distribution could be envisioned, which yield to a different number of physically separated devices, as shown in Fig. 3.2. The first obvious solution to reduce the scheduler complexity, labeled distribution level DL1, is to implement the scheduler in two separate devices, each containing respectively  $N$  input and  $N$  output selectors. This allows to roughly divide by two the scheduler hardware complexity. Another extreme, labeled distribution level DL3, is to distribute the selectors over  $2N$  separate devices, each device implementing one selector. This is referred to as fully distributed solution, which permits a hardware complexity reduction by a factor of  $N$ .



**Figure 3.2.** Distribution levels of a centralized multi-chip scheduler: DL0 is a monolithic, single-chip implementation, DL3 is a fully distributed multi-chip implementation.

As an intermediate step, in [39] a further possible solution is proposed: the  $N$  input selectors are physically separated in  $N$  devices, whereas all  $N$  output selectors are implemented in a single device. This solution has a major drawback: it reduces the hardware complexity by a factor of two only. However, having the output selectors in a single device permits coordination among output selectors with no delay. This permits to implement schedulers with more iterations in a single cell time, thus preserving good performance for increasing RTTs; however, this limits scheduler scalability. As such, we focus on schedulers suited to a fully distributed multi-chip scenario (distribution level DL3).

## Distributed Scheduler under Unicast Traffic Conditions

### 4.1 Scheduling algorithms with RTT latencies

Dealing with RTTs among devices has a profound impact on scheduler design. In [39], the proposed two-phase scheduler is a direct extension of the DRRM (Dual Round Robin Matching) scheduler[20], originally conceived for a monolithic implementation, to a distributed environment.

Let us briefly summarize DRRM behavior, focusing on an enhanced version of DRRM which achieves lower delays, thanks to a modified pointer update rule similar to that used in FIRM (Fcfs in Round robin Matching) [21]. In every iteration, first a request is sent by any unmatched input to the first unmatched backlogged output in the round-robin order starting from the current request pointer position. If an output receives more than one request, it grants the one that appears first in the round-robin order starting from the current position of the grant pointer. Request pointers are updated, in the first iteration to point to the output selected in the request phase, and further to one position (modulo  $N$ ) beyond the output selected if and only if the request is granted in the first iteration. Grant pointers are updated to one position (modulo  $N$ ) beyond the input granted in the first iteration.

In a monolithic implementation, all decisions are taken, and known, in a single time slot, and pointers are updated accordingly. In a distributed implementation, first, request information is delayed by  $RTT/2$  (assuming symmetric RTTs) and the request pointer update cannot be performed immediately, since grants will be available  $RTT/2$  slots later. A straightforward extension of DRRM to a distributed scenario would imply that requests to be issued in the next time slot are based on pointer positions not updated, thus breaking the round-robin de-synchronization mechanism and leading to throughput degradation. Moreover, request selectors would not be able to accurately know the number of underway grants, thus negatively affecting request decisions.

The distributed extension of DRRM proposed in [39] is based on the following ideas. Let us focus on a single iteration case. The key idea to keep the pointer de-synchronization is to ensure that every pointer is updated at most

once every RTT slots. As such, each input and output selector keeps a distinct (request and grant) pointer for every RTT slots. Traditional pointer update rules are used: Request pointers are only updated when the corresponding grant arrives, one RTT after issuing the corresponding request, whereas grant pointers are updated immediately after issuing a grant, since issued grants are accepted by definition. In other words, with respect to a monolithic implementation, RTT staggered schedulers are running in pipeline dealing with requests and grants.

This implies that the scheduler complexity (in terms of required pointers) increases linearly with RTT, since RTT pointer registers are required per selector. An additional counter (modulo RTT) is needed to indicate the current pointer to be used, whereas the combinatorial selection logic does not need to be duplicated, since a multiplexer to select the proper register among the RTT registers is enough to permit a correct behavior.

Another issue is related to pending requests. Indeed, only after RTT slots it is possible to know whether an issued request was granted. In the meantime, input selectors should issue further requests. If the number of submitted requests exceeds the number of enqueued cells, it may happen that a slot is reserved for a VOQ that will become empty by the time the grant is received at the input selector, thus wasting system resources. To solve this problem, a PRC (Pending Request Counter) per VOQ plus a request history per input selector are introduced [39]. Basically, new requests are issued only if the number of pending requests is smaller than the number of cells currently stored in the corresponding VOQ. This choice, which further complicates the selector design, is fundamental to obtain good performance at low loads or under heavily unbalanced traffic.

Further problems are related to the issue of dealing with more iterations in a single time slot (see [39] for details). We disregard issues related to iterations, which can be reasonably used only if all output selectors share the same physical device. Indeed, iterations are based on the knowledge of the results of previous iterations in the same time slot; in a fully distributed scheduler with RTTs, knowledge of results of previous iterations requires RTT time slots, thus basically preventing the possibility of iterating in this scenario.

## 4.2 Synchronous Round Robin

Let us now describe the SRR scheduler, initially disregarding issues related to RTT for simplicity. The SRR scheme is based on a cyclic, TDMA-like, preferential scheduling of VOQs. This preferential scheduling is obtained by logically numbering the slots with an incremental counter  $s$ , ranging from 0 to  $N - 1$ , i.e., a modulo  $N$  counter. Slots are logically organized in frames, named SRR frames; each frame comprises  $N$  slots.

Let us describe the input and output selectors behavior at time slot  $s$ :



1. the input scheduler associated to input  $i$  preferentially selects for a transmission the VOQ with destination output  $|i + s|_N$ . In other words, a preferential request is issued for output  $|i + s|_N$ .
2. if the preferential VOQ is empty, a request for the longest VOQ is attempted; ties among VOQs are broken according to a round-robin scheme.

Output schedulers grant the preferential request, if issued; otherwise, a randomly chosen request is granted among non-preferential conflicting requests.

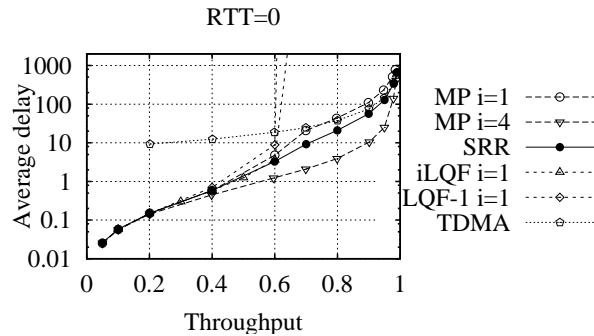
Other possible solutions exist to break ties among VOQs at input selectors, such as random or round-robin choice among non-preferential VOQs; however, the longest VOQ choice provides the best compromise between complexity and performance, as discussed later. Note that the random choice among conflicting non-preferential requests at output selectors may be not the most natural choice. Indeed, given that input selectors choose non preferential requests exploiting a longest queue first algorithm, output selectors could select the request corresponding to the longest VOQ among conflicting requests. However, this would require an increase in signalling complexity, since VOQ lengths should be sent to output selectors, and also a complexity increase at output selectors, since conflicting request should be compared according to their length. Selecting longest queues at inputs is easier, since queues could be kept simply ordered by length; indeed, in a given time slot, at each input at most one departure and one arrival can occur. Moreover, performance results show that the benefit of the longest queue selection at outputs is marginal for both balanced and unbalanced traffic.

Regardless of the fact that the request is granted or not, a new selection is made in the next time slot, according to the above described algorithm. In summary, for low switch loads SRR behaves similarly to a single-queue FIFO strategy. At high loads, when all queues are always non-empty, the SRR preferential scheduling deterministically orthogonalizes input request attempts, so that a single preferential request is received by any output in a given time slot. More precisely, for network loads larger than or equal to the channel capacity, input selectors behave exactly like in a Time Division Multiple Access (TDMA) scheme: during a SRR frame, whose length is equal to  $N$  slots, all inputs have exactly one access opportunity for transmissions toward each output, and they exploit this access opportunity deterministically, thereby avoiding potential conflicts at outputs. In this sense, SRR is throughput optimal under overloaded uniform traffic conditions.

A nice property of SRR is that it can be used without any modification i.e., with no complexity increase, to deal with RTTs. The only difference is that grants will be received one RTT later with respect to request issue, thus negatively affecting delays. However, we will show that even without exploiting any pending request counter, performance at low loads are comparable with those of iterative algorithms exploiting pending request counters.

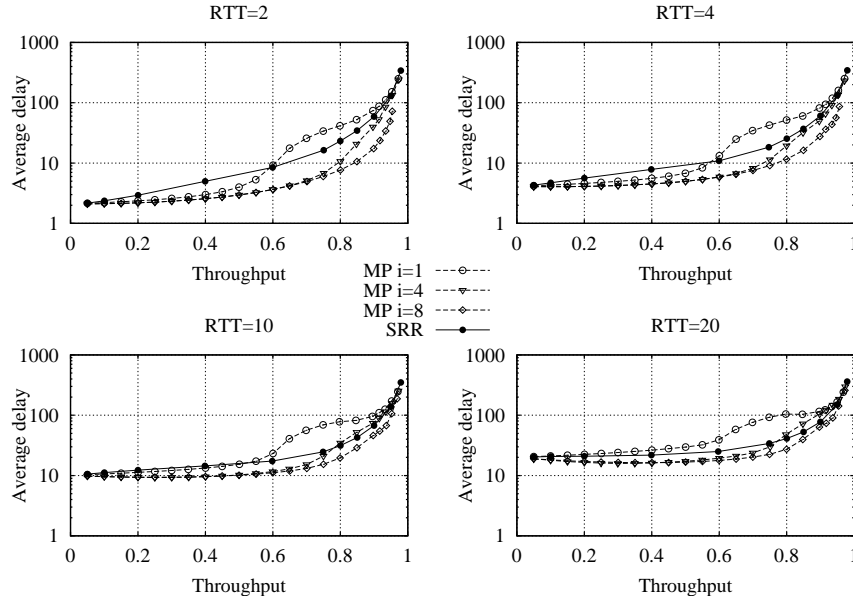
### 4.3 Performance results

We show performance results based on simulation runs exploiting a proprietary simulation environment developed in C language. Statistical significance of the results are assessed by running experiments with an accuracy of 1% under a confidence interval of 95%. We compare SRR with the distributed extension of DRRM presented in [39]. The switch has  $N = 16$  inputs and outputs running at the same speed, and is loaded with either Bernoulli or Bursty traffic with geometrically distributed burst sizes with an average burst size of 10 cells; cells in the burst are all directed to the same output. Performance indices are either average delays vs normalized switch throughput or maximum achievable normalized throughput in overload. SRR performance are reported as solid lines with black dots, the modified version of DRRM, which accounts for multiple pointers to deal with RTTs, is labeled MP (Multi Pointer) and plotted using dashed lines; different symbols refer to a variable number of iterations.



**Figure 4.1.** Performance comparison between SRR, MP, iLQF and TDMA under uniform Bernoulli traffic in a monolithic scheduler implementation (RTT=0).

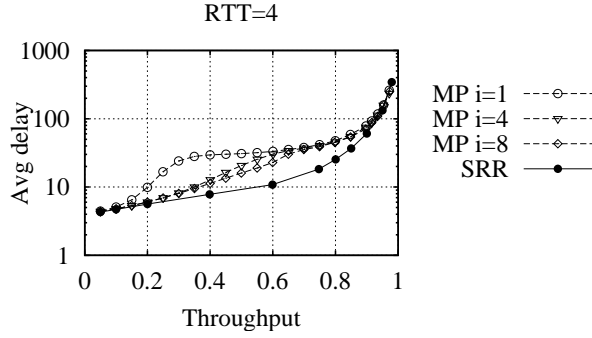
Let us first examine, in Fig.4.1, a scenario in which  $RTT=0$ , which corresponds to the traditional single-chip implementation of schedulers, under uniform Bernoulli traffic. Besides DRRM and SRR, we plot also a TDMA scheme, iLQF (Longest Queue First) [10] with one iteration, and LQF-1, a simplified version of iLQF with one iteration where inputs send a single request only (for the longest queue) per time-slot to outputs. Note that the TDMA scheme corresponds to SRR using the preferential scheduling only whereas LQF-1 corresponds to SRR using the non-preferential scheduling only. Clearly, the preferential scheduling scheme of SRR is fundamental to obtain good performance results at high loads: indeed, iLQF with  $i = 1$  iteration saturates at 0.65, whereas LQF-1 saturates at 0.61. At low loads the non-preferential



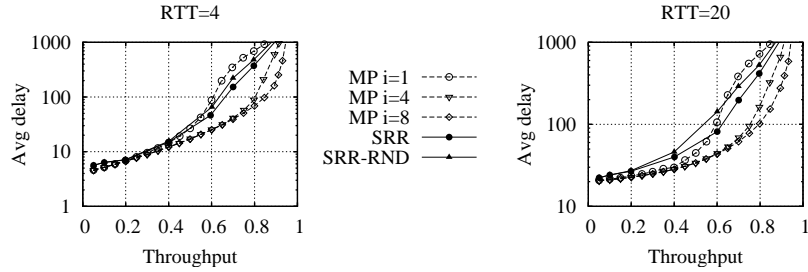
**Figure 4.2.** Performance comparison between SRR and MP under uniform Bernoulli traffic for variable RTTs.

scheme of SRR becomes dominant, so that performance are close to those of modified DRRM and much better than those of a pure TDMA.

In Fig.5.1 we report delays as a function of the switch throughput for variable RTTs under uniform Bernoulli traffic. No throughput limitations are observed for both SRR and modified DRRM; SRR shows remarkably low delays, improving performance as RTT increases. Only a slight delay impairment can be noticed for low-medium loads and small RTTs; this is due to the missing pending request counters, which are instead used in the modified DRRM. Recall that pending request counters are fundamental to avoid issuing too many request for the same VOQ; indeed, if the number of requests issued exceeds the number of enqueued cells, which may happen at low-medium loads and for large RTTs, it may happen that the VOQ becomes empty by the time the grant is received, thus wasting system resources. Indeed, modified DRRM without pending request counters performs much worse, as shown in Fig.4.3, where SRR clearly outperforms the modified DRRM. Moreover, recall that SRR, besides not exploiting any additional counter, does not require any iteration, thus allowing a fully distributed implementation; nevertheless, delay performance are comparable with those of modified DRRM with  $\log N = 4$  iterations, and improve with increasing RTTs. Minor differences exist in this scenario if the non-preferential VOQ choice at inputs in SRR is done on a



**Figure 4.3.** Performance comparison between SRR and MP under uniform Bernoulli traffic when MP is not using the PRC counters.



**Figure 4.4.** Performance comparison between SRR and MP under uniform Bursty traffic for variable RTTs.

round-robin basis instead of using a longest queue choice.

Similar performance are shown by SRR for Bursty traffic, as reported in Fig.4.4. Delay properties are remarkable, since SRR shows much better performance than modified DRRM with  $i = 1$  iteration also under correlated traffic. Note that SRR with the random choice of non-preferential VOQs at input selectors, labelled SRR-RND in the plot, show much worse performance than SRR for medium loads, thus justifying the choice of the longest VOQ when the preferential queue is empty.

## Distributed Scheduler under Multicast Traffic Conditions

### 5.1 Multicast scheduling algorithms

We define the fanout set of a multicast cell as the set of outputs to which the cell should be transferred. The cell fanout is the number of outputs in the fanout set. A multicast scheduler may not be able to transfer a multicast cell in a given time slot to all outputs in the cell fanout set, since some outputs may be matched to other inputs by the scheduler. In this case, to enhance performance, most multicast schedulers try to send a copy of the multicast cell to the largest available set of outputs; this is often named fanout splitting discipline. Fanout splitting disciplines may leave a residue, i.e., a copy of the multicast cell that must reach the subset of output ports that were not matched to the given input port in previous time slots.

We focus on two previously proposed multicast scheduling algorithms, selected due to their ease of implementation and good performance: WBA (Weight-Based Algorithm) [36] and mRRM [44]. Both rely on a single FIFO queue at each input for multicast. We briefly remind the WBA and mRRM scheduler behavior referring to a scenario where  $RTT=0$  for simplicity. Both schedulers are based on a two-phase request-grant algorithm.

WBA assigns weights to input cells based on their age and fanout at the beginning of every cell time. Once weights are assigned, each OS chooses the heaviest input among those subscribing to it. More precisely, at the beginning of every cell time, each IS computes the weight of the new multicast cell/residue at its HOL based on the age of the cell (the older, the heavier), and the fanout of the cell (the larger, the lighter). Then, each IS sends this weight to all outputs that the cell/residue at its HOL wishes to reach. Each OS grants to the input with the highest weight, independently of other outputs, breaking ties with a random choice. Note that a positive weight should be given to age to avoid input starvation. However, to maximize throughput, fan-out are weighted negatively. Several weight assignments algorithm can be adopted (see [36] for details). We use the weight definition chosen in the simulator available on the Web site at <http://klamath.stanford.edu/tools/SIM/>:

The cell weight is equal to the number of inputs minus the cell fanout plus the cell age.

Multicast Round Robin (mRRM) was designed to be simple to implement in hardware. A single pointer to inputs is collectively maintained by all outputs. Each output selects the next input that requests it at, or after, the pointer, following a round robin order. At the end of the cell time, the single pointer is moved to one position beyond the first input that is served. However, the single pointer update rule at OSs of mRRM does not permit a fully distributed multi-chip implementation, since output selectors need to be aware of other output selector choice to update the pointer value and this would imply a delay equal to the RTT for pointer update rule that would break the mechanism.

## 5.2 Improved Multicast Round Robin

Direct extensions of WBA and mRRM can be envisioned to deal with RTTs among input/output selectors induced by multi-chip implementation. Since grants are received at IS with a delay of RTT slots, a multicast cell is at the head-of-the-line for at least RTT slots. To avoid issuing multiple requests for the same cell, thus wasting resources, a number of queues equal to  $RTT+1$  is needed. ISs choose the queue from which the request is issued according to a round-robin fixed scheme among input queues. As such, at most one request per input queue is issued in each RTT. OSs keep working as in the basic WBA and mRRM scheme.

Further extensions can be envisioned and are studied later when presenting simulation results. Indeed, the single FIFO scheme when  $RTT=0$  and its direct extension to  $RTT+1$  FIFO queues when  $RTT > 0$  does not solve the issue of HoL blocking for multicast flows. As such, in some experiments, we introduce a limited number of  $k = 2$  FIFO queues at each input and  $k = 2 \times (RTT + 1)$  for  $RTT > 0$ . In this case, whereas output selectors operate as in the basic scheme, input selectors choose one among the  $k$  queues according to queue weights for WBA and to a round-robin scheme for mRRM.

The introduction of more queues at inputs, either to deal with RTTs or to reduce the HoL blocking, introduces the issue of multicast flow assignment to queues. Although several possible assignment strategies were studied in the past [29, 30], in this paper we simply adopt a packet-by-packet load balancing scheme among available queues. We are aware that this may introduce out-of-sequence delivery, but we prefer to avoid studying assignment scheme in this initial work and concentrate on the issue related to dealing with RTTs.

Let us now describe the IMRR (Improved Multicast Round Robin) scheduler, initially disregarding issues related to RTT for simplicity when a single FIFO queue is available. The same concept of preferential input is kept as in the mRRM algorithm: at each time slot, all output selectors keep a single pointer to a preferential input. However, the pointer update rule is stateless;

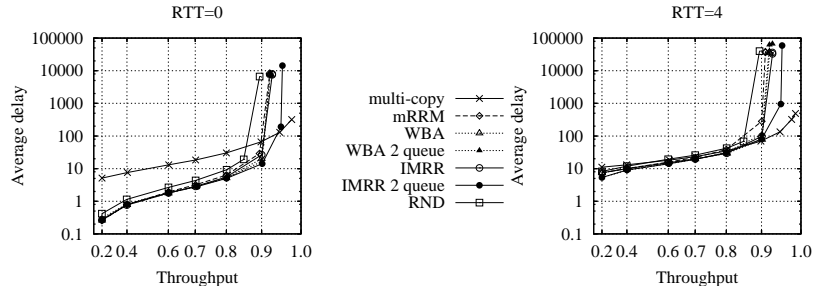


Figure 5.1. Performance comparison under uniform Bernoulli cell arrivals.

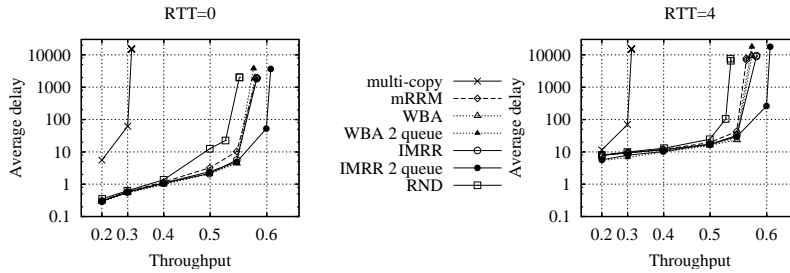
regardless of the granted cell, at each time slot, the pointer is increased (modulo  $N$ ) according to a round robin scheme. This is fundamental to guarantee that the scheduler can run in the fully distributed case, since no coordination among output scheduler is required. Input selectors issue a request containing a weight equal to the fanout of the selected cell. Output selectors choose the request from the preferential input, if any, otherwise they grant the request corresponding to the smallest fanout.

When more than one queue is present at inputs to avoid HoL (i.e.,  $k = 2$  queues for  $\text{RTT}=0$  and  $k = 2 \times (\text{RTT}+1)$  queues for  $\text{RTT} > 0$ ), input selectors issue a request for the cell with largest weight, the weight being queue length plus cell fanout.

### 5.3 Performance results

We show performance results based on simulation runs exploiting a proprietary simulation environment developed in C language. Statistical significance of the results are assessed by running experiments with an accuracy of 1% under a confidence interval of 95%. We compare IMRR with the previously presented extensions of mRRM and WBA, with a random (RND) scheduler, and with the multi-copy approach, where multicast cells are replicated at inputs and treated as unicast cells. In the multi-copy approach, the unicast SRR scheduler [31] is run, given its good performance and adaptability to the fully distributed multi-chip scenario.

The switch has  $N = 16$  inputs and outputs running at the same speed; cells are generated according to an i.i.d. Bernoulli process, i.e., at each time slot, an input port receives a cell with probability  $\rho$ ,  $0 \leq \rho \leq 1$ , equal to the input load. Later, we also consider packet arrivals, i.e., trains of cells willing to reach the same output, the number of cells being drawn from a uniform distribution ranging from 1 to 16 cells. Performance indices are either average delays vs normalized switch throughput or maximum achievable normalized throughput in overload.



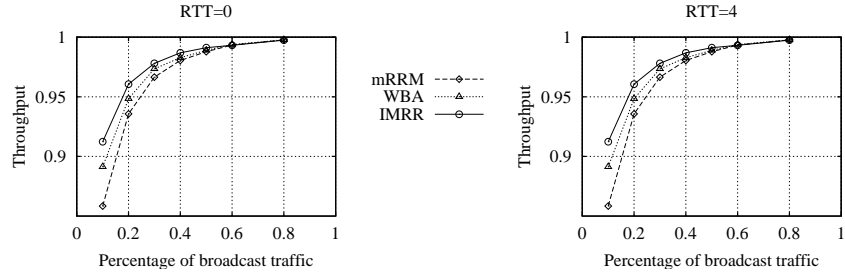
**Figure 5.2.** Performance comparison under gathered traffic with cell arrivals.

In terms of traffic distribution among input and output ports, we initially consider both i) a uniform scenario, in which all input (and output) ports are equally loaded and the fanout set of a new cell is generated randomly, according to a uniform distribution, ii) a gathered scenario, where the traffic is gathered over few ( $M = 5$ ) active input ports and equally distributed over all  $N = 16$  output port. In the gathered scenario, the fanout set is chosen according to a non-uniform binomial distribution, with mean fanout  $h_m = 3.66$ . More precisely, the probability  $P_f$  of choosing a fanout set of size  $f$  is  $P_f = N/h_m \binom{N}{f} (h_m/N)^f (1 - h_m/N)^{N-f}$ . This is a traffic pattern well known to be hard to schedule [29]. Indeed, when all inputs are equally loaded, the maximum sustainable traffic leads to a normalized input load which is at most  $1/E[f]$ ,  $E[f]$  being the average cell fanout size. If instead the traffic is gathered among few inputs, the normalized input load for sustainable traffic can approach 1, so that the efficiency in serving cells queued at the inputs becomes important on performance. Note that the considered gathered traffic scenario is far from being unrealistic. Multicast applications often generate sustained and long-lasting flows, that may only engage few inputs and several outputs at a given router or switch.

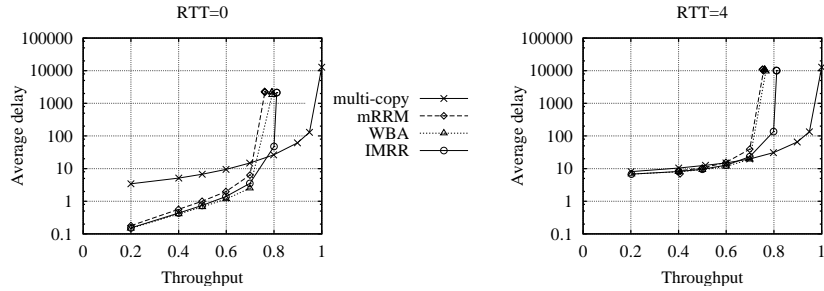
In both uniform and gathered traffic scenarios, unicast traffic is not given any special attention and is treated as a special case of multicast traffic with fanout set equal to one. To analyze switch behavior in a more realistic environment which includes a significant percentage of unicast traffic, we also examine two mixed traffic scenarios, the first one comprising a variable mix of unicast and broadcast traffic only, the second one in 50% of unicast and 50% of multicast traffic load the switch.

IMRR performance are reported as solid lines with circles, the modified version of WBA is plotted using dotted lines with triangles, whereas the modified version of mRRM is identified by dashed lines with diamonds; the multi-copy approach is plotted with a solid line with crosses, and the random scheduler with squares. White symbols refer to the case of a single FIFO for  $\text{RTT}=0$  and  $\text{RTT}+1$  queues for  $\text{RTT}>0$ , whereas black symbols are used for  $k=2$  FIFOs for  $\text{RTT}=0$  and  $k=2 \times (\text{RTT}+1)$  FIFOs for  $\text{RTT}>0$ .





**Figure 5.3.** Performance comparison for uniform traffic and a variable percentage of unicast and broadcast traffic only.



**Figure 5.4.** Performance comparison for uniform traffic: 50% unicast and 50% multicast traffic.

In Fig.5.1 we report delays as a function of the switch throughput for variable RTTs under uniform multicast Bernoulli traffic with cell arrivals. Only the multi-copy approach permits to obtain 100% throughput; however, the price to be paid is a significant increase in the average delay at low-medium loads when  $RTT=0$ . When increasing RTT to 4 time slots, all algorithms show similar delay performance. No major differences are evident among multicast schedulers, apart from a slight throughput increased obtained by the proposed IMRR when using  $k$  FIFO queues per RTT. The RND scheduler performs worse, as expected.

Despite its good performance in the uniform scenario, recall that the multi-copy approach is reported only as a reference case; indeed, it cannot be used in practice, since, for example, it is unable to sustain a broadcast flow overloading a single input. Indeed, when studying the performance of the switch under gathered traffic in Fig.5.2, the throughput limitation of the multi-copy approach becomes evident. In this scenario, the proposed IMRR approach provides improved throughput especially when using  $k = 2$  FIFO queues per RTT. The WBA scheduler, despite its higher complexity, is unable to exploit  $k = 2$  FIFO queues to obtain performance benefits. The RND scheduler still

presents the worse performance among the multicast schedulers (excluding the multi-copy approach).

IMRR shows good throughput performance also when considering the traffic scenario with unicast and broadcast traffic only, reported in Fig.5.3. When broadcast traffic becomes dominant, all algorithms show similar, good, performance. However, when unicast traffic becomes more significant, differences among algorithms become evident even when using a single FIFO queue per RTT, and IMRR outperforms the other algorithms.

In Fig.5.4 we report delays as a function of the switch throughput when considering the scenario in which 50% of the traffic is uniform multicast and 50% is uniform unicast. Also in this case IMRR provides throughput benefits with respect to WBA and mRRM both when  $RTT=0$  and  $RTT > 0$ .

Finally, we study switch performance when considering packet arrivals at inputs instead of cell arrivals. Packet size is uniformly chosen among 1 and 16 cells. When a packet is generated, it is segmented in cells and cells are sequentially stored in the proper queue for successive transmission. Cells access the switching fabric and are transmitted independently.

In Fig. 5.5 we report delays under uniform Bernoulli traffic. The surprising result is that WBA provides worse performance than IMRR and even of RND, both when  $RTT=0$  and  $RTT=4$ . Recall that WBA requires a significant complexity increase with respect to the two other schedulers. This behavior is highlighted not only by the uniform packet size distribution, but also by other variable packet size distributions such as a trimodal distribution similar to the distribution of Internet packet size. The reason for the throughput decrease can be explained by looking at the matching size PDF (Probability Density Function) reported in Fig. 5.6 for  $RTT=0$  (similar results not shown hold for  $RTT=4$ ) in overload. The matching size is computed as the number of edges that the scheduler selects in each time slot. The uniform packet size distribution penalizes the ability of the multicast schedulers in selecting large size matchings, as visible when comparing the left plot for fixed packet size with the right plot. This is due to the following phenomenon. When two badly conflicting packets reach the HoL of their respective queue, they start competing for the same set of resources (output ports). Non conflicting packets in other queues can be transferred, but, since this conflicting situation lasts for several time slots due to the packet size, there is an increasing chance that other conflicting packets reach the HoL of other queues, creating a self-sustaining conflicting situation. This behavior is shared by all schedulers, and the maximum throughput is reduced from 0.9 to 0.8.

WBA further exacerbates this problem, since the weighted metric increases the probability of making the same matching choice at output ports in consecutive time slots. This is not true for both IMRR and RDN that, thanks to the round-robin or random selection, make independent choice in each time slot. To confirm this, in Fig. 5.7 the matching persistency PDF is reported when  $RTT=0$  (similar results, not shown, hold for  $RTT=4$ ). The matching persistency is computed according to the following algorithm: considering two

consecutive time-slots, compute the persistency as the number of edges that are selected in both matching. Clearly, the matching persistency ranges from 0 to 16. WBA shows an increase in matching persistency due to the memory effect created by the weighted metric when selecting the matching edges. When using a fixed packet size distribution, with a packet size equal to the cell size, this memory effect disappears. As such, weighted metrics, often proposed when running multicast scheduler, should be reconsidered, since they seem to offer good performance but only when considering cell-based arrivals, a scenario not really significant in today networks.

In summary, IMRR presents equivalent or better performance than other previously proposed multicast schedulers. IMRR does not require the computation of any delay based metrics, a rather complex task in today high-speed switches. Furthermore, it is suited to a fully distributed implementation. Besides this additional constraint, IMRR shows good performance both in the traditional case of monolithic implementation as well as when considering RTT induced by the multi-chip implementation of multicast schedulers.

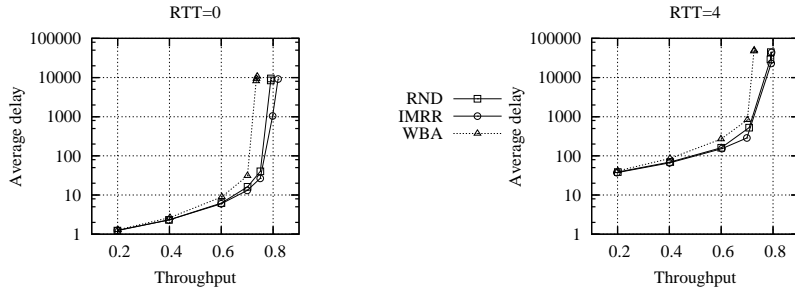


Figure 5.5. Performance comparison under uniform Bernoulli packet arrivals.

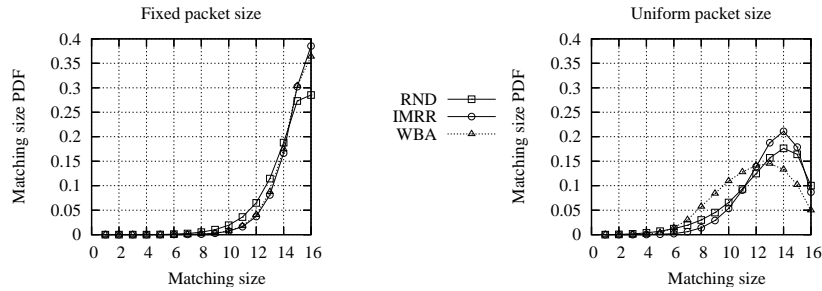
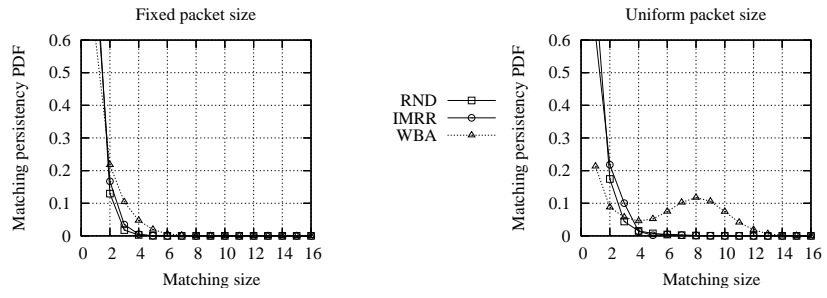


Figure 5.6. Matching PDF for RTT=0 under uniform Bernoulli packet arrivals.



**Figure 5.7.** Matching persistency for  $RTT=0$  under uniform Bernoulli packet arrivals.

## Distributed Integration of Unicast and Multicast Scheduling

### 6.1 Introduction

The problem of integration between the unicast and the multicast scheduling has been studied, from a theoretical point of view in [40]. These authors also proposed an integration scheme that consists of scheduling multicast traffic first and using the remaining resource for unicast. The main disadvantage of this scheme, which we call sequential, is that it easily leads to starvation of unicast traffic: a single input loading the switch with broadcast traffic would suffice to prevent unicast from getting any service at all.

In [41] the authors have proposed an improvement of the sequential scheme, in which unicast and multicast traffic are alternatively scheduled first in different time slot to avoid starvation. The choice of which scheduler runs first in a given interval must be based on traffic characteristics, in particular it must be based on the ratio of unicast to multicast traffic. If the ratio is not estimated properly or its value changes over time, system throughput can be severely degraded.

In [42] the authors showed that an approach to achieve integrated scheduling is to treat multicast traffic as unicast, but distributing the burden of cell replication over multiple ports. The main disadvantage with this scheme is that it potentially introduces high latency.

In [43] it is introduced an integration scheme called FILM (FILter&Merge) in which the integration scheme is based on two blocks: the request filter and the integration block. In this scheme the two schedulers run in parallel and independently. The integration block decides which unicast and multicast edges will be part of the integrated matching. The set of edges that are excluded from the integrated matching is called remainder. The request filter is a block capable of reserving a subset of the switch inputs and outputs by dropping the corresponding unicast and multicast requests. Reservations are made at any time slot based on information received from the integration block and the status of the queue. This scheme has the advantage that it doesn't have

to know the characteristics of the traffic and it avoids the starvation of a type of traffic.

## 6.2 Frame Integration

We define frame as a succession of slot of predetermined length. The use of the frame allows us to obtain a simple integration scheme. The integration scheme is based on the idea of labelling each slot either as a unicast or as a multicast slot. In unicast slots, priority is given to unicast traffic; in multicast slots multicast traffic has higher priority.

## 6.3 Description of Scheduler

In a "multicast slot", a multicast scheduling algorithm defines a multicast matching and a unicast scheduling algorithm defines a unicast matching. All the edges found by the multicast scheduler are to be part of integrated matching adding with all the edges found by the unicast scheduler not in conflict with the multicast edges. As the multicast scheduler we choose uses IMRR [49] and as the unicast scheduler SRR [31].

In a "unicast slot", for the unicast traffic, is used a modified version of the SRR algorithm. The modification aim to improve the performance of the multicast traffic which uses an LQF algorithm. The SRR scheme is based on a cyclic, TDMA-like, preferential scheduling of VOQs. In the request phase, the unicast scheduler sends the request for all the preferential VOQ. If a preferential VOQ is empty, a request for the longest VOQ is attempt; ties among VOQs are broken randomly. The multicast scheduler sends the requests in agreement whit the fanout set of the cell in head of the queues, i.e, it sends a request for any output in its fanout-set. Output schedulers grant the preferential request, if issued. Otherwise the request sent by the heaviest queue is chosen, in agreement to a LQF scheme, also taking into account the multicast request, in contrast to the original SRR scheme.

To improve performance, i.e., decreasing the conflict at the output selector, the common characteristic of IMRR and SRR to define preferential requests is exploited. Within a unicast slot, if an input sends a SRR preferential request, it doesn't send a request for multicast traffic. In the same way, in a "multicast slot", if an input sends a preferential request, according to the IMRR scheme, it doesn't send a request for unicast traffic.

## 6.4 Dynamic Frame

We call a frame as static when it is predetermined and it doesn't change for the duration of the simulation; therefore in this case, the frame scheme always

uses the same frame. Otherwise we call a frame dynamic when it changes in relation to the incoming traffic. In this case, the frame scheme will use a frame that will be redefined every time-window. A time-window corresponds, in our case, to 256 slot. The system calculates, at the end of that time, the total of unicast cells arrive to the inputs and the total of multicast cells multiplied by the respective fanout.

The frame is defined by assigning the slots to a unicast (multicast) traffic in proportion to the number of unicast (multicast) cells arrived in that time-window. We use a dynamic frame in order to obtain a scheme which doesn't depend on estimates of the traffic characteristics and which doesn't lose performance because of changes in the traffic. Thanks to the use of the dynamic frame, the frame scheme fits the incoming traffic and it obtains good performance as shown in the following paragraph.

## 6.5 Simulation Result

We have studied the performance of the system by simulation and then compared to the results obtained by the same simulation for the FILM scheme.

The simulated system is an 8x8 switch with inputs buffers that can store 1000 cells. Cells are generated according to an i.i.d Bernoulli process, i.e, in a time slot every input port receives a cell with probability  $\rho$ , equal to the input load. Each cell has a probability  $P$  of being a multicast cell. The fanout of multicast cells is uniformly distributed between 2 and 8. The traffic is uniform, i.e., all outputs have the same probability of being the destination of a unicast cell or of belonging to the fanout of a multicast cell. The total load on the switch is  $\rho(P * F + (1 - P))$ , where  $F$  is the average fanout, in our case  $F = 5$ , whereas  $P$  e  $\rho$  are varied to obtain the desired multicast load on the switch.

The tests for the frame scheme were made both for static frame, i.e, fixed frame with the 50% of unicast slots and 50% of multicast slots, and also for dynamic frame. We observed the total throughput as well as the individual throughputs of unicast and multicast traffic as the fraction of multicast traffic (MCF) grows from 0 (unicast only) to 1 (multicast only). Ideally, the throughput achieved by each traffic type should be equal to the corresponding share of the output load, and the total should be 100%.

The graphs 6.1 and 6.2 show the throughput achieved by FILM compared with the throughput achieved by the static frame scheme and the dynamic frame scheme. Both frame schemes always get higher performance than FILM, above all when the multicast traffic is predominant, i.e., when MCF is greater than 0.5. In the case of  $MCF = 0.7$  the difference between the dynamic frame scheme and FILM is more than 7%. It is interesting how on the FILM scheme, when the unicast is the predominant traffic type, i.e.,  $0 < MCF < 0.5$ , the unicast throughput exhibits a degradation whereas in the dynamic frame scheme the unicast data flow is always transferred completely. For example,

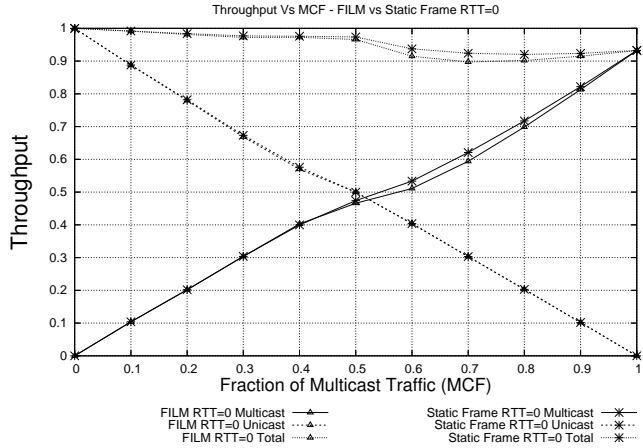


Figure 6.1. Static Frame Vs. FILM RTT=0

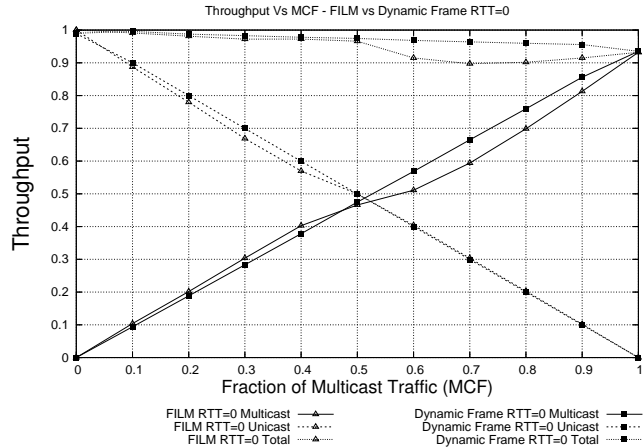


Figure 6.2. Dynamic Frame Vs. FILM RTT=0

when MCF=0.3, the unicast throughput reaches only the 94%, as we can see on the graphs 6.1 and 6.2

The graphs 6.3 and 6.4 show the throughput achieved by FILM compared with the throughput achieved by the static frame scheme and the dynamic frame scheme in case of RTT=8. Both frame schemes don't suffer losses in terms of throughput anyway, when the multicast traffic is predominant, the FILM scheme has significant losses in terms of throughput caused mainly by traffic multicast. The worst case is MCF=0.8, when the FILM scheme loses about 8% of the throughput and the difference with the dynamic frame scheme



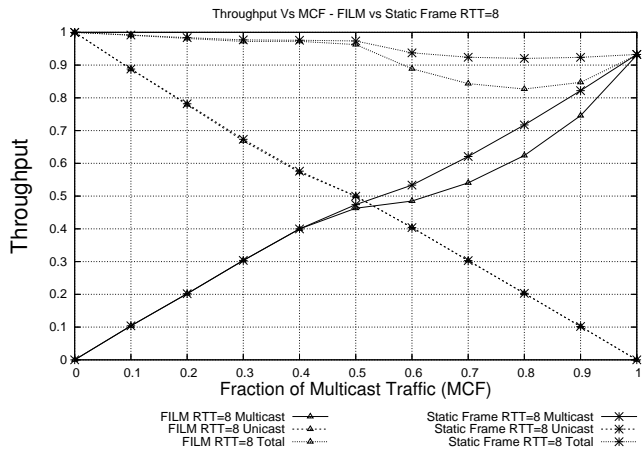


Figure 6.3. Static Frame Vs. FILM RTT=8

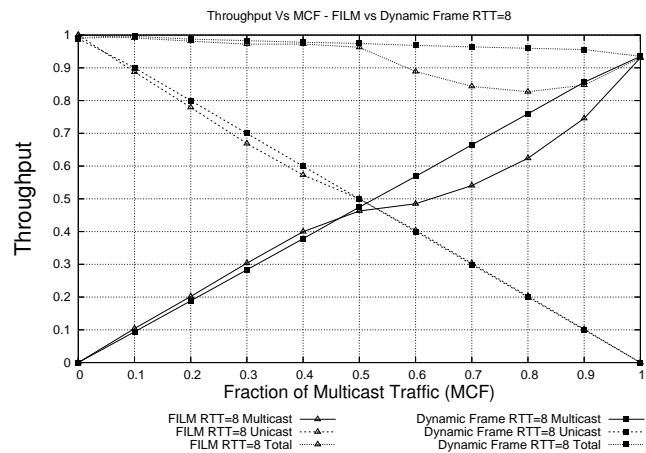


Figure 6.4. Dynamic Frame Vs. FILM RTT=8

becomes 15%.

These two graphs 6.5 6.6 show the performance in terms of delay of the static frame scheme when MCF=0.5 by varying the ratio of unicast slot to multicast slot within the frame. The best performance are obtained by frame with 50% unicast slots and 50% multicast slots, in analogy with the value of MCF. To change the ratio inside the frame, the performance worsens if we increase the unicast slots. The worst case is when the frame is composed only from unicast slots. Whereas the performance doesn't vary considerably if we increase the ratio of multicast slots.

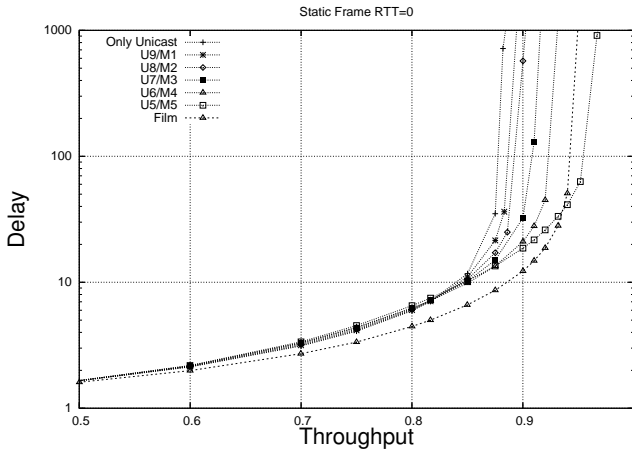


Figure 6.5. Static Frame RTT=0

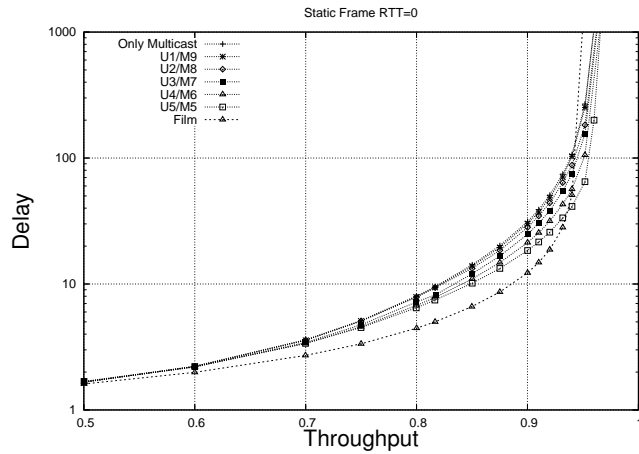


Figure 6.6. Dynamic Frame RTT=0

The graphs 6.7 and 6.8 show the average delay of the cells in function of offered load when  $MCF=0.5$ , i.e., when each traffic type is responsible for half of the output load. For value of  $RTT=0$  the FILM scheme obtains better performance compared to both of the frame schemes, in terms of delay. In the case of  $RTT=8$  both of frame schemes obtain better performance than FILM scheme for load values higher to 0.7.

The graphs 6.9 and 6.10 show the average delay of the cells in function of offered load when  $MCF=0.8$ . For  $RTT=0$  and for loads less than 0.8, the performance are similar for all three of the scheme analysed, but for high loads,

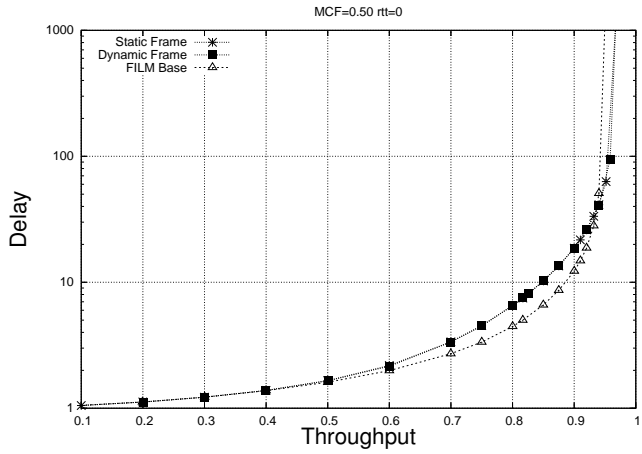


Figure 6.7. Frame Vs. FILM MCF=0.50 RTT=0

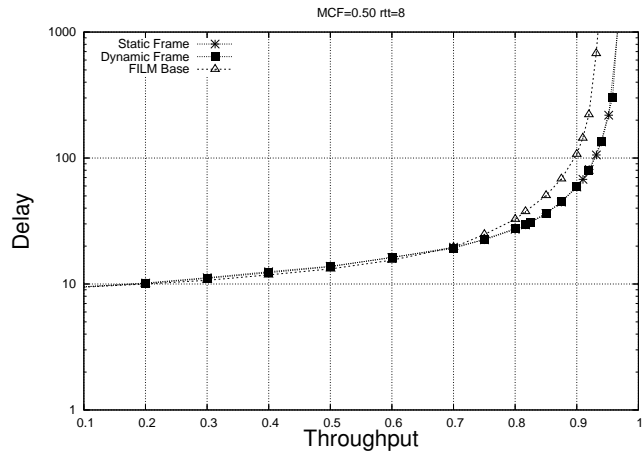


Figure 6.8. Frame Vs. FILM MCF=0.50 RTT=8

the frame scheme, in particular dynamic, gets the best results. A similar trend is obtained in the case of RTT=8 where the dynamic frame scheme obtains the best performance for all of the load values.

The graphs 6.11 and 6.12 show the trend of average delay of the cells both for static and dynamic frame schemes for increasing value of MCF. The biggest differences between the curves are for loads between 0.5 and 0.93. In this range the best performance are obtained in the case of MCF=0.9 for both frame schemes, anyway we have the worst performance for MCF=0.1. Comparing the two frame schemes, we can see that for values of MCF minors of 0.5, i.e.,

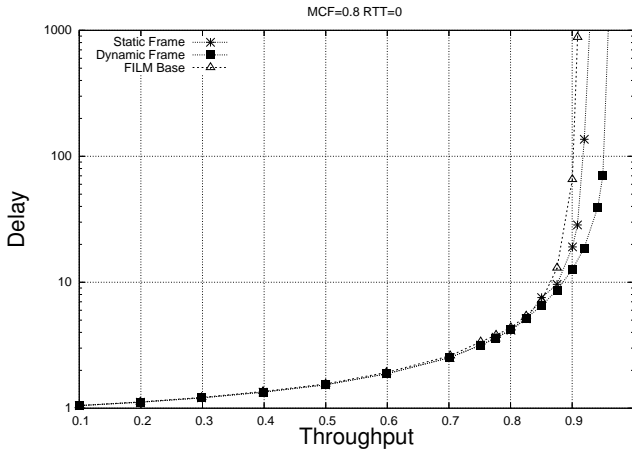


Figure 6.9. Frame Vs. FILM MCF=0.80 RTT=0

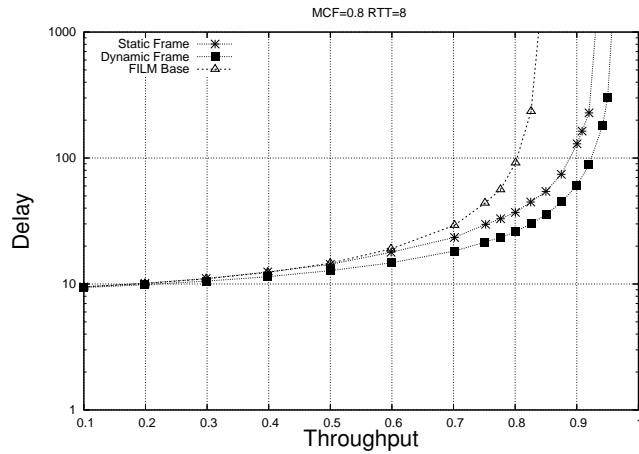


Figure 6.10. Frame Vs. FILM MCF=0.80 RTT=8

when the unicast traffic is predominant, the dynamic scheme obtain the best performance, whereas for values of MCF greater than 0.5 the performance between the two schemes are similar.

In the tests show on the graphs 6.13,6.14 and 6.15 we have used an 8x8 switch where an input receives only broadcast cells and the remaining 7 inputs receive cells unicast. Cells are generated according to an i.i.d Bernoulli process, i.e, every input port receives a cell with probability  $\rho$ . The graphs show the results for MCF equal to 0.2, 0.5 and 0.9. For MCF=0.2 and MCF=0.9 the frame schemes obtain the best results. In this scenario the dynamic scheme

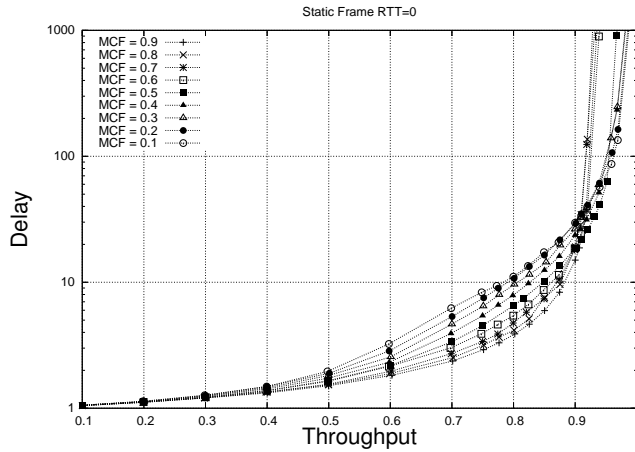


Figure 6.11. Static Frame RTT=0

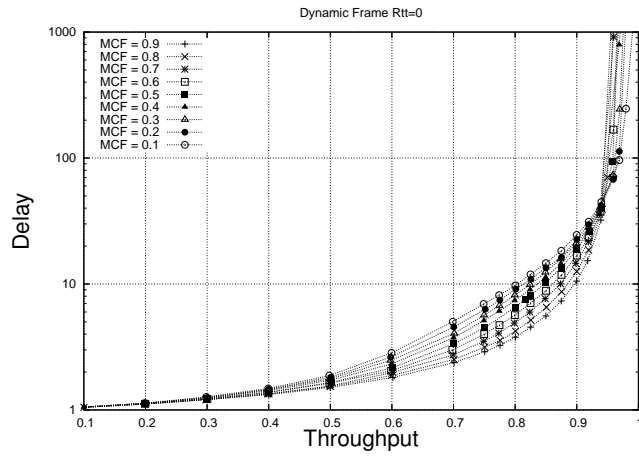


Figure 6.12. Dynamic Frame Rtt=0

obtains better performance than the static scheme.

In the tests show on the graphs 6.16 and 6.17 we have changed the type of traffic. The simulated system is a 16x16 switch in which the cells are generated according to an i.i.d Bernoulli process. The unicast traffic is uniform and fanout-set of traffic multicast is determined by "binomial fanout" described in [29]. From the graphs 6.16 and 6.17 the frame scheme obtains better performance than the FILM scheme both for RTT=0 and for RTT=8. This is due to a better management of multicast data flows in the frame scheme.

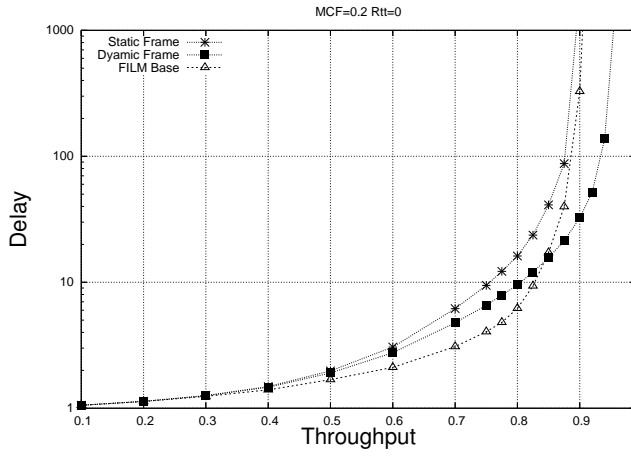


Figure 6.13. Frame Vs. FILM MCF=0.2 RTT=0

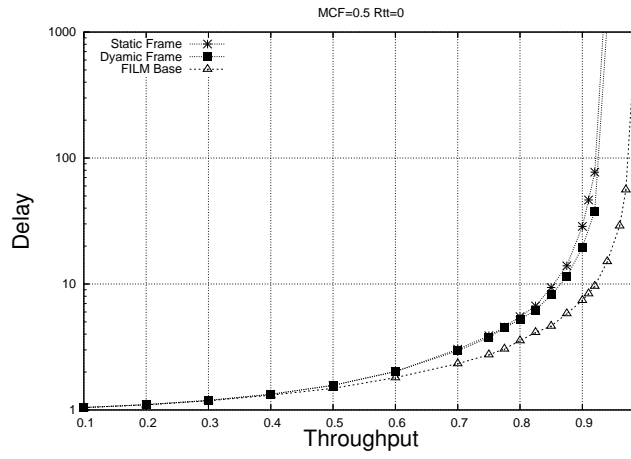


Figure 6.14. Frame Vs. FILM MCF=0.5 RTT=8

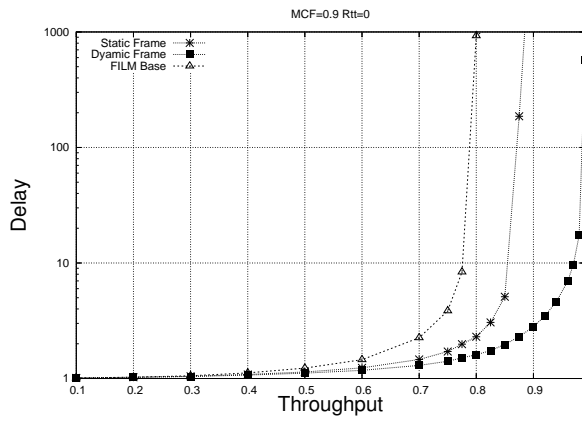


Figure 6.15. Frame Vs. FILM MCF=0.90 RTT=0

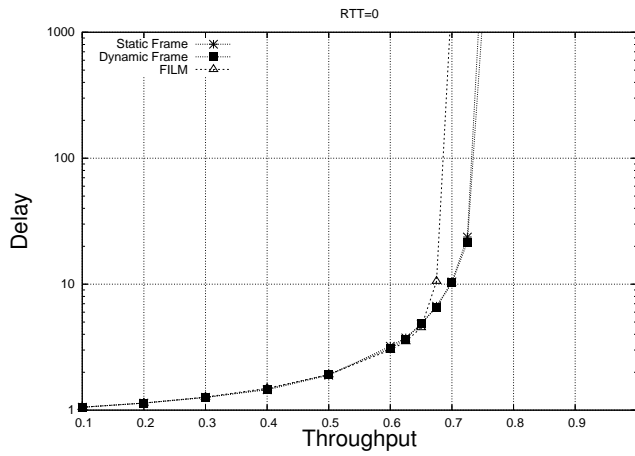


Figure 6.16. Frame Vs. FILM MCF=0.8 RTT=0

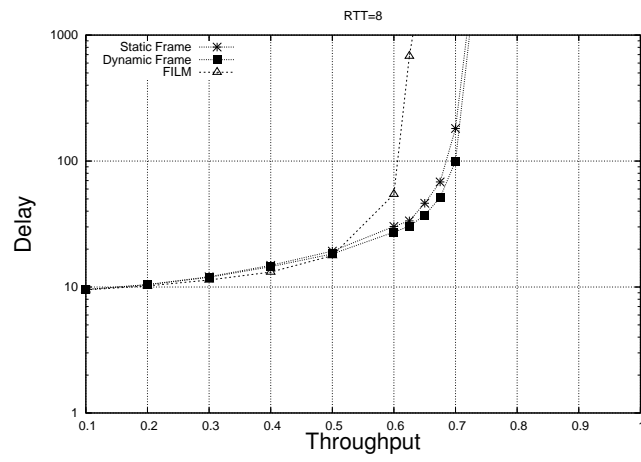


Figure 6.17. Frame Vs. FILM MCF=0.8 RTT=8



A Switching Architecture for Asynchronous  
SANs



## Introduction to Storage Area Networks

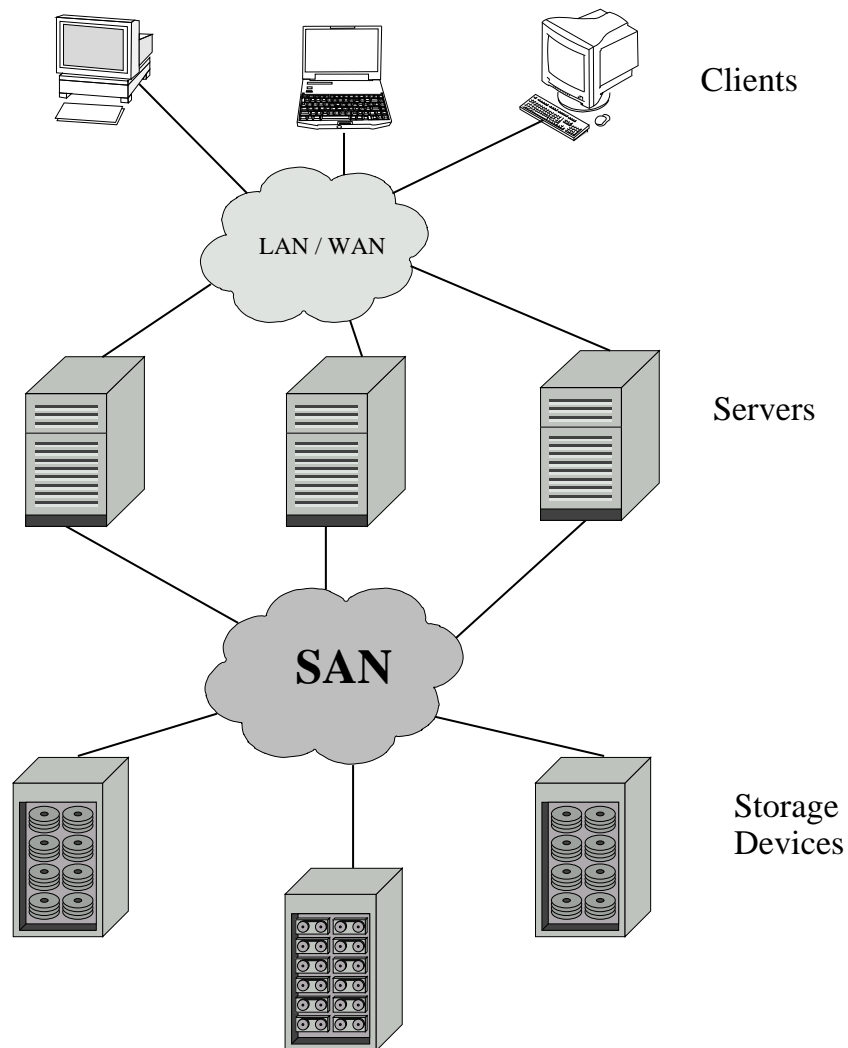
### 7.1 Storage Area Networks

Storage Area Networks (SANs) have emerged as the key solution to address the performance, scalability, reliability and maintainability issues posed by DAS. The SAN is a dedicated network infrastructure that provides meshed, any-to-any connectivity between servers and storage devices.

The introduction of networking concepts and technologies as a replacement of a single, direct connection, redefines the relationship between servers and storage devices and enables the design of new information systems, as depicted in Figure 7.1. Storage resources are now a separated and well-delimited component of the system and servers become the front-end towards the users.

This novel organization of storage resources enables the implementation of new paradigms, providing several benefits [45]:

- **Storage consolidation:** as servers are no-longer directly connected to disks, all the disks can be physically relocated in one or more disk arrays. Disk arrays are devices able to host tens or hundreds of disks. By using the management interface of the disk array, the storage administrator can allocate to each server a proper fraction of the total capacity. Additional space can be provided without disruption by adding disks to the array and reconfiguring it. Storage consolidation can take place even across multiple disk arrays.
- **Remote replication and disaster recovery:** data can be protected from disk faults by using a technique called “mirroring”. A pool of physical disks of equal capacity is combined in a single, virtual disk of the same capacity. Data written to the virtual disk is physically stored on all the disks in the pool. If any of the disks in the pool fail, data is immediately available on the others and the server can continue its operations without disruption. As a SAN can connect devices located tens of Kilometers away,



**Figure 7.1.** An information system employing a SAN

data can be replicated on remote sites, providing protection even in case of disasters, such as natural calamities or terrorist attacks.

- **Server clustering:** a cluster is a set of servers working concurrently on the same set of data. Clustering provides higher performance (as the servers work in parallel) and higher reliability (if one of the server fails, it simply goes out of the cluster). Although complex issues exist at the operating system and application level (inter-process communication, concurrent data access, etc.) a SAN effectively promotes clustering because it allows easy

sharing of common data.

- **LAN-free, server-free backup:** data stored in multiple disk arrays can be backed up directly to large, shared tape drives, without traversing the LAN and without involving the servers. All operations are scheduled and managed from a single, central location.
- **Storage resources management:** the ability to have a consistent and unified view of all the storage devices greatly simplifies monitoring and allocation of resources, as well as provisioning and planning.

In general, the deployment of a SAN enables *virtualization*, i.e. the capability to provide to computing nodes a logical view of available storage resources that is independent of the physical location and the specific characteristics of the devices.

## 7.2 Networking Technologies for SANs

SANs are networks in all respects and present all the features typical of networking technologies. The most important characteristics inherited from the networking world are:

- serial transport, to ship data over long distances at high rates
- data packetization, to achieve high link efficiency and fair sharing of network resources
- addressing schemes that support very large device population
- routing capabilities, to provide multiple, redundant paths between source and destination devices
- a layered architecture, to support the transport of different protocols at the upper layers and the usage of different interfaces at the lower ones.

SANs can be built using different networking technologies, however, it is important to remember that servers, operating systems and applications still expect from the storage interface a “channel-like” behavior, i.e. high-speed, low-latency, error-free communications. Networking technologies used to implement SANs must therefore be carefully chosen and deployed in order to satisfy these strict requirements.

Today the preferred networking technology for SANs is Fibre Channel, although different solutions such as iSCSI (based on TCP/IP and Ethernet[46]) or Infiniband[37] have been proposed.

### 7.2.1 Fibre Channel

Fibre Channel is a multi-purpose, standard-based networking technology, specifically designed for computing environments. Its design is based on the assumption that the transport media (copper cable or optical fiber) is reliable, hence error recovery mechanisms are reduced to a minimum and are mostly

left to upper layer protocols. Data are fragmented and encapsulated in network protocols with minimum overhead, in order to achieve high efficiency. Intermediate nodes guarantee that frames will not be discarded, duplicated or delivered out-of-order under any circumstances. A simple, credit-based mechanism is used for flow and congestion control. These characteristics of the data-path make a full hardware-based implementation feasible. Incoming frames can be processed by end nodes at very high speed and do not incur the latency induced by large reassembly and reordering buffers.

### 7.2.2 Credit-based flow control

Flow control mechanisms are used to regulate the rate at which a transmitter sends frames, in order to achieve efficient bandwidth utilization without overwhelming the receiver. These mechanism represent one of the most important characteristics of a networking technology and have a very strong influence on the design of network devices.

In Fibre Channel networks flow control mechanisms are based on the concept of *credit*. A credit represents the ability of a receiver to accommodate one frame. The receiver grants to the transmitter an initial number of credits, typically proportional to the size of its buffers. The transmitter is authorized to send one frame for each credit it has received; after that it has to stop until it receives more. As soon as the receiver has finished processing an incoming frame (for instance, it has passed it to upper layers) it can free the resources that were used by that frame and grant a new credit.

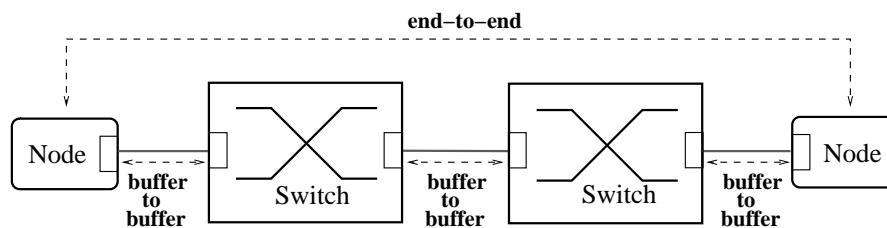


Figure 7.2. Flow control levels

Fibre Channel provides two levels of flow control: “buffer-to-buffer” and “end-to-end”. Buffer-to-buffer flow control takes place between pairs of adjacent ports, such as a link between a node and a switch or between two switches. It operates on all the packets traversing the link, without the capability to discriminate among multiple flows. End-to-end flow control, on the contrary, operates only between end-nodes and is performed per flow, i.e. if a node is receiving multiple flows, it controls each of them separately. The two levels are illustrated in Figure 7.2.

Credit-based flow control mechanisms guarantee that a device accepts incoming frames only if it has the resources to service them. Switches can use such mechanisms to regulate incoming traffic, but once they have accepted a frame, they are not allowed to drop it.



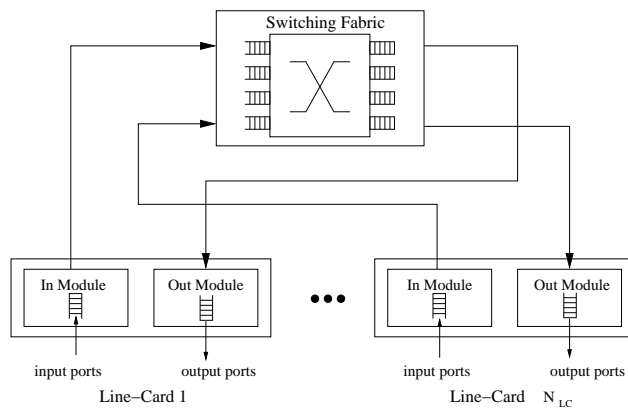


---

## The Switching Architecture

### 8.1 System Overview

The switch is composed by a buffered switching fabric, and a given number of line-cards, comprising input and output buffers, as shown in Fig. 8.1. Every line-card is composed by an input port and an output port (port multiplexing is not considered for simplicity). Line-cards receive packets and store them in input buffers. The switching fabric transfers (multicast) packets from the line-card input buffer to the line-cards hosting the destination output ports, exploiting its intrinsic multicast capability. Switching fabric I/O links are not oversubscribed nor constitute a bottleneck. Backpressure control signals regulate buffer access to avoid data loss.



**Figure 8.1.** Switch architecture: components and communications channels

## 8.2 Line-cards

Line-cards contain two separate buffering stages for multicast packets entering and exiting the switch, named respectively *In-module* and *Out-module*. The In-module memory is organized as a single FIFO queue. Each position in the queue is dimensioned for a maximum transfer unit (MTU). If a smaller packet is enqueued, the residual part of that memory portion remains unusable. This choice potentially results in inefficient use of the In-module memory when dealing with small-size packets, but the buffer management policy can be implemented using simply one counter. Normally this is not a major issue, because line-cards can host a moderately large amounts of memory. The Out-module stores multicast packets received from the switching fabric in a buffer organized as a single FIFO queue. This very fast memory is accessed “per-byte”, hence the number of available positions depends on the size of enqueued packets.

## 8.3 Switching fabric

The switching fabric consists of a crossbar with no internal buffers in cross-points, but with a small single on-chip high-speed FIFO queue at each input and output. Buffers in the switching fabric are needed due to the fully asynchronous switch behavior. The crossbar may have a moderate internal speed-up  $K$  to mitigate the effect of Head-of-the-line (HOL) blocking of FIFO queues; however, in the simulation analysis, we consider  $K = 1$ . Fabric output queues are larger than fabric input queues to sustain temporary overload conditions. Both input and output queues are accessed per-byte to maximize space efficiency.

Each fabric output has a *fabric scheduler* that controls access from fabric inputs. When an input wants to be connected to an output, it sends a *request* to the corresponding output fabric scheduler. The output scheduler sends *grants* to inputs. In case multiple inputs request the same output, the output scheduler solves the contention according to a round-robin (or random) policy.

The crossbar has an internal multicasting capability: it can replicate a packet to multiple outputs at the same time with no extra cost. The asynchronous behavior poses different challenges with respect to synchronous slotted switching when dealing with multicast traffic. In synchronous switching, decision are taken synchronously by all output schedulers at time slot boundaries. As such, no output remains unnecessarily idle if enough traffic is available at inputs. When considering asynchronous behavior, output schedulers make independent decisions at different times. However, some sort of grant synchronization at inputs can be useful, to avoid sending multicast packets only according to a multi-copy scheme, i.e.. as independent unicast packets. Indeed, the multi-copy approach is known to be an inefficient scheduling technique. For example, when considering as an admissible traffic pattern a single

broadcast flow in overload at a given input, the multi-copy approach gives a maximum throughput of  $1/N$ . On the other hand, waiting to gain access to all the intended outputs before transmitting a packet can be counterproductive, because it forces outputs that have already granted access to stay idle while the other outputs become free.

To exploit the benefits of crossbar replication without compromising efficient usage of output ports, the formerly proposed scheduling scheme [47] was based on three phases:

- *Waiting phase*: every input sends a request for every output in the fanout set of the HoL packet stored in the fabric input buffer. Each input collects the grants received from the outputs until a timeout  $T$  expires;
- *Multicast transmission phase*: After timeout expiration, each input sends the multicast packet to every output that granted its request, using the internal multicast capability of the crossbar;
- *Unicast transmission phase*: If the input has not received the grants from all outputs during the waiting phase, it sends an individual copy of the packet to the remaining destinations as soon as each output grants the request.

The timeout is used to obtain a “grant synchronization” effect at inputs: inputs waiting for grants from outputs belonging to the multicast packet fanout set may better exploit the fabric multicast capability if more outputs grant the request during timeout expiration. Indeed, the larger the number of received grants, the smaller the number of transmissions required to transfer a multicast cell to the outputs in the fanout set. On the other hand, while waiting for timeout expiration, no transmission occur: thus, the timeout value must be carefully set to balance these two effects on performance. In other words, strictly enforcing a no-fanout splitting policy, i.e., a multicast packet is transferred only once, when all the outputs in the fanout set are available, may induce performance degradation due to blocking. On the other hand, splitting the multicast packet in too many transmissions, when using a fanout splitting policy, increases too much the load on the switching fabric, thus, reducing performance.

### 8.3.1 Improvement of multicast scheduling

The modification we propose to enhance switch performance is to substitute the unicast transmission phase with a number of multicast transmission phases until the multicast packet is fully transmitted. This modification requires a minor complexity increase, and provides significant benefits. Note that no timeout expiration is necessary after the first multicast transmission phase, since inputs aggregate grants received from outputs during the packet transmission time. Furthermore, the adoption of the multicast transmission phase

only, permits, as shown later, to avoid the use of the timeout to synchronize grants, a parameter whose value should be set properly to obtain good performance.

This scheduler is named round robin scheduler in the remainder of the chapter, since multiple requests received by an output scheduler while the output is engaged in a packet transmission are served according to a round robin order. It is well known that round-robin (or random) schedulers do not perform particularly well, since no information on the relative importance or urgency of packets is used when selecting the input to which the request is granted. A better solution can be to use some “weighted” metric when sending requests from inputs and when selecting the input to which to issue the grant at outputs. Examples of weighted schedulers defined for synchronous switches are LQF (Longest Queue First) [10] for unicast traffic, WBA (Weight Based Arbiter) [36] and GS (Greedy Scheduler) [29] for multicast traffic. One natural choice of weight is the queue length (LQF), since the longer the queue the higher the input load; the scheduler tries to favor highly-loaded inputs to improve performance. However, since the maximum queue length is finite, this weight is significant only when losses are not experienced, i.e., in low-medium loads. On the contrary, all the queues experiencing losses have constant queue length, independently from their congestion level and the queue length metric would not help in this scenario. When presenting WBA, several metrics were proposed and compared. Given the similar performance provided by the various metrics, we choose the weight equal to the number of inputs minus the packet fanout plus the packet age (measured as the difference between the current time minus the time at which the packet entered the queue). Indeed, packet age, although being a complex metric to be managed, permits to discriminate packets experiencing long starvation periods. Moreover, taking into account also the packet fanout permits to favor multicast packets with large fanout sets, packets known to be more difficult to schedule. Finally, we also considered the GS weight, the product of the queue length by the actual fanout size of the packet at the head of the queue. In contrast with the WBA metric, the GS metric does not require any packet delay computation.

However, two major differences can be highlighted in asynchronous switching with respect to the more traditional synchronous scenario. First, in synchronous switches, all outputs receive weight information at the same time, at slot boundaries, from all inputs. As such, coordination among output selectors can be envisioned to optimize packet selection. Indeed, since all outputs make an input selection in each time slot, it is more likely that several outputs select the same input request if the associated weight is much larger than other weights. Second, due to the packet segmentation process at inputs and to the cell-based packet transfer in the switching fabric, several requests with different weights are sent for a given packet in consecutive time slots, until the multicast packet is fully extracted from the input queue. None of these two properties hold in asynchronous switches, since each output selects a new request independently, when a packet transmission ends, and the request

is issued only once, when the packet reaches the head of the corresponding FIFO queue. As such, weighted metrics could be less effective than in the synchronous case.

Finally, also multicast schedulers like MRR cannot be easily used in an asynchronous scenario. Indeed, MRR is based on the idea of keeping, in all outputs, a common pointer (a modulo  $N$  counter) to inputs. The pointer is used to preferentially grant, according to a round-robin scheme, requests incoming from inputs in a given time slot. This common reference clearly favors the possibility of selecting the same input at many outputs, thus preserving as much as possible the no-fanout splitting property of the scheduler. This scheduler cannot be used in asynchronous switches, since no coordination can be easily enforced among outputs.

To summarize, besides the round-robin scheduler, three weighted schedulers running on three different metrics are also studied:

- LQF metric: each request contains the length of the FIFO queue at the corresponding input;
- WBA metric: the weight is equal to the number of inputs minus the cell fanout plus the packet age;
- GS metric: the weight is the product of the queue length by the actual fanout size of the cell at the head of the queue.

## 8.4 Control mechanisms for lossless delivery

To support lossless delivery, the switch adopts an internal backpressure mechanism that regulates access to buffers to prevent overflow. When the buffer occupancy overcomes a *high threshold*, a backpressure signal is activated to block packet transmissions from upstream buffering stages. When the buffer occupancy becomes smaller than a *low threshold*, the backpressure signal is deactivated and transmission can restart. In case of persistent congestion, all the buffers in the data path eventually fill-up and the backpressure signal propagates back to the source(s).

Four backpressure signals are available:

1. from the Out-modules to the fabric output queues;
2. internally to the fabric, from fabric output queues to fabric input queues;
3. from fabric input queues to the In-modules;
4. from the In-module to the input ports.

Backpressure prevents packet losses: however it is not selective, i.e. it blocks all flows, even those which are not responsible for congestion. In [6] we illustrated the benefits achieved by controlling individually unicast flows with centralized arbitration. The same result cannot be easily obtained for multicast, because the number of possible flows traversing the switch grows exponentially (rather than quadratically) with the number of ports  $N$ . This

implies that switch resource can be hardly assigned per-flow. In particular, both on the ingress and egress side of line-cards packets are stored in a single FIFO queue, regardless of their fanout set.

## Performance Results

### 9.1 Performance results under multicast traffic conditions

We show performance results based on simulation runs exploiting a proprietary simulation environment developed in C language. Statistical significance of the results are assessed by running experiments with an accuracy of 1% under a confidence interval of 95%.

Unless otherwise specified, we refer to a switch with  $N = 16$  input and output ports, where all input and output lines run at the same data rate, normalized to 1, and no internal speedup is available. When backpressure is enabled, the high threshold, which triggers the backpressure signal, is set to the buffer size, the low threshold is set to 80% of the buffer size.

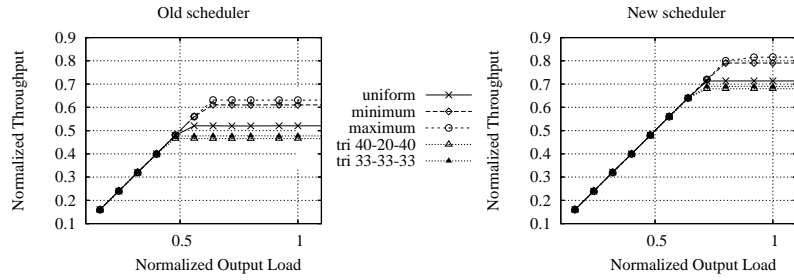
The average amount of offered traffic at each input (output) is called the input (output) load. Input (output) loads are normalized to line rates: a load equal to 1 means a fully utilized input (output) line. The traffic at the input of a switch is said to be *admissible* if no input load is larger than 1, and no output load is larger than 1.

We first consider Bernoulli arrivals with uniform multicast traffic distribution, both in terms of input/output port distribution and fanout distribution, and then we also examine a Bernoulli arrival process in a gathered scenario, as described in 5.3.

Multicast applications often generate sustained and long-lasting flows, that may only engage few inputs and several outputs at a given router or switch.

Five packet size distributions are considered:

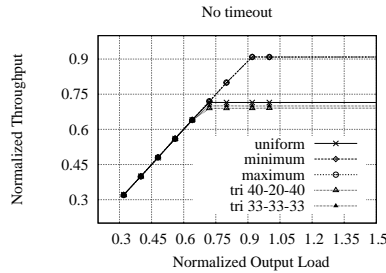
- constant packet size, all packets of minimum size (mTU, minimum transfer unit of 120 bytes),
- constant packet size, all packets of maximum size (MTU=2000 bytes),
- uniform packet size, ranging from mTU to MTU
- trimodal packet size (120 bytes, 1040bytes, 2000 bytes) with probability 40%, 30%, 40% respectively,



**Figure 9.1.** Performance comparison between the old and the new switching fabric scheduler

- trimodal packet size (120 bytes, 1040bytes, 2000 bytes) with probabilities 33%, 33%, 33% respectively.

We first show the performance benefit of the newly proposed multicast round robin fabric scheduler; backpressure is activated among all buffering stages. Fig. 9.1 clearly shows that the new scheduler outperforms the old scheduler. Constant packet size permit to obtain higher throughput mainly thanks to a better efficiency in the use of input fabric FIFO queues. Increasing the packet size variance by using trimodal or uniform distributions worsen performance. This is a peculiar behavior of asynchronous architecture, whereas cell-based synchronous switches suffer less this impairment, thanks to the packet segmentation process at input ports.

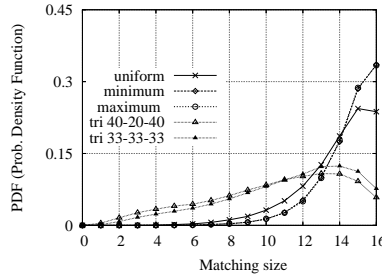


**Figure 9.2.** Performance when no timeout is adopted in the fabric scheduler

Fig. 9.2 shows that when adopting the new multicast scheduler, the initial timeout used to synchronize grants is not needed. Indeed, whereas performance for trimodal and uniform traffic distributions are not modified, since the performance limitation is due to the inability of filling up input fabric FIFO queues, performance increase significantly, reaching about 90% of link capacity, when using fixed size packets. Indeed, when using fixed size packets, due to the buffering stage at fabric input, inputs tend to synchronize packet



transmissions. For example, when a broadcast packet is scheduled for transmission, all output ports become free at the same time. Being all the packets of the same fixed size, all successive transmissions become exactly synchronized, increasing switch performance. If a timeout expiration is used in the first phase of the multicast scheduling algorithm, this synchronization effect is partially lost, since each multicast packet requires a different number of transmissions to be completely transferred to the outputs belonging to the fanout set. In summary, using a timeout in the scheduling phase either does not provide performance advantages or worsen performance. Thus, no timeout is used in the multicast scheduling phase when adopting the round robin scheduler.



**Figure 9.3.** Matching size distribution in overload under uniform traffic

Again, the asynchronous architecture suffers the increase on the packet size variance due to the independent behavior of schedulers at output ports. This is confirmed by the “matching” size distribution shown in Fig. 10.2. No matching can be defined in an asynchronous architecture. However, we periodically sample the switching fabric configuration counting the number of active input/output connections: we call this number matching size. The scheduler difficulty in creating large size matching is clearly increasing with increasing variance in the packet size distribution. Thus, the saturation throughput is directly tied to the variance of the packet size distribution.

In Fig. 10.4, the beneficial effect of backpressure is shown when dealing with non-admissible traffic. Indeed, whereas backpressure activation or deactivation makes no evident difference when the output load is below or close to 1, in deep overload the absence of a backpressure mechanism induces higher losses for trimodal packet size distributions (and marginally higher losses for uniform packet size distribution).

This is a rather counter intuitive behavior. Indeed, when backpressure is active, the packet size distribution in the input FIFO buffers at fabric input is kept constant, regardless of input load, since the source is blocked until 80% of the buffer becomes available. This justifies why no differences are evident when increasing the input load. On the contrary, when backpressure is inac-

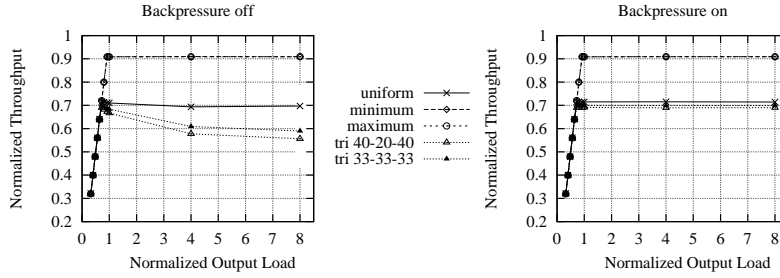


Figure 9.4. Backpressure effect

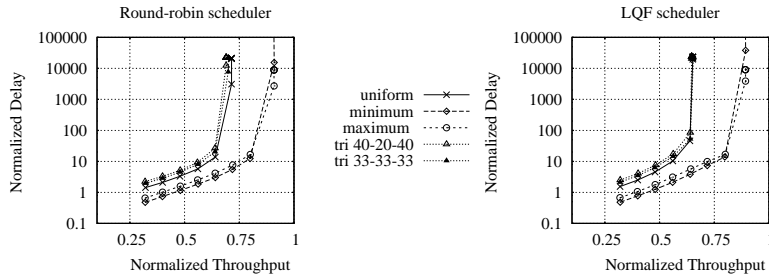


Figure 9.5. Packet delays for two different multicast schedulers

tive, in deep overload, small packets have a higher chance of being stored in input FIFOs at fabric inputs. Indeed, when a small packet is transferred, only small packets can be stored in the buffer; when a large packet is transferred, it is enough to store few small packets in the buffer to prevent the possibility of storing a new large packet. Thus, a large number of small packets is stored in fabric buffers, and the average size of packets stored in the input FIFOs decreases as the load increases. This should intuitively lead to an increase in throughput when backpressure is inactive, since the switch should behave similarly to the case of fixed packet size, being the packet size variance decreasing as the input load increases.

However, consider a case when a large packet is transferred from a given input to a set of outputs, and suppose that many small packets are stored in other input queues. Suppose also that small packets are blocked due to contention. This blocking behavior induces a significant throughput decrease, since the transmission time of a large packet with respect to the transmission time of a small packet is significant. In deep overload, this event is more likely to occur with respect to the case of variable packet size distribution, since many small packets are stored in input buffers. Since performance losses are more evident in this “blocking” scenario when inputs store many small packets, this justifies the throughput decrease.

In summary, activating backpressure does not provide performance penalties and helps stabilizing system performance in overload. As such, we will activate the backpressure mechanism in all subsequent simulations. To understand if the penalty provided by the packet size variance is an intrinsic feature of asynchronous architectures or if it depends on the adopted scheduler, we run simulations with all the “weighted” schedulers previously described. Results are reported in Fig. 9.5 as normalized packet delays, i.e. delay normalized to the packet size, for uniform multicast traffic. We report the results for the LQF scheduler, because no difference were visible by varying the weight metric. The weighted metric does not provide any benefit with respect to the round-robin scheduler; sometimes, performance are even worse, as with trimodal and uniformly variable packet size in both delay and saturation throughput. This results confirms similar observations presented in [29] for synchronous architectures. Performance advantages for more complex weighted schedulers, such as the GS scheduler, are evident only when using more than one FIFO queue at each input, a queue architecture not studied in this paper. Besides, the same general trend by which asynchronous architectures suffer for the packet size variance is maintained.

The same conclusion can be drawn when examining the switch under gathered traffic, in Fig. 9.6. As expected, in this scenario the maximum achievable throughput is drastically reduced. The general phenomenon of better performance for fixed packet size is even highlighted by this traffic pattern. Results not reported confirm that schedulers based on weighted metrics do not provide evident benefits to switch performance.

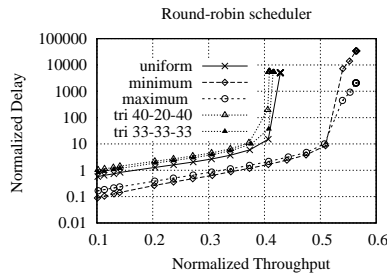


Figure 9.6. Packet delays for gathered traffic

Finally, in Fig. 9.7 we report the normalized delay for uniform and gathered traffic when adopting a speedup  $k = 2$  for the round robin scheduler. Performance increase, but only for uniform traffic the maximum throughput reaches one regardless of packet size distribution. For gathered traffic, only with fixed packet size the maximum throughput is achieved. The packet size variance penalty is still visible in terms of packet delays at low-medium load.

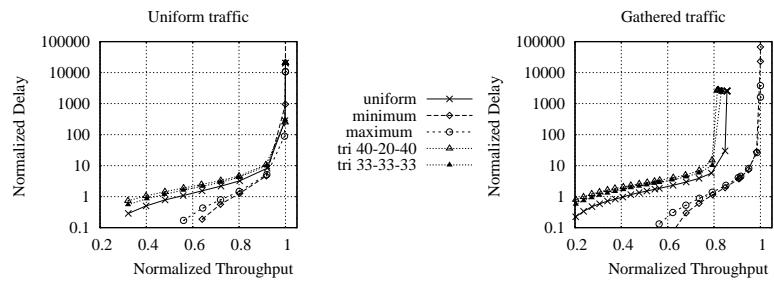


Figure 9.7. Packet delays for speedup  $k = 2$ , for uniform and gathered traffic

**Synchronous vs Asynchronous Switching**



## Synchronous versus Asynchronous under Multicast Traffic

### 10.1 Introduction

IQ architectures were proposed many years ago as the only viable solution to build high-speed large-size packet routers. Traditional data network IQ switches and routers operate in a *synchronous* fashion: time is divided in intervals of equal size, called time-slots, and all modules (line-cards and switching fabric) have a common time reference. Variable-size packets are segmented into fixed-size data units called *cells*, transferred synchronously through the switching fabric within a time-slot and reassembled at the output line-cards.

The interest in asynchronous switching architectures has recently raised e.g., in the Storage Area Networks (SANs) scenario. Asynchronous behavior may provide advantages in terms of scalability, cost and simplicity [4]. In an asynchronous switch, line-cards and the switching fabric run on independent clock domains. As such, global clock distribution is not needed, thus avoiding a very complex task especially when the system is distributed over multiple racks. Furthermore, no synchronized transmission through the switching fabric is required and variable-length packets can be supported natively, without the need for segmentation and reassembly buffers. Finally, fabric arbitration can be simplified because output contentions can be solved independently, without employing centralized scheduling algorithms. The distributed nature of asynchronous architecture raises some concern against their performance.

Switch architectures are compared under multicast traffic. Although multicast traffic represents only a small portion of today traffic, multicast support in switches and routers is a must. Indeed, multicast traffic in SANs enables applications such as disaster recovery, remote data replication and distributed multimedia systems. Furthermore, besides being used in multi-media data applications, broadcast traffic is important in Ethernet switches both to support applications that rely on LAN broadcast capability, e.g., ARP, as well as to internally distribute forwarding tables. Finally, whereas some comparisons were made for unicast traffic [48], to the best of our knowledge no results are available for multicast traffic.

Multicast packets are characterized by their fanout set, the set of output ports to which they are directed. The packet fanout is the number of different destinations of a multicast packet, i.e., the cardinality of the fanout set.

## 10.2 System Model

Although synchronous and asynchronous architectures may have slightly different characteristics derived from technological constraint, we wish to abstract as much as possible the system model to obtain two architectures comparable in a fair way.

We focus on an  $N \times N$  switch,  $N$  being the number of line-cards, with a buffer-less crossbar with intrinsic multicast capability as switching fabric: it can replicate a packet to multiple outputs at the same time with no extra cost.

A single FIFO queue is available in each line-card. Although this choice introduces the well-known HoL (Head of the Line) blocking phenomenon, it is normally considered as a reasonable scenario for multicast traffic. Indeed, to completely avoid HoL blocking, at each input port,  $2^N - 1$  separate queue, one per multicast flow, would be needed. This is an unreasonable number in large size switches. Dealing with a reduced number of queues  $k \ll 2^N$  is a viable solution, but it introduces either the problem of mapping multicast flows to queues or out of sequence delivery.

To support lossless delivery, the switch may adopt an internal backpressure mechanism that regulates access to buffers to prevent overflow. When the buffer occupancy overcomes a *high threshold*, a backpressure signal is activated to block packet transmissions from upstream buffering stages. In this situation, packet generation is stopped. When the buffer occupancy becomes smaller than a *low threshold*, the backpressure signal is deactivated and transmission can restart.

We consider two-phase request-grant multicast schedulers. One Input Selector (IS) and one Output Selector (OS) exist in each line-card. In the request phase, an IS issues a requests for the HoL multicast packet to all OS whose output port is in the packet fanout set. In the grant phase, OSs solve request contentions, normally independently, by choosing a single request to grant. The various multicast schedulers associate different metrics, if any, to requests and exploit different contention resolution algorithms at OSs.

When considering asynchronous behavior, ISs and OSs schedulers make independent decisions at different times, since requests/grants are issued when the input/output becomes idle, i.e., when a packet transmission ends. In synchronous switching, requests and grants are issued synchronously by all ISs/OSs at time slot boundaries, since all inputs and outputs become free at the same time, being engaged in transmissions that last exactly one time slots. Centralized single-chip scheduler implementation is largely dominant in synchronous switches; therefore, coordination among ISs and OSs could be easily



envisioned. However, most multicast schedulers do not exploit this coordination feature, which is indeed often used in unicast schedulers, for example when iterating the scheduling phase several times in a given time slot.

Being decisions made independently by each IS/OS in both cases, the main difference between asynchronous and synchronous scheduling behavior relies in the timing at which scheduling decisions are made. However, whereas in synchronous switches all inputs and outputs are always available at time slot boundaries, in asynchronous switches, at high load, when an input and a set of outputs becomes available, the chance of finding other available inputs is relatively small. As such, it is likely that the same input/outputs is selected again for a new transmission. Obviously, this effect is mitigated by multicast transmissions with a large fanout set, since in this case many outputs become idle at the same time, and by the fact that the next multicast packet at the HoL queue in the idle input may have a fanout set different from the previous one. Still, a performance penalty could be expected in the asynchronous case.

Note that, in asynchronous switches, some sort of grant synchronization at inputs can be useful, to avoid sending multicast packets only according to a multi-copy scheme, i.e. as independent unicast packets. an approach known to be an inefficient scheduling technique. On the other hand, waiting to gain access to all the intended outputs before transmitting a packet can be counterproductive, because it forces outputs that have already granted access to stay idle while the other outputs become free. We disregard all these issues to keep a relatively simple independent scheduling at each port.

First, we consider the random (RND) scheduler. When a packet transmission ends, one among the multiple requests received by an OS while the output was engaged in a packet transmission is randomly chosen. Random schedulers have performance limitations, since no information on the relative importance or urgency of packets is used when selecting the input to which the request is granted. To overcome these limitations, “weighted” metrics were proposed. A weight is associated with each request at each IS and it is used to select the input to which to issue the grant at OSs. As in Chapter 8.3, we consider weighted schedulers such as LQF (Longest Queue First) [10] for unicast traffic, WBA (Weight Based Arbiter) [36] and GS (Greedy Scheduler) [29] for multicast traffic remembering that:

- LQF metric: each request contains the length of the FIFO queue at the corresponding input;
- WBA metric: the weight is equal to the number of inputs minus the cell fanout plus the packet age;
- GS metric: the weight is the product of the queue length by the actual fanout size of the cell at the head of the queue.

As we already said, in synchronous switches, all outputs receive weight information at the same time, at slot boundaries, from all inputs and, due to the packet segmentation process at inputs and to the cell-based packet transfer in the switching fabric, several requests with different weights are

sent for a given packet in consecutive time slots, until the multicast packet is fully extracted from the input queue. None of these two properties hold in asynchronous switches, since each output selects a new request independently, when a packet transmission ends, and the request is issued only once, when the packet reaches the head of the corresponding FIFO queue. As such, weighted metrics could be less effective than in the synchronous case.

### 10.3 Performance results

Performance results are based on simulation runs. Statistical significance of the results is assessed by running experiments with an accuracy of 2% under a confidence interval of 95%.

We refer to a switch with  $N = 16$  input and output ports, where all input and output lines run at the same data rate, normalized to 1. When backpressure is enabled, the high threshold, which triggers the backpressure signal, is set to the buffer size, the low threshold is set to 80% of the buffer size. Each FIFO queue size is fairly large (10000 packets) to examine a situation close to the asymptotic case of infinite buffer size. The average amount of offered traffic at each input (output) is called the input (output) load. Input (output) loads are normalized to line rates: a load equal to 1 means a fully utilized input (output) line. 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.

As usual, we consider Bernoulli arrivals with uniform multicast traffic distribution, both in terms of input/output port distribution and multicast flow distribution, and a Bernoulli arrival process in a gathered scenario, where the traffic is gathered over few active input ports ( $M = 5$ ) and equally distributed over all  $N = 16$  output ports, with a fanout set chosen according to a non-uniform binomial distribution, with mean fanout  $h_m = 3.66$  [29].

Four packet size distributions are considered:

- minimum constant packet size, all packets of 120 bytes,
- maximum constant packet size, all packets of 2000 bytes,
- uniform packet size, ranging from 120 to 2000 bytes
- trimodal packet size (120 bytes, 1040bytes, 2000 bytes) with probability 40%, 20%, 40% respectively,

We compare in Fig. 10.1 the asynchronous architecture with a traditional synchronous cell-based switch under the RND scheduler. Packet delays are normalized to the packet size. When a packet reaches an empty switch, it is immediately transmitted with simply a store and forward packet delay in both architectures.

No major differences are evident in terms of delays at low loads in Fig. 10.1. The synchronous switch shows a higher saturation throughput with respect to the asynchronous switch when considering trimodal and uniform packet size

distribution. The segmentation effect helps in compensating the increasing variance in the packet size distribution, whereas the asynchronous architecture suffers this effect due to the independent behavior of schedulers at output ports. This is confirmed by the matching size distribution shown in Fig. 10.2 on the left for the asynchronous case. Formally, matching cannot be defined in an asynchronous architecture. We periodically sample the switching fabric configuration counting the number of active input/output connections: we call this number matching size. The scheduler difficulty in creating large size matching is clearly increasing with increasing variance in the packet size distribution. Thus, the saturation throughput is directly tied to the variance of the packet size distribution in the asynchronous case.

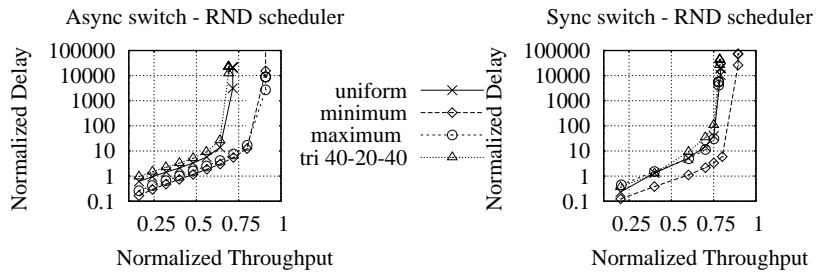


Figure 10.1. Packet delays for RND scheduler under uniform traffic

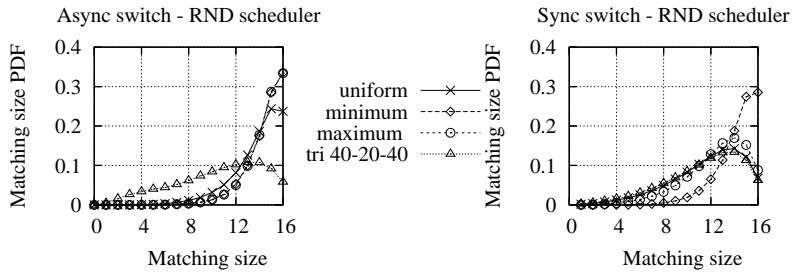


Figure 10.2. Matching size distribution in overload under uniform traffic

However, when considering fixed size packet distributions, the asynchronous switch shows a slightly higher saturation throughput. This is due to the synchronization effect induced by the fixed size packets. Indeed, constant packet size permit to obtain higher throughput mainly thanks to a better efficiency in the use of input FIFO queues. Note that this effect does not hold in synchronous architectures with fixed maximum packet size distribution. Indeed, even if the centralized synchronous multicast scheduler makes independent de-

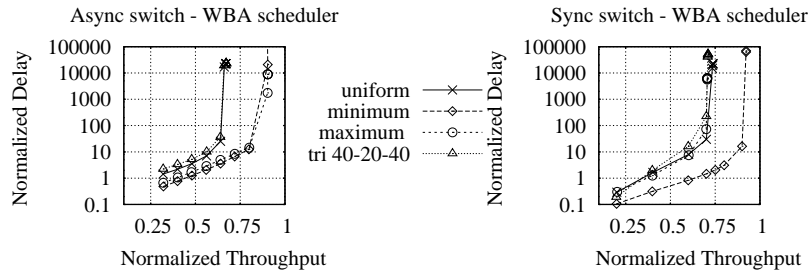


Figure 10.3. Packet delays for WBA scheduler under uniform traffic

cision at each time slot, when two or more large packets difficult to schedule, i.e., creating output conflicts, reach the input FIFOs head-of-the-line, there is a large chance that other conflicting packets reach the head-of-the-line at other inputs. This situation lasts for many time slots, and it becomes self-sustaining, since the longer the conflicting period, the higher the chance of having an increasing number of conflicting packets. Note that the “new” conflicting packets will contend with the “old” one for many time slots. Thus, packet transmission times increase significantly and throughput loss becomes significant. This phenomenon disappears for fixed minimum size packets, since the conflicting state lasts a single time slot. In asynchronous architectures, once the conflict is solved, packets are completely transferred toward the destination, and there is no self-sustaining conflicting behavior.

In summary, increasing the packet size variance, by using trimodal or uniform packet size distributions, worsen performance in asynchronous architectures, whereas cell-based synchronous switches suffer less this impairment, thanks to the packet segmentation process at input ports, but suffer for large packet size.

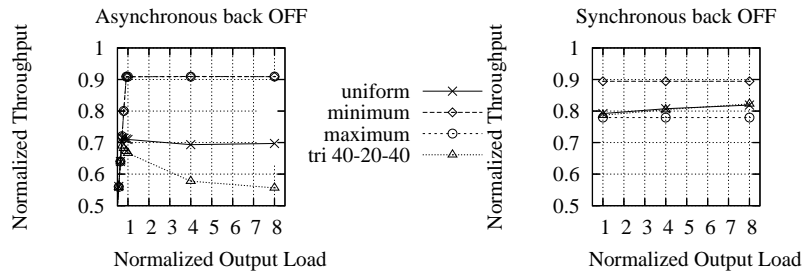


Figure 10.4. Backpressure effect

In Fig. 10.4, the beneficial effect of backpressure for asynchronous architectures is shown when dealing with non-admissible traffic. Indeed, whereas

backpressure activation or deactivation makes no evident difference when the output load is below or close to 1, in deep overload the absence of a backpressure mechanism induces higher losses for trimodal packet size distributions (and marginally higher losses for uniform packet size distribution).

This is a rather counter intuitive behavior. Indeed, when backpressure is active, the packet size distribution in the input FIFO buffers at fabric input is kept constant, regardless of input load, since the source is blocked until 80% of the buffer becomes available. This justifies why no differences are evident when increasing the input load. On the contrary, when backpressure is inactive, in deep overload, small packets have a higher chance of being stored in input FIFOs at fabric inputs. Indeed, when a small packet is transferred, only small packets can be stored in the buffer; when a large packet is transferred, it is enough to store few small packets in the buffer to prevent the possibility of storing a new large packet. Thus, a large number of small packets is stored in fabric buffers, and the average size of packets stored in the input FIFOs decreases as the load increases. This should intuitively lead to an increase in throughput when backpressure is inactive, since the switch should behave similarly to the case of fixed packet size, being the packet size variance decreasing as the input load increases.

However, consider a case when a large packet is transferred from a given input to a set of outputs, and suppose that many small packets are stored in other input queues. Suppose also that small packets are blocked due to contention. This blocking behavior induces a significant throughput decrease, since the transmission time of a large packet with respect to the transmission time of a small packet is significant. In deep overload, this event is more likely to occur with respect to the case of variable packet size distribution, since many small packets are stored in input buffers. Since performance losses are more evident in this “blocking” scenario when inputs store many small packets, this justifies the throughput decrease.

In summary, activating backpressure does not provide performance penalties and helps stabilizing system performance in overload.

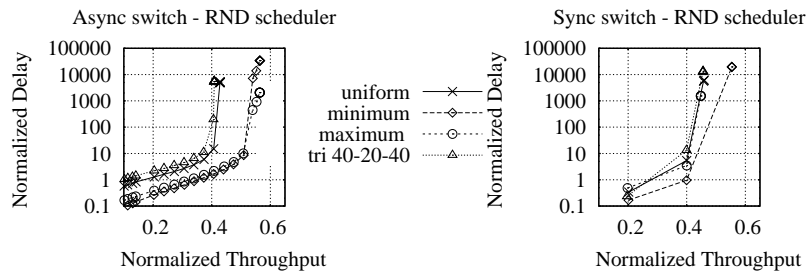


Figure 10.5. Packet delays for RND scheduler under gathered traffic

To understand if the described characteristics are an intrinsic feature of the studied architectures or if they depend on the adopted scheduler, we run simulations with all the “weighted” schedulers previously described. We report the results for the WBA scheduler only, because no difference were visible by varying the weight metric. In asynchronous architectures, the weighted metric does not provide any benefit with respect to the random scheduler; sometimes, performance are even worse, as with trimodal and uniform packet size. In the synchronous case, an improvement is visible only when dealing with minimum packet size, i.e. when the cell is matched to the packet size. For other packet size distributions WBA worsen performance. The same general trend by which asynchronous architectures suffer for the packet size variance and synchronous ones provide performance advantages only for fixed packet size matched to the cell size is maintained.

This results confirms similar observations presented in [29] for synchronous architectures. Performance advantages for more complex weighted schedulers, such as the GS scheduler, are evident only when using more than one FIFO queue at each input, a queue architecture not studied in this paper.

The same conclusion can be drawn when examining the switch under gathered traffic, in Fig. 10.5. As expected, in this scenario the maximum achievable throughput is drastically reduced. The general phenomena are confirmed or highlighted by this traffic pattern. Results not reported confirm that schedulers based on weighted metrics do not provide evident benefits to switch performance also in the gathered traffic scenario.

## Parte IV

---

## Conclusion





## Conclusion

In the first part of this thesis we have dealt with the problem of fully distributed multi-chip scheduling implementation in input queued switches. Multi-chip implementation implies that i) decisions taken by input and output selectors should be independent, being the selectors realized in different devices and ii) any information exchange among selectors implies a RTT delay, which may be larger than few tens of slot time.

First of all, we have proposed a new scheduler for unicast traffic, named SRR, i) is suited to a fully distributed implementations, ii) does not require any complexity increase as a function of increasing RTTs, iii) does not require any iteration to improve matching selection.

SRR shows performance comparable with those of a previously proposed distributed scheduler, a modified version of DRRM able to deal with RTT among devices. This is a remarkable achievement, since the modified DRRM scheme is not suited to a fully distributed implementation, requiring i) all output selectors in the same device to permit iterations, ii) a number of pointers and counters linearly increasing with RTTs. The only SRR penalty is the need of keeping ordered by queue length the VOQs at input selectors, a relatively easy task given that at most one cell can arrive and at most one cell can depart in each time slot.

After unicast, we have dealt with multicast, that is, as known, more complex to schedule than unicast. The proposed modified multicast scheduler, named IMRR, is suited to a fully distributed multi-chip implementation, and shows performance improvements with respect to previously proposed multicast schedulers directly adapted to the multi-chip scenario. Unfortunately, all algorithms show increased average delays at low loads for increasing RTTs, a problem that would be nice to study and solve in the future.

At the last we devoted our attention to the problem of scheduling concurrently unicast and multicast traffic in an input-queued switch. We have defined a integration scheme based on frame, which using schedulers as SRR and iMRR. The use of these two schedulers makes the frame scheme suitable to a fully distributed implementation, does not require any complexity

increase as a function of increasing RTTs and does not require any iteration to improve matching selection. On the contrary the FILM scheme requires an increase of complexity as a function of increasing RTTs and it use iteration.

The use of the dynamic frame allows the frame scheme to get good performance for any type of traffic in input and for each MCF's value. The definition of the dynamic frame does not increase the complexity of the frame scheme, which remains  $O(N^2)$ .

We have shown how the Frame Scheme obtain performance comparable to the FILM scheme in the case of RTT different from zero. This is a remarkable achievement because the FILM scheme is not suited to a fully distributed implementation.

In the second part an asynchronous loss-less switching architecture was described, and its performance under multicast traffic was studied. Simulations are used to analyze switch performance under various traffic patterns and for different schedulers.

Asynchronous architectures suffer variability in packet size distribution, which reduces switch performance both in terms of throughput and delays. This general trend holds regardless of the considered scheduler and the multicast traffic pattern. More complex schedulers based on "weighted" metrics do not provide evident benefits in this scenario, where a single FIFO queue is available. A moderate speedup helps in reducing this performance penalty. Backpressure mechanisms may be beneficial to stabilize performance in overload and do not penalize switch performance in any of the examined scenarios.

Finally, in the last part, we study the differences between asynchronous and synchronous system. Asynchronous architectures suffer variability in packet size distribution, which reduces switch performance both in terms of throughput and delays. This general trend holds regardless of the considered scheduler and the multicast traffic pattern. More complex schedulers based on "weighted" metrics do not provide evident benefits in this scenario, where a single FIFO queue is available. A moderate speedup helps in reducing this performance penalty. Backpressure mechanisms may be beneficial to stabilize performance in overload and do not penalize switch performance in any of the examined scenarios.

In summary, synchronous or asynchronous architectures provide comparable performance. Asynchronous schedulers suffers from an increase in the variance of packet size distribution, whereas synchronous switches suffer from large packet size with respect to the cell size. Given the technological advantages of asynchronous switching as outlined in the Introduction, asynchronous switching seems a competitive solution with respect to the more traditional synchronous approach. Note that we have disregarded issues related to the fixed size penalty, which is implicit in synchronous switches. Taking this into account would make asynchronous switching even more competitive.

---

## Riferimenti bibliografici

1. H. J. Chao, C. Lam, and E. Oki, *Broadband Packet Switching Technologies*. John Wiley & sons, Sept. 2001.
2. A. Pattavina, *Switching Theory*. John Wiley & sons, 1998.
3. J. Y. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*. Boston, MA: Kluwer, 1990.
4. M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos, "Variable packet size buffered crossbar (cicq) switches," in *Proc. IEEE International Conference on Communications (ICC 2004)*, vol. 2, (Paris, France), pp. 1090–1096, June 20–24, 2004.
5. K. Yoshigoe and K. Christensen, "A parallel-pollled virtual output queued switch with a buffered crossbar," in *Proc. IEEE Workshop on High-Performance Switching and Routing HPSR 2001*, (Dallas, TX), pp. 271–275, May 2001.
6. A. Bianco, P. Giaccone, E. M. Giraudo, F. Neri, and E. Schiattarella, "Performance analysis of storage area network switches," in *Proc. IEEE Workshop on High-Performance Switching and Routing HPSR 2005*, (Hong Kong), 2005.
7. W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. San Francisco, CA: Morgan Kaufmann, 2003.
8. N. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space division switch," *IEEE Trans. Commun.*, vol. 35, pp. 1347–1356, Dec. 1987.
9. Y. Tamir and G. Frazier, "High performance multi-queue buffers for vlsi communication switches," in *Proc. 15th Ann. Symp. Comp. Archi.*, pp. 343–354, June 1988.
10. N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Trans. Commun.*, vol. 47, pp. 1260–1267, Aug. 1999.
11. S. T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input output queued switch," *IEEE J. Sel. Areas Commun.*, vol. 17, pp. 1030–1039, June 1999.
12. I. Stoica and H. Zhang, "Exact emulation of an output queueing switch by a combined input output queueing switch," in *Proc. 6th IEEE/IFIP IWQoS '98*, (Napa Valley, CA), pp. 218–224, May 1998.

13. J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speed-up," in *Proc. IEEE INFOCOM 2000*, vol. 2, (Tel Aviv, Israel), pp. 556–564, Mar. 2000.
14. C. Minckenberg, R. Luijten, F. Abel, W. Denzel, and M. Gusat, "Current issues in packet switch design," *ACM Computer Commun. Rev.*, vol. 33, pp. 119–124, Jan. 2003.
15. L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE/ACM Trans. Automat. Control*, vol. 37, pp. 1936–1948, Dec. 1992.
16. R. E. Tarjan, *Data Structures and Network Algorithms*. Murray Hills, NJ: Bell Labs, 1983.
17. T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-speed switch scheduling for local area networks," *ACM Trans. Comput. Syst.*, vol. 11, pp. 319–352, Nov. 1993.
18. N. McKeown, *Scheduling Algorithms for Input-Queued Switches*. PhD thesis, University of California at Berkeley, 1995.
19. N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 7, pp. 188–201, Apr. 1999.
20. H. Chao and J. Park, "Centralized contention resolution schemes for a large-capacity optical ATM switch," in *Proc. IEEE ATM Workshop*, (Fairfax, VA), pp. 11–16, May 1998.
21. D. Serpanos and P. Antoniadis, "Firm: A class of distributed scheduling algorithms for high-speed ATM switches with multiple input queues," in *Proc. IEEE INFOCOM 2000*, vol. 2, (Tel Aviv, Israel), pp. 548–555, Mar. 2000.
22. Y. Li, S. Panwar, and H. Chao, "On the performance of a dual round-robin switch," in *Proc. IEEE INFOCOM 2001*, vol. 3, (Anchorage, AK), pp. 1688–1697, Apr. 2001.
23. A. Mekittikul and N. McKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," in *Proc. IEEE INFOCOM '98*, (San Francisco, CA), pp. 792–799, Apr. 1998.
24. M. Ajmone Marsan, A. Bianco, and E. Leonardi, "RPA: A simple, efficient, and flexible policy for input buffered ATM switches," *IEEE Commun. Lett.*, vol. 1, pp. 83–86, May 1997.
25. A. Smiljanić, R. Fan, and G. Ramamurthy, "RRGS-round-robin greedy scheduling for electronic/optical terabit switches," in *Proc. IEEE GLOBECOM 1999*, (Rio de Janeiro, Brazil), pp. 584–555, Dec. 1999.
26. Y. Tamir and H.-C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 13–27, Jan. 1993.
27. W. T. Chen, C. F. Huang, C. Y. L., and W. Y. Hwang, "An efficient cell-scheduling algorithm for multicast atm switching systems," *IEEE/ACM Trans. Networking*, vol. 8, pp. 517–525, Aug. 2000.
28. P. Gupta and N. McKeown, "Designing and implementing a fast crossbar scheduler," *IEEE Micro*, vol. 19, pp. 20–28, Jan./Feb. 1999.
29. A. Bianco, P. Giaccone, E. Leonardi, F. Neri, and C. Piglion, "On the number of input queues required to support multicast traffic in input queued switches," in *Proc. IEEE Workshop on High-Performance Switching and Routing HPSR 2003*, (Torino, Italy), pp. 49–54, June 24–27, 2003.

30. A. Bianco, P. Giaccone, C. Piglione, S. Sessa “Practical Algorithms for Multicast Support in Input Queued Switches”, in *Proc. IEEE Workshop on High-Performance Switching and Routing HPSR 2006*, (Poznan, Poland), June 2006
31. A. Scicchitano, A. Bianco, P. Giaccone, E. Leonardi, E. Schiattarella, “Distributed scheduling in input queued switches”, in *Proc. IEEE International Conference on Communications (ICC 2007)*, (Glasgow, Scotland), June 2007
32. M. Ajmone Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, “Multicast traffic in input-queued switches: Optimal scheduling and maximum throughput,” *IEEE/ACM Trans. Networking*, vol. 11, pp. 465–477, June 2003.
33. J. Hayes, R. Breault, and M. Mehmet-Ali, “Performance analysis of a multicast switch,” *IEEE/ACM Trans. Commun.*, vol. 39, pp. 581–587, Apr. 1991.
34. J. Y. Hui and T. Renner, “Queueing analysis for multicast packet switching,” *IEEE Trans. Commun.*, vol. 42, pp. 723–731, feb/mar/apr 1994.
35. Z. Liu and R. Righter, “Scheduling multicast input-queued switches,” *J. Scheduling*, vol. 2, pp. 99–114, 1999.
36. B. Prabhakar, N. McKeown, and R. Ahuja, “Multicast scheduling for input-queued switches,” *IEEE J. Sel. Areas Commun.*, vol. 15, pp. 855–866, June 1997.
37. G. F. Pfister, “An introduction to the InfiniBand architecture,” in *High Performance Mass Storage and Parallel I/O: Technologies and Applications* (H. Jin, T. Cortes, and R. Buyya, eds.), ch. 42, pp. 617–632, New York, NY: IEEE Computer Society Press and Wiley, 2001.
38. C. Minkenberg, “Performance of i-SLIP scheduling with large round-trip latency,” in *Proc. IEEE Workshop on High-Performance Switching and Routing HPSR 2003*, (Torino, Italy), pp. 49–54, June 24–27, 2003.
39. C. Minkenberg, F. Abel, and E. Schiattarella, “Distributed crossbar schedulers,” in *To appear in Proc. IEEE Workshop on High-Performance Switching and Routing (HPSR 2006)*, (Poland), 2006.
40. M. Andrews, S. Khanna, and K. Kumaran, “Integrated scheduling of unicast and multicast traffic in an input-queued switch,” in *Proc. IEEE INFOCOM 1999*, vol. 3, pp. 1144–1151, Mar. 1999.
41. M. Song and W. Zhu, “Integrated queueing and scheduling for unicast and multicast traffic in input-queued packet switches,” in *Proc. 2nd IASTED International Conference on Communication and Computer Networks*, (M.I.T., Cambridge, MA), Nov. 2004.
42. A. Smiljanić, “Scheduling of multicast traffic in high-capacity packet switches,” *IEEE Communications Magazine*, pp. 72–77, Nov. 2002.
43. C. Minkenberg, E. Schiattarella, “Fair Integrated Scheduling of Unicast and Multicast Traffic in an Input-Queued Switch,” in *To appear in Proc. IEEE Workshop on High-Performance Switching and Routing (HPSR 2006)* (Poland), 2006.
44. N. McKeown and B. Prabhakar, “Scheduling multicast cells in an input-queued switch,” in *Proc. IEEE INFOCOM 1996*, vol. 1, pp. 271–278, Mar. 1996.
45. T. Clark, *Designing Storage Area Networks*. Addison Wesley, 2 ed., 2003.
46. J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, “Internet Small Computer Systems Interface (iSCSI).” RFC 3720 (Proposed Standard), Apr. 2004.

47. A. Bianco, P. Giaccone, E. M. Giraudo, F. Neri, and E. Schiattarella, "Multicast support for storage area network switches," in *IEEE GLOBECOM 2006*, San Francisco,(USA),2006.
48. I. Iliadis, W.E. Denzel, "Analysis of Packet Switches with Input and Output Queueing", *IEEE Transactions on Communications* , vol. 41, no. 5, pp. 731-740, May. 1993.
49. A. Bianco, A. Scicchitano, "Multicast multi-chip schedulers in input queued switches," in *To appear in Proc. IEEE Workshop on Qos in Multiservice IP Networks (QoS-IP 2008)*(Italy), 2008.