Multicast Traffic Scheduling Based On High-Speed Crossbar Switches

Lotfi Mhamdi, Stamatis Vassiliadis
Computer Engineering Lab, Delft University of Technology,
P.O. Box 5031, 2600 GA Delft,The Netherlands
Phone:+31(0)15 27 89656 E-mail: lotfi@ce.et.tudelft.nl

Abstract— **The tremendous growth of the Internet coupled with newly emerging applications has created a vital need for multicast traffic support by backbone routers and ATM switches. In this paper, we first introduce the multicast traffic scheduling problem. We focus our study on the multicast traffic scheduling in crossbar based input queued (IQ) switches. Due to the centralized scheduling complexity in IQ switches, growing interest is given to the *buffered* crossbar-based switching architecture, where a limited small amount of memory is embedded in each crosspoint of the crossbar fabric. In this paper, we show that a buffered crossbar switch can efficiently support multicast traffic and high throughput can be achieved with distributed and simple scheduling algorithms. Furthermore, we show that the presence of internal buffers is of key importance in optimizing the scheduling and the cost of the switch. Simulation results showed that the buffered crossbar based distributed scheduling algorithms achieve high performance under a wide range of realistic traffic patterns.**

Keywords— **Switching, Scheduling, Mulitcast, Performance, Hardware requirement**

I. Introduction

The explosive growth of the Internet in number of users and service variety is parallel to the growth in transmission links capacity due the advances in fiber optic bandwidth that has created huge supply of wide-area network bandwidth. As a result, switches/routers are becoming the bottleneck of the network. Traditionally, network nodes (IP routers, ATM switches, Ethernet switches) were designed for point-to-point communication (unicast). However, the variety of services on the Internet nowadays has resulted in the emergence of new applications such as teleconferencing, distance learning, IPTV etc. These new applications have lead to a high demand for high-speed switches/routers capable of dealing with point-to-multipoint communication (multicast). Numerous proposals for identifying suitable architectures for efficient multicast support have been investigated and implemented. These architectures can be classified based on various attributes such as queuing schemes, scheduling algorithms, and/or switch fabric topology.

The crossbar-based architecture [14] is widely considered the most suitable switching architecture due to its low cost, scalability and more importantly its *intrinsic multicast capabilities* [8]. Alongside the switching fabric architecture and the traffic supported, the queuing structure of a router is equally important. The Input Queued (IQ) switching architecture is the mostly used because of its low requirement in terms of internal speed up. When first-in-first-out (FIFO) queueing discipline is used at the input queues, the throughput of an IQ is limited to 58.6% due to the Head-of-Line (HoL) blocking problem [5]. The HoL blocking can be completely eliminated by adopting virtual output queuing (VOQ) at each input of the switch [12] which scales the achievable throughput of the switch to 100%. The VOQ structure requires, however, a scheduling algorithm that manages the departure of cells from the input ports. As a result, the switching performance essentially depends on its scheduling algorithm.

Unlike unicast traffic, where a packet (cell) at an input port is destined to only one output port, a multicast cell queued at an input port can have 2 or more destination output ports known as its fanaout set. While different architectures have been proposed for multicast traffic handling [2] which are based on copy networks, in this paper we consider the crossbar-based switching architecture due to its architectural intrinsic multicast capabilities. There has been a great deal of research work on multicast scheduling in the literature. Most of them are based on a multicast FIFO queue architecture [14]. However, because of a similar HoL blocking problem as for the unicast traffic, the performance is low. Avoiding the HoL problem in this case would require a FIFO queue for every fanout set per input. This implies maintaining up to $2^N - 1$ separate FIFO queues per input, where N is the number of ports of the switch [7]. This is clearly impractical for even small sized switches. As a compromise, researchers have proposed to use a small number of queues, k ($1 \leq k \ll 2^N - 1$) per input [3].

This article focuses on the multicast scheduling problem in crossbar-based IQ switches. The scheduling algorithms for this architecture are centralized and have high

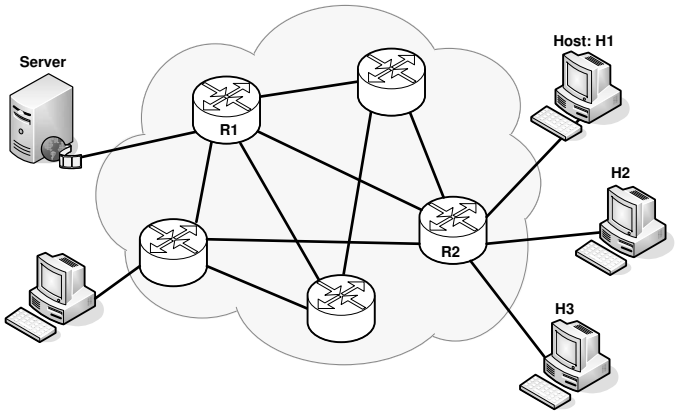

Fig. 1. Multicast Traffic Support in Core Routers

computational complexity. We, therefore, study the multicast problem in a slight alternative to the crossbar switch, where small buffers are embedded within the crossbar fabric chip. This architecture is known as the Combined Input and Crossbar Queued (CICQ) switch. The existence of internal buffers avoids the need for centralized scheduler and relies on simple and distributed scheduler over the input and output ports. We show the superiority of the CICQ architecture to its bufferless predecessor and its high capability to support multicast traffic flows.

The remainder of this article is organized as follows: Section II presents background knowledge and related work and introduces the multicast problem. We present different multicast switching architectures, with a focus on the crossbar-based fabric switches. Section III surveys the existing scheduling algorithms proposed to handle multicast traffic flows. Section IV presents and analyzes the performance of the bufferless as well as the buffered crossbar switching architectures under multicast traffic arrivals. Finally, Section V concludes the paper.

II. The Multicasting Problem

Multicast traffic handling, in its simplest form, is the capability of a router to transfer a cell to multiple destination output ports with the minimum cost in terms of data processing and time. This is important because of the growing proportion of multicast traffic on the Internet (audio, video, IPTV, etc.). If we consider the example in Figure 1, and assume that the three hosts connected to router R2 are receiving the same media content from the server. If the Server sends the same message to hosts, H1, H2, and H3, it either sends the same message three times (one per destination) or it can send the message only once over routers R1 and R2. Once reaching R2, the message gets split into three copies, one copy per destination host. Obviously, the latter case is a better choice as it optimizes the network resources and the time taken for the hosts to receive the data. In order to achieve this, routers R1 and R2 must be designed to support multicast traffic.

The number of destination output ports of a multicast cell is known as its fanout set. If we consider an an $N \times M$ router with multicast capabilities, a multicast cell arriving at any of the N input ports can have any set of destinations between 2 and M. In order to avoid the HoL problem, the router must maintain up to $2^M - 1$ separate FIFO queues per input in order to cover all possible fanout set configurations. This architecture is known as the multicast VOQ (MC-VOQ) [7]. Because of the huge number of queues maintained at each input and the extensive amount of information exchange in order to schedule the traffic, this architecture is considered impractical. Instead, researchers have used just one FIFO queue per input. While using just one queue per input is practical, it has poor performance due to the HoL problem. Another solution was to maintain a small number, k, of queues per input for multicast traffic. This was a good compromise to achieve good performance while maintaining affordable hardware requirements. Because k is much smaller than $2^M - 1$, cells with different fanout sets will have to be queued in the same input queue. This mapping is known as the multicast cell placement policy.

A. The Multicast FIFO Architecture

If we consider that router R2 (in Figure 1) uses just one FIFO queue per input for multicast traffic, its architecture can be described as depicted in Figure 2. By considering that the crossbar fabric operates at the same speed as the external lines, at each time slot[1] every input can send at most one cell and every output can receive at most one cell. Because of the intrinsic multicast capabilities of the crossbar fabric, a cell can be sent to all its destinations at the cost of one by simply closing those crosspoints corresponding to its output ports subject to their availability.

Subject to output availability and the scheduling algorithm used a cell may not reach all its destinations, indicated by its fanout set, during one time slot. There are two known service disciplines used to deal with such situation [14]. The first is known as *no fanout splitting* and the latter is known as *fanout splitting*. When no fanout splitting discipline is used, a cell must traverse the crossbar fabric only once. Meaning that a cell gets switched to its output destination ports if and only if all its destination outputs are available at the same time. If one, or more, of

[1] A time slot is defined as the time between two cell consecutive arrivals/departures to/from an input/output port of the router

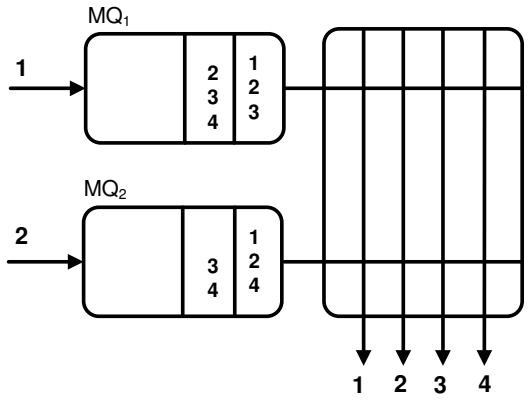

Fig. 2. a 2x4 Multicast Crossbar Switch

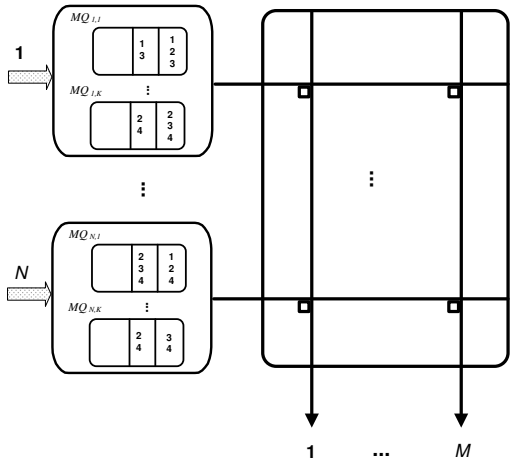

Fig. 3. An $N \times M$ Multicast k FIFO Queues Buffered Crossbar Switch

the output destinations is/are busy, the cell loses contention and all of its copies remain in the input port. If we consider no fanout splitting discipline in Figure 2, then either of the two HoL cells of queues MQ_1 and MQ_2 will be switched out but *not both*. The reason is because both cells have output ports 1 and 2 in their fanout sets and knowing that an output port can receive at most one cell and the no fanout splitting discipline does not allow partial cell switching resulting in only one cell of the two being eligible for transfer. The no fanout splitting discipline is easy to implement, however it results in low throughput because it is not work conserving[2]. This can be seen from the example above as either output 3 or output 4 will receive a cell but not both depending on which MQ has been selected.

When, however, fanout splitting discipline is used, a cell can be *partially* sent to its destination output ports over many time slots. Copies of the cell that are not switched, due to output contention, during one time slot continue competing for transfer during the following time slot(s). The flexibility of allowing partial cell transfer comes at a little increase in implementation complexity, however it provides higher throughput because it is work conserving [4]. In this paper, we consider fanout splitting. Consider the example of Figure 2 again and assuming a fanout splitting discipline is used, then both the HoL cells of MQ_1 and MQ_2 can send copies to a subset of their output ports. Output 3 and 4 are receiving one cell each and therefore both copies destined to them, in the input queues, are transferred with no contention. Additionally, both HoL cells of MQ_1 and MQ_2 have cells destined to outputs 1 and 2. However, we know that each output can receive at most one cell at a time. Therefore, at the end of the time slot, we will have remaining cells for output ports 1 and 2. These remaining cells are referred to as the *residue*.

[2]A work conserving policy ensures that an output port is never idle so long as there are cells destined to it in the input ports

Depending on the policy used, the residue can either be *concentrated* on the input ports or it can be *distributed* over the input ports. As defined in [14], the residue is the number of cells left at the HoL of the input queues after losing contention for the output ports at the end of each time slot. In the example of Figure 2, the residue is $\{1, 2\}$. A concentrating policy is one that leaves the residue on the minimum number of input ports. If we consider a concentrating policy in Figure 2, the residue with be left (concentrated) on either MQ_1 or on MQ_2 but not on both. On the other hand, a distributing policy is one that leaves the residue on the maximum number of input ports. Using a distributing policy in Figure 2 would result in the residue being distributed over MQ_1 and MQ_2 but not on one queue only.

B. The Multicast k FIFO Queues Architecture

Due to the impracticality of the MC-VOQ switching architecture [7] and to the low performance of the multicast FIFO architecture, a good compromise is to use the multicast k FIFO queues architecture. It is a queueing architecture with a small number of input multicast FIFO queues per input ($1 \leq k \ll 2^M - 1$). This queueing architecture has been studied in the context of bufferless crossbar switches [1] [6]. Figure 3 depicts the multicast k FIFO architecture for an $N \times M$ buffered crossbar switch, a crossbar switch where small buffers are added inside the fabric chip [10]. Because the number of multicast queues maintained at each input is much smaller than the cardinality of the fanout set, a cell placement strategy is required in order to enqueue each incoming cell into its corresponding MQ by following certain criteria.

The cell placement has a significant impact on the switch performance. Previous work [3] has defined the criteria for designing a good cell placement policy: *i)* HoL

cells should contain diverse fanout sets that can span a large part of the set of all outputs for which the input holds packets. This ensures more scheduling opportunities and work conservation. *ii)* Cells with the same or similar fanout sets should be placed in the same multicast queue. This would reduce the HoL problem and avoid the out of sequence delivery problem. There have been many cell placement schemes, such as majority [3] and minimum distance queue (MDQ) [1]. While these schemes have met most or all of the above mentioned criteria, their major disadvantage lies in their hardware implementation. A more recent and practical scheme, named the *Modulo*, has been proposed by [11]. If a cell with fanout number f arrives at input i, the Modulo scheme assigns the cell to the multicast queue: $MQ_{i,j}$ where $\{j \mid j = f \ mod(k)\}$. Unless otherwise stated, in this paper, we use the Modulo scheme.

III. Scheduling Multicast Traffic

Similar to unicast traffic, the multicast traffic scheduling problem has drawn considerable attention and many scheduling algorithms have been proposed. This section surveys some of these algorithms based on the input queueing architecture for which they have been designed.

A. Algorithms For The Multicast FIFO Architecture

Several algorithms have been proposed for this architecture, mostly designed for the bufferless crossbar fabric switches.

- *The Concentrate Algorithm:* As the name indicates, the concentrate algorithm [14] always concentrates the residue onto as few inputs as possible. The purpose of this algorithm is to provide a basis for evaluating the performances of other algorithms, since it achieves high throughput for the FIFO queue structure. However, this algorithm doesn't meet the fairness requirement due to the starvation problem it creates. The Concentrate algorithm is not considered a practical algorithm. It requires up to M iterations per cell time to complete, which makes it difficult to implement at high speed.
- *The mRRM Algorithm:* The Multicast Round-Robin Matching (mRRM) was proposed by [13]. A single round-robin pointer is collectively maintained by all of the outputs. Each output selects the next input that requests it at, or after, the pointer. At the end of the packet time, the pointer is moved to one position beyond the first input that is served. Designed to be simple to implement in hardware, mRRM tends to concentrate the selection onto a small number of inputs, yet maintains fairness.
- *The TATRA Algorithm:* The general multicast scheduling problem can be mapped onto a variation of the popular block-packing game Tetris. TATRA is based on the Tetris model and was first introduced in [13]. TATRA has the properties of guaranteeing at least one input packet is discharged each packet time, and also residue concentration. Designed to approximate the concentrate algorithm with less complexity, unfortunately TATRA is a complex algorithm since it cannot be parallelized. Moreover, TATRA treats all inputs uniformly which is of no value when the inputs are non-uniformly loaded or when some inputs request a higher priority.
- *The MXRR Algorithm:* The Multicast crosspoint Round Robin (MXRR) algorithm is the first, and one the few algorithms, proposed for the buffered crossbar (CICQ) switching architecture [9]. Due to the distributed and lower computational complexity of the CICQ based scheduling algorithms, MXRR has been shown to exhibit higher performance than all other multicast FIFO queue based algorithms while keeping simple hardware requirements.

B. Algorithms For The Multicast k FIFO Queues Architecture

The low performance and high complexity of the multicast FIFO architecture have stressed the need for the multicast k FIFO queues architecture and many scheduling algorithms have been proposed. These algorithms have been designed for the bufferless as well as for the buffered crossbar switching architectures.

- *Bufferless Crossbar Based Algorithms*: Algorithms for this architecture include the random scheduler (RS), the Greedy Scheduler (GS) and the Greedy Min-Split Scheduler (GMSS) [1]. The first algorithm either makes decisions randomly among the input and and output ports. In addition to its costly hardware cost, this scheme has poor performance as it leaves idle outputs due to the contention effect. The second algorithm tries to overcome this and assigns weights to the queues such as queue length and then makes its selection based on weight ordering. The third algorithm is also weighted algorithm and tries to combine the advantages of the previous. As it requires sorting, however, its implementation can prove difficult and prevents it from running at high rates.
- *Buffered Crossbar Based Algorithms:* A group of scheduling algorithms have recently been proposed for the multicast k FIFO queues architecture designed for the CICQ switching architecture [15]. These algorithms were proposed along with a class of cell placement schemes. The input arbitration was based on some policies such as giving preference to HoL cells that would result in the minimum reside left. Another input scheduling was based on selecting the cell with the maximum number of reachable destinations first. A third policy is to give preference to cells with the maximum service ratio, defined as the the

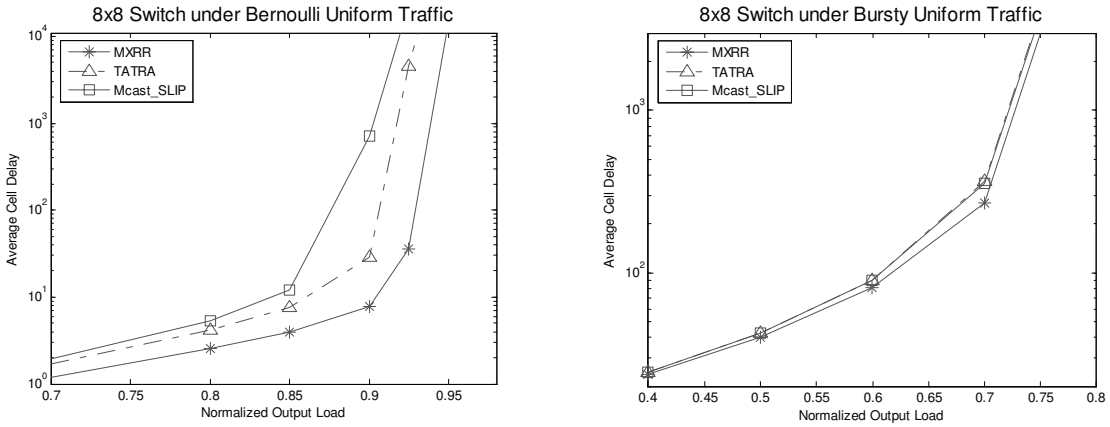

Fig. 4. Average Cell Delay of 8×8 Multicast FIFO Crossbar Switch

number of reachable destination outputs divided by the fanout number of a cell. The output arbitration was based on round robin and Longest Queue First (LQF).

IV. Performance Study

This section studies and analyzes the performance of different queueing and switching architectures. In particular this study is aimed at comparing the bufferless crossbar switching architecture to the buffered CICQ switching architecture when employing the multicast k FIFO queues. The performance results presented in this section are done for two different switch sizes (8×8 and 16×16 respectively). We carried out the performance under two input traffic patterns: Bernoulli uniform and Bursty uniform. We compared the TATRA algorithm and Multicast ISLIP [12] bufferless crossbar switch and the MXRR algorithm for the buffered crossbar architecture. This study is targeting the multicast FIFO queueing architecture. We also compared the performance of the multicast ISLIP algorithm with that of MXRR for the multicast k FIFO queues architecture.

Figure 4 depicts the average delay performance of the TATRA and multicast ISLIP for the an 8×8 bufferless crossbar switch compared to the MXRR algorithm running an 8×8 buffered crossbar switch. As the figure shows, MXRR has better delay performance than the other two. This result holds for uniform Bernoulli arrival (left graph on Figure 4) as well as for bursty uniform arrivals with a burst length of 16 cells (right graph on Figure 4).

In order to better analyze the behavior of each algorithm, we tested the algorithms under the same settings as above but with a larger sized switch. This is important because, as the switch sizes increases, the fanout sets of the cells increases making it harder for the algorithm to schedule the traffic due to increased contention. For this end, Figure 5 depicts the average cell delay of each of the

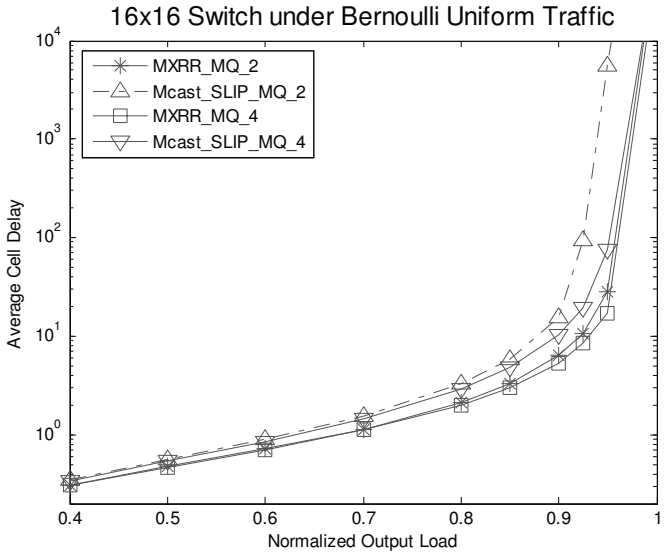

Fig. 6. Average Cell Delay of 16×16 Multicast k FIFO Crossbar Switch With Different Numbers of Input Queues, k = 2, 4

three algorithms for a 16×16 switch. Again, the MXRR algorithm keeps the shortest cell delay amongst the three algorithms both under Bernoulli uniform and bursty uniform arrivals.

In the previous experiments, we tested three algorithm for the multicast FIFO queueing architecture. In the following simulation, we compared the delay performance of the mcast_SLIP bufferless algorithm with the MXRR algorithm because of their similarities, as non weighted algorithms. Figure 6 depicts the average delay performance for each of mcast_SLIP and MXRR for the multicast k FIFO queues architecture. We used 2 and 4 MQs per input for a 16×16 switch under Bernoulli unform traffic arrivals. We can see from Figure 6 that MXRR outperforms the bufferless mcast_SLIP algorithm irrespective of the number MQs used per input. MXRR still achieves

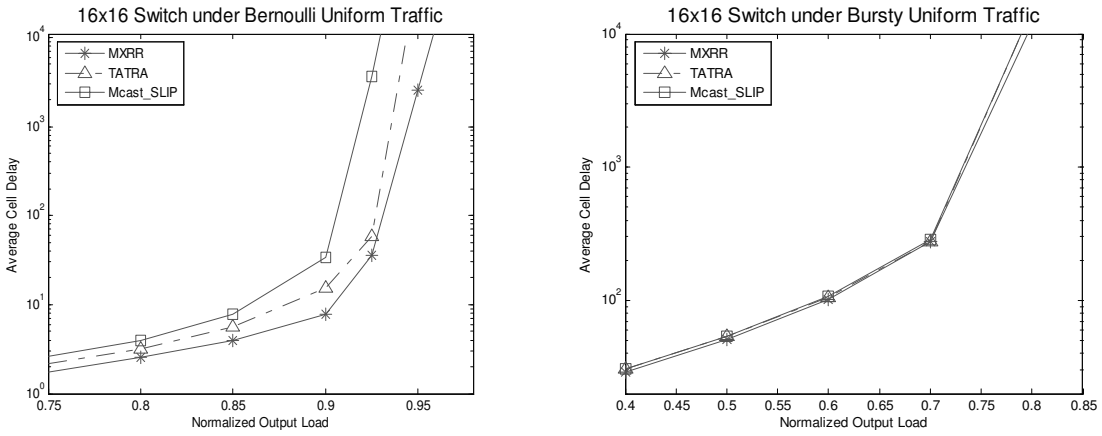

Fig. 5. Average Cell Delay of 16×16 Multicast FIFO Crossbar Switch

higher performance while using half the number of MQs that mcast_SLIP does.

V. Conclusion

Building high speed switches and Internet routers with multicast support is becoming important due to the growing proportion of multicast flows. This paper studies the multicast traffic problem and surveys existing multicast switching architecture as well as their scheduling algorithms. The crossbar based switching architecture is known to have good performance because of its intrinsic multicast capabilities. A variation of the crossbar-based architecture, the buffered crossbar switching (CICQ) architecture is arguably the best candidate for handling multicast traffic. The CICQ is shown to exhibit good delay performance in the presence of multicast flows. Additionally, the distributed scheduling algorithms for the CICQ are shown to be readily implementable allowing them to run at high speed.

References

[1] A. Bianco, P. Giaccone, E. Leonardi, F. Neri, and C. Piglione. "On Tthe Number Of Input Queues To Efficiently Support Multicast Traffic In Input Queued Switches". *IEEE HPSR*, pages 111–116, June 2003.

[2] M. Guo and R. Chang. "Multicast ATM switches: survey and performance evaluation". *Computer Communication Review*, 28(02):98–131, Apr. 1998.

[3] S. Gupta and A. Aziz. "multicast scheduling for switches with multiple input-queues". *Proc. of Hot Interconnects*, pages 28–33, 2002.

[4] J. Y. Hui and T. Renner. "Queuing Analysis for Multicast Packet Switching". *IEEE Transactions on Communications*, 42(02):723–731, Feb. 1994.

[5] M. Karol, M. Hluchyj, and S. Morgan. "Input Versus Output Queuing on a Space-Division Packet Switch". *IEEE Trans. on Commun.*, 35(09):1337–1356, Dec. 1987.

[6] M. Song and W. Zhu. "throughput analysis for multicast switches with multiple input queues". *IEEE Communications Letters*, 08:479–481, July 2004.

[7] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri. "optimal multicast scheduling in input-queued switches". *IEEE ICC*, pages 2021–2027, June 2001.

[8] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri. "multicast traffic in input-queued switches: Optimal scheduling and maximum throughput". *IEEE/ACM Transactions on Networking*, 03(11):465–477, June 2003.

[9] L. Mhamdi and M. Hamdi. Scheduling multicast traffic in internally buffered crossbar switches. *IEEE ICC*, 02:1103–1107, June 2004.

[10] L. Mhamdi, M. Hamdi, C. Kachris, S. Wong, and S. Vassiliadis. "High-Performance Switching Based On Buffered Crossbar Fabrics". *Computer Networks Journal*, 50(13):2271–2285, Sep. 2006.

[11] L. Mhamdi and S. Vassiliadis. Integrating Uni- and Multicast Scheduling in Buffered Crossbar Switches". *IEEE HPSR*, pages 99–1046, June 2006.

[12] M. N. *"Scheduling Algorithms for Input-Queued Cell Switches"*. PhD thesis, University of California at Berkeley, May 1995.

[13] B. Prabhakar and N. McKeown. Designing a Multicast Switch Scheduler". *'Proceedings of the 33rd Annual Allerton Conference on Communication, Control, and Computing*, pages 984–993, Oct. 1995.

[14] B. Prabhakar, N. McKeown, and R. Ahuja. "Multicast Scheduling for Input-Queued Switches". *IEEE Journal on Selected Areas in Communoications*, 15(15):885–866, June 1997.

[15] S. Sun, S. He, Y. Zheng, and W. Gao. Multicast scheduling in buffered crossbar switches with multiple input queues. *IEEE HPSR*, pages 73–77, May 2005.