Efficient Multicast Support in High-Speed Packet Switches

Lotfi Mhamdi, Georgi Gaydadjiev and Stamatis Vassiliadis
Computer Engineering Laboratory
Delft University of Technology, The Netherlands
Email: {L.Mhamdi, G.N.Gaydadjiev, S.Vassiliadis}@ewi.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. Considerable research work has been done on Input Queued (IQ) switches to handle multicast traffic flows. Unfortunately, all previously proposed solutions were of no practical value because they either lack performance or were simply too complex to implement. Internally Buffered Crossbar (IBC) switches, where a limited small amount of memory is added in each crosspoint of the crossbar fabric, on the other hand, have been considered as a robust alternative to buffer-less crossbar switches to improve the switching performance. However, very little has been done on multicasting in IBC switches. In this paper, we fill this gap and study the multicasting problem in IBC switches. In particular, we propose a novel IBC based multicast architecture along with a simple scheduling scheme named Multicast cross-point Round Robin (MXRR). Our scheme was shown to handle multicast traffic more efficiently and far better than all previous schemes for both the multicast FIFO architecture as well as the multicast k FIFO queues architecture. Yet, MXRR is both practical and achieves high performance.**

***Index Terms*— Switching, Scheduling, Mulitcast, Performance, Hardware requirements**

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 to 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, new applications such as teleconferencing, distance learning, IPTV and more are emerging. These new applications have lead to higher 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 [1] [2] [3]. These architectures can be classified based on various attributes such as queuing schemes, scheduling algorithms, and/or switch fabric topology.

The crossbar-based architecture [4] is widely considered the most suitable switching architecture due to its low cost, scalability and more importantly its *intrinsic multicast capabilities* [3]. 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 [6] 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 fanout set. While different architectures have been proposed for multicast traffic handling [7] 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 [4]. 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 output ports of the switch [8]. 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 [9] [10].

This article focuses on the multicast scheduling problem in crossbar-based IQ switches. In particular, we propose a novel architecture to handle multicast traffic. Our architecture is based on multicast FIFO queues at the ingress ports with an internally buffered crossbar (IBC)

This paper is based on "Scheduling Multicast Traffic in Internally Buffered Crossbar Switches" by L. Mhamdi, and M. Hamdi, which appeared in IEEE International Conference on Communications, (ICC 04), Paris, France, June 2004. © 2004 IEEE.

This work was supported by the European Commission in the context of the SARC integrated project #27648 (FP6).

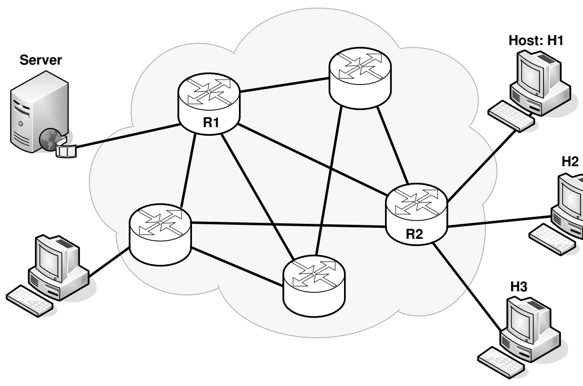

Figure 1. Multicast Traffic Support in Core Routers

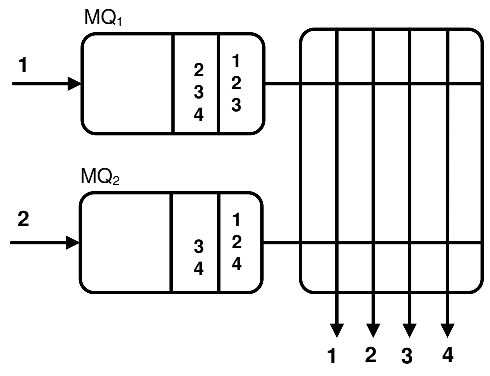

Figure 2. a 2x4 Multicast Crossbar Switch

fabric. Throughout the rest of this article, we will refer to this architecture as *Multicast Internally Buffered Crossbar* (MIBC) switch. The existence of internal buffers avoids the need for centralized scheduler and relies on simpler, distributed schedulers over the input and output ports. We propose a scheduling algorithm named the *Multicast cross-points Round Robin* (MXRR) algorithm. We further extend the MXRR algorithm and propose the MXRR_k scheduling algorithm that is suitable for the Multicast k FIFO queues architecture. We show the superiority of the MIBC 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 and survey the existing scheduling algorithms. Section III introduces the MIBC architecture along with our proposed MXRR scheduling scheme. We also present the MXRR_k algorithm for the multicast k FIFO queue architecture. Section IV presents a simulation study and compares the MXRR algorithm to some state-of-the art schemes. Finally, section V gives some concluding remarks.

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 to cover all possible fanout set configurations. This architecture is known as the multicast VOQ (MC-VOQ) [8]. 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. This approach is very practical, however it has poor performance due to the HoL problem. Another solution is to maintain a small number, k, of queues per input for multicast traffic. This was a good compromise to achieve high performance while maintaining affordable hardware requirements. Cells with different fanout sets will have to be placed in the same input queue because k is much smaller than $2^M - 1$. 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 rate 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 (multiple copies) can be sent to all its destinations at the cost of one by simply closing those crosspoints corresponding to its output ports provided they are available.

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 [4]. The first is known as *no fanout splitting*

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

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 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 [11]. 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*.

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 [4], 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.

[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

B. 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 [4] 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 [12]. 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 [12]. 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.

C. The Multicast k FIFO Queues Architecture

Due to the impracticality of the MC-VOQ switching architecture [8] 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 [13] [14]. 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 [15]. 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 multicast queue (MQ) by following certain criteria.

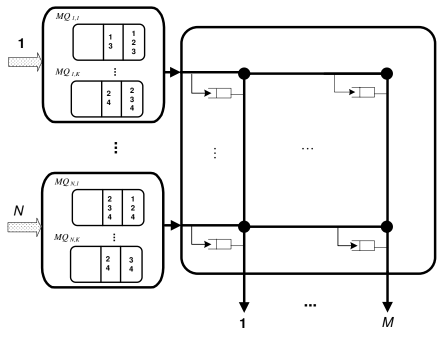

Figure 3. An $N \times M$ Multicast k FIFO Queues Buffered Crossbar Switch

The cell placement has a significant impact on the switch performance. Previous work [9] 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 [9] and minimum distance queue (MDQ) [13]. 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 [16]. Unless otherwise stated, in the remainder of this article, we use the *modulo* scheme.

D. 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) [13]. 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 prevent 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 IBC switching architecture [17]. 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 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).

As a summary, we argue that each of the above presented schemes tries to address some issues but fails to meet other vital requirements. So far, none of these algorithms proved simultaneously efficient in terms of high throughput, practical in terms of implementation complexity or fair with respect to the input FIFO queues. In the following section, we propose a new architecture along with a scheduling scheme that overcomes the limitations of previous proposals.

III. THE MULTICAST INTERNALLY BUFFERED CROSSBAR SWITCH ARCHITECTURE (MIBC)

Our choice of the multicast internally buffered crossbar switch architecture is motivated by the fact this architecture has key advantages that can serve to ensure that the scheduling algorithm can be simple and efficient at the same time. The presence of internal buffers drastically improves the overall performance of the switch due to the advantages it offers. First, the adoption of internal buffers makes the scheduling totally distributed, hence reducing dramatically the arbitration complexity and making it linear. Second, and most importantly, these internal buffers reduce (or avoid) the output contention. Meaning, they allow the inputs to send cells to an output irrespective of simultaneous cells transfer to the same output. If an output is not ready to receive a cell from an input, the input still can send it to the internal buffer provided that this internal buffer has enough room for that cell.

A. Switch Model

We consider the switch model defined in Figure 4. Fixed size packets, or cells, are considered. Upon their arrival to the switch, variable length packets are segmented into cells for internal processing and re-assembled before they leave the switch. A processing cycle has a fixed length, called a cell or time slot. There are N input cards, each one contains a FIFO multicast queue. The internal fabric consists of NM buffered crosspoints (XP). When an arriving cell, to an input port i, $\forall\ 1 \leq i \leq N$, has its fanout vector containing the output j, $\forall\ 1 \leq j \leq M$, it must go through the crosspoint $XP_{i,j}$, before continuing its journey to the output buffer.

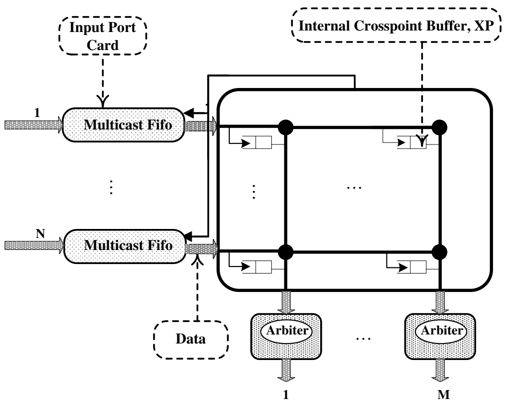

Figure 4. The $N \times M$ multicast Internally Buffered Crossbar (MIBC) Switch Architecture.

As with unicast scheduling, a multicast scheduling cycle consists of the following three steps: input scheduling, output scheduling and delivery notifying. During the input scheduling phase, each input, i, selects, in an independent and parallel way, the HoL cell of its multicast FIFO queue and send it to the internal buffer corresponding to its fanout set. Likewise, each output, j, selects, independently and in parallel, a non empty crosspoint buffer, $XP_{i,j}$, and send its cell to the output queue. Then, the delivery notifying is performed to carry the flow control between the internal buffers and the input queues. For each delivered cell, the flow control mechanism "informs" the corresponding input of the internal buffer status.

B. The Multicast Cross-point Round Robin Algorithm: MXRR

This section introduces the Multicast cross-point Round Robin algorithm, MXRR. The description of each scheduling phase of the MXRR is as follows:

```
Input Scheduling:
  . For each input, $i$, do
      . Send the FIFO HoL multicast cell to the set
        of internal buffers corresponding to its fanout
        vector.
      . If one or more internal buffers are not free, the
        cell stays at the HoL of that input and waits
        for the next input scheduling phase to send to
        its remaining internal buffers.

Output Scheduling:
  . All the output pointers are, artificially, set to the same
    initial position and incremented, each time slot, by
    one $mod\ (N)$.
  . For each output, $j$, do
      . Starting from its pointer's index, select the first
        non empty cross point and send its queued cell
        to the output buffer.
```

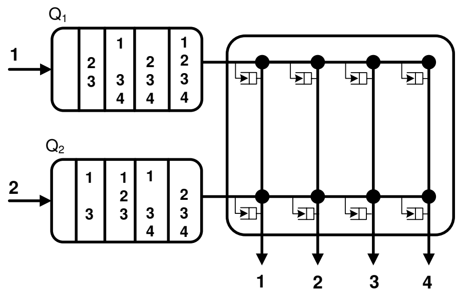

Figure 5. A 2×4 MIBC Switch

The MXRR algorithm exhibits good properties such as low cell latency, fairness and non starvation, simplicity in design and high throughput. To better see these properties, and without loss of generality, consider the following example of a 2×4 MIBC switch in Figure 5. Let us assume that the output pointers are all pointing to input 1 and all the internal buffers are empty. During the input scheduling phase, both HoL cells of Q1 and Q2 will be completely transferred to the internal buffers. During the output scheduling phase, since the output pointers have index 1 each, therefore every output j, will select the internal buffer $XP_{1,j}$, $\forall\ 1 \leq j \leq 4$. This means that, during this time slot, the HoL cell of $Q1$ is completely served. At the beginning of the second time slot, $XP_{2,2}$, $XP_{2,3}$, and $XP_{2,4}$ are occupied, therefore the second cell of $Q2$ (which becomes the HoL cell) cannot send its cell to all the outputs $1, 3, 4$. It only can send to $XP_{2,1}$ which leaves a residue of $3, 4$. $Q1$, however, can send its HoL cell completely to the internal buffers. During the output scheduling phase, since the output pointer's indexes are incremented to 2, therefore each output j, will select the internal buffer $XP_{2,j}$, $\forall\ 1 \leq j \leq 4$. This means that, during this time slot, the HoL cell of $Q2$ is completely served and part of the second cell as well. From this example, we draw the following properties and advantages of the MXRR scheduling scheme:

- Low cell latency: The MXRR scheme guarantees the total service of at least one packet each time due to the output pointers setting (which point to the same internal buffer and advance synchronously). Moreover, the time a packet waits at the HoL is bound by number of input ports, N. The time a packet waits at internal buffer, XP, is also bound by the number of inputs, N. So the delay experienced by every HoL packet inside the switch is no more than $2N$ time slots.
- Fair and starvation free: Since the output pointers move artificially and in a synchronous fashion irrespective of the chosen packet, the starvation problem will never occur. The chance of service for any two cells from two different input ports is exactly the same due to the round robin pointer movement.
- Simple in hardware implementation: Each input does FIFO arbitration. The outputs, on the other hand, work in a totally distributed and parallel manner. No

computation and comparison of weights is needed to make an arbitration decision. Each output arbiter just performs simple round-robin arbitration.

- Enhanced performance: Achieves high throughput and has lower packet latency than all previously proposed buffer less algorithms. We will examine this property in Section IV, which contains the simulation results.

C. The Multicast k FIFOs Algorithm: MXRR_k

While employing the Multicast FIFO architecture is simple and practical, it suffers poor performance due to the HoL problem. In order to completely eliminate the HoL blocking problem, multicast cells having the same fanout sets must be placed in the same *separate* multicast queue (MQ), which requires as many MQs as the multicast VOQ (MC-VOQ) architecture would and this is clearly infeasible for even a small switching system. A good alternative is to use a small number, k, of MQs per input to accommodate the incoming multicast cells. This is a good compromise to achieve good performance while maintaining affordable hardware requirements. In our model (see Figure 3), each input maintains a small number, k, of multicast FIFO queues per input, where $\{k \mid 1 \leq k \ll 2^M - 1\}$. At each input, multicast queues are denoted by $MQ_{i,j}$ where $\{(i,j) \mid 1 \leq i \leq N; 1 \leq j \leq k\}$. An input multicast queue, MQ, is considered eligible (denoted EMQ) if it is not empty and at least one of its destination output ports corresponds to a free XP. 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. In our architecture, we use the *modulo* cell assignment policy [16]. If a cell with fanout number Φ arrives at input i, the *modulo* scheme assigns the cell to the multicast queue: $MQ_{i,j}$ where $\{j \mid j = f\ mod(k)\}$.

Based on the Mulitcast k FIFO queues architecture, the specification of the $MXRR\_k$ is as follows:

```
Input Scheduling:
  . For each input, $i$, do
      . Starting from the pointer's location, select the
        first eligible queue $EMQ_{i,j}$ and send its HoL
        cell copies[3] to the free internal buffers ($XP_{i,j}$).
      . Move the pointer to the location $(j+1)\ (mod\ k)$.
Output Scheduling:
  . All the output pointers are, artificially, set to the same
    initial position and incremented, each time slot, by
    one $mod\ (N)$.
  . For each output, $j$, do
      . Starting from its pointer's index, select the first
        non empty cross point buffer and sends its
        queued cell to the output buffer.
```

[3]Only copies destined to c outputs are sent, where $\{c \mid c \in \{1, ..., M\}\ and\ XP_{i,c}\ is\ not\ full\}$. Other copies will have to compete in later time slots.

The $MXRR\_k$ output scheduling remains the same as that of MXRR because every cell is treated the same independently of whether it is coming from an input FIFO or an input multicast queue, MQ. Additionally, maintaining the same output scheduling keeps the same property of ensuring the *complete* service of at least one cell per time slot. $MXRR\_k$ differs from $MXRR$ in its input scheduling. First, it uses a round robin priority pointer in servicing the input multicast queues, MQ. Second the delay a cell waits in the HoL under $MXRR\_k$ is bound by $\sup(k, N)$ time slots, where N is the number of input ports of the switch. The queuing delay inside the crossbar fabric is the same as that of MXRR (N time slots). So the delay experienced by a HoL cell inside the switch under the $MXRR\_k$ algorithm is bound by $\sup(k, N) + N$ time slots. The round robin mechanism of $MXRR\_k$ allows it be fair and starvation free while kept simple in hardware.

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 MIBC switching architecture. 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 [6] bufferless crossbar switch with the MXRR algorithm for the buffered crossbar architecture. The choice of TATRA was because of its high performance. 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. The choice of the multicast ISLIP is because it is considered as practical.

Figure 6 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 6) as well as for bursty uniform arrivals with a burst length of 16 cells (right graph on Figure 6).

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 7 depicts the average cell delay of each of the 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. MXRR is expected to have a shorter cell delay as the internal buffer size increases. This

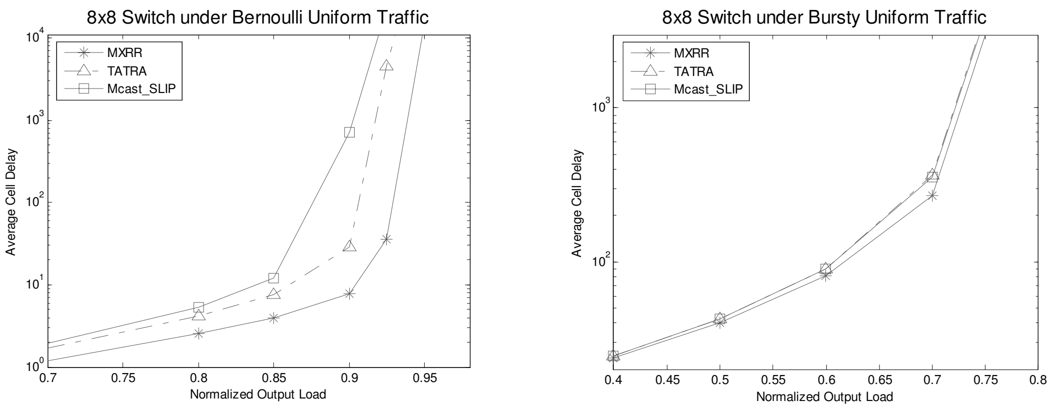

Figure 6. Average Cell Delay of 8×8 Multicast FIFO Crossbar Switch

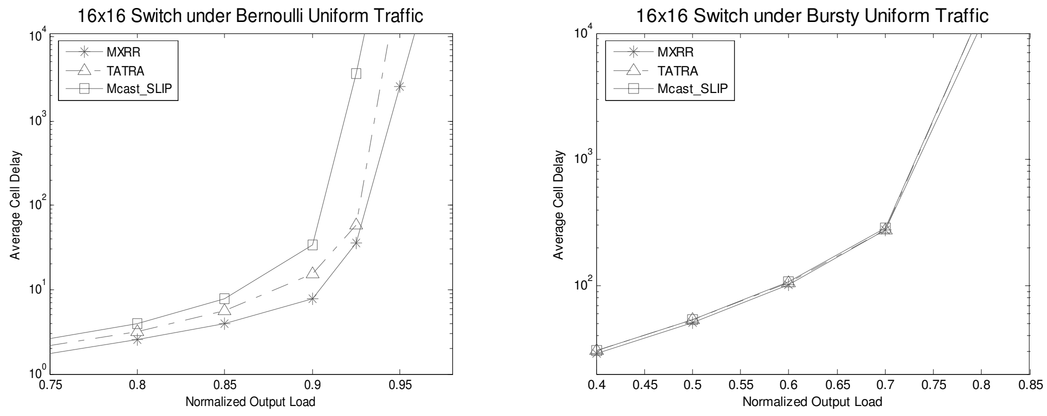

Figure 7. Average Cell Delay of 16×16 Multicast FIFO Crossbar Switch

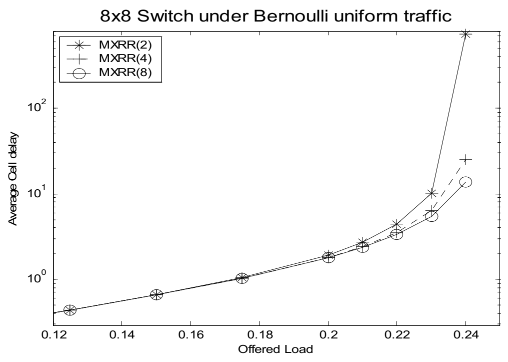

Figure 8. Average Cell Delay Of MXRR With Different Internal Buffer Sizes

is because the scheduler is exposed to more scheduling opportunities and less HoL blockings. Figure 8 illustrates this behavior, where the internal buffer size is set to 2, 4 and 8 cells per crosspoint respectively. Please note that offered load, in Figure 8, refers to input load.

In the previous experiments, we tested the three algorithms 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 9 depicts the average delay performance for each of mcast_SLIP and MXRR_k for the multicast k FIFO queues architecture. We used 2 and 4 MQs per input for a 16×16 switch under Bernoulli unform traffic. We can see from Figure 9 that MXRR outperforms the bufferless mcast_SLIP algorithm irrespective of the number MQs used per input. MXRR_k still achieves higher performance while using half the number of MQs that mcast_SLIP does. The MXRR algorithm, not only outperforms mcast_SLIP in terms of cell cell delay, but most importantly in its simpler hardware requirements, allowing it to run at very high rates.

V. Conclusions

Building high speed switches and Internet routers with multicast support is becoming important due to the growing proportion of multicast traffic on the Internet. This paper studies the multicast traffic problem and surveys existing multicast switching architectures as well as their scheduling algorithms. In addition, we propose a new switching architecture, named MIBC, along with its appropriate scheduling algorithms to efficiently support multicast traffic flows. The MXXR algorithm, as well as its MXRR_k variation, was shown to outperform all state-of-the-art algorithms both in cell delay and throughput. Our proposed algorithm requires simple hardware

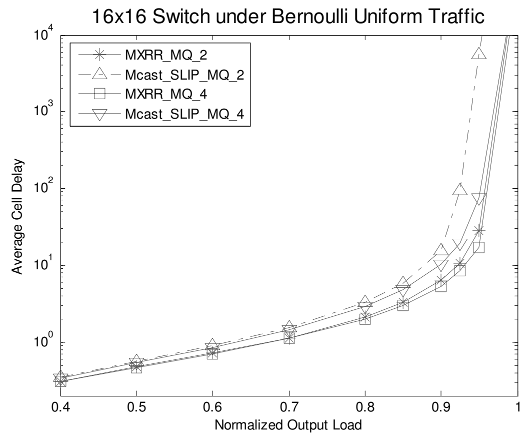

Figure 9. Average Cell Delay of 16×16 Multicast k FIFO Crossbar Switch With Different Numbers of Input Queues, $k = 2, 4$

implementation and is able to run at very high rates. We expect that, with more carefully designed scheduling schemes for the MIBC switching architecture, even higher performance can be achieved.

REFERENCES

[1] H. J. Chao, B.-S. Choe, J.-S. Park, and N. Uzun, "Design and Implementation of Abacus Switch: A Scalable Multicast ATM Switch," *IEEE J. Select. Areas Commun.*, vol. 15, no. 5, pp. 830–843, Sept. 1997.
[2] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, "Scaling Internet Routers Using Optics," *ACM SIGCOMM*, pp. 189–200, Jun. 2003.
[3] 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*, vol. 03, no. 11, pp. 465–477, Jun. 2003.
[4] B. Prabhakar, N. McKeown, and R. Ahuja, "Multicast Scheduling for Input-Queued Switches," *IEEE Journal on Selected Areas in Communoications*, vol. 15, no. 5, pp. 885–866, Jun. 1997.
[5] M. Karol, M. Hluchyj, and S. Morgan, "Input Versus Output Queuing on a Space-Division Packet Switch," *IEEE Trans. on Commun.*, vol. 35, no. 09, pp. 1337–1356, Dec. 1987.
[6] McKeown. N., "Scheduling Algorithms for Input-Queued Cell Switches," Ph.D. dissertation, University of California at Berkeley, May 1995.
[7] M. Guo and R. Chang, "Multicast ATM switches: survey and performance evaluation," *Computer Communication Review*, vol. 28, no. 02, pp. 98–131, Apr. 1998.
[8] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Optimal Multicast Scheduling in Input-Queued Switches," *IEEE ICC*, pp. 2021–2027, Jun. 2001.
[9] S. Gupta and A. Aziz, "Multicast scheduling for switches with multiple input-queues," *Proc. of Hot Interconnects*, pp. 28–33, 2002.
[10] A. Bianco, P. Giaccone, C. Piglione, and S. Sessa, "Practical Algorithms for Multicast Support in Input Queues Switches," *IEEE HPSR*, pp. 187–192, Jun. 2006.
[11] J. Y. Hui and T. Renner, "Queuing Analysis for Multicast Packet Switching," *IEEE Transactions on Communications*, vol. 42, no. 02, pp. 723–731, Feb. 1994.
[12] B. Prabhakar and N. McKeown, "Designing a Multicast Switch Scheduler," *Proceedings of the 33rd Annual Allerton Conference on Communication, Control, and Computing*, pp. 984–993, Oct. 1995.
[13] 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*, pp. 111–116, Jun. 2003.
[14] M. Song and W. Zhu, "Throughput analysis for multicast switches with multiple input queues," *IEEE Communications Letters*, vol. 08, pp. 479–481, Jul. 2004.
[15] L. Mhamdi, M. Hamdi, C. Kachris, S. Wong, and S. Vassiliadis, "High-Performance Switching Based On Buffered Crossbar Fabrics," *Computer Networks Journal*, vol. 50, no. 13, pp. 2271–2285, Sep. 2006.
[16] L. Mhamdi and S. Vassiliadis, "Integrating Uni- and Multicast Scheduling in Buffered Crossbar Switches," *IEEE HPSR*, pp. 99–1046, Jun. 2006.
[17] S. Sun, S. He, Y. Zheng, and W. Gao, "Multicast scheduling in buffered crossbar switches with multiple input queues," *IEEE HPSR*, pp. 73–77, May 2005.

Lotfi Mhamdi received the B.S. degree from South University, Tunisia, in 2000 and the Master of Philosophy (MPhil) Degree in computer science from the Hong Kong University of Science and Technology (HKUST), Hong Kong, in 2002. He is currently at the final stage of his Ph.D. in Computer Engineering at TU Delft, The Netherlands. His research work spans the area of high-speed networks including the design, analysis, scheduling, and management of high-speed switches and Internet routers. He is a member of IEEE.

Georgi Gaydadjiev was born in Plovdiv, Bulgaria, in 1964. He is currently assistant professor at the Computer Engineering Laboratory, Delft University of Technology, The Netherlands. His research and development experience includes 15 years in hardware and software design at System Engineering Ltd. in Pravetz Bulgaria and Pijnenburg Microelectronics and Software B.V. in Vught, the Netherlands. His research interest include: embedded systems design, advanced computer architectures, hardware/software co-design, VLSI design, cryptographic systems and computer systems testing. He is IEEE and ACM member and member of the HiPEAC Steering committee.

Stamatis Vassiliadis was born in Manolates, Samos, Greece 1951. Regrettably, Prof. Vassiliadis deceased in April 2007. He was a chair professor in the Electrical Engineering department of Delft University of Technology (TU Delft), The Netherlands. He had also served in the EE faculties of Cornell University, Ithaca, NY and the State University of New York (S.U.N.Y.), Binghamton, NY. He worked for a decade with IBM where he had been involved in a number of advanced research and development projects. For his work he received numerous awards including 24 publication awards, 15 invention awards and an outstanding innovation award for engeneering/scientific hardware design. His 72 USA patents rank him as the top all time IBM inventor.

Dr. Vassiliadis received an honorable mention Best Paper award at the ACM/IEEE MICRO25 in 1992 and Best Paper awards in the IEEE CAS (1998, 2001), IEEE ICCD (2001), PDCS (2002) and the best poster award in the IEEE NANO (2005). He is an IEEE and ACM fellow and a member of the Royal Dutch Academy of Science.