On the Integration of Unicast and Multicast Cell Scheduling in Buffered Crossbar Switches

Lotfi Mhamdi, *Member*, *IEEE*

Abstract—Internet traffic is a mixture of unicast and multicast flows. Integrated schedulers capable of dealing with both traffic types have been designed mainly for Input Queued (IQ) buffer-less crossbar switches. Combined Input and Crossbar Queued (CICQ) switches, on the other hand, are known to have better performance than their buffer-less predecessors due to their potential in simplifying the scheduling and improving the switching performance. The design of integrated schedulers in CICQ switches has thus far been neglected. In this paper, we propose a novel CICQ architecture that supports both unicast and multicast traffic along with its appropriate scheduling. In particular, we propose an integrated round-robin-based scheduler that efficiently services both unicast and multicast traffic simultaneously. Our scheme, named Multicast and Unicast Round robin Scheduling (MURS), has been shown to outperform all existing schemes under various traffic patterns. Simulation results suggested that we can trade the size of the internal buffers for the number of input multicast queues. We further propose a hardware implementation of our algorithm for a 16×16 buffered crossbar switch. The implementation results suggest that MURS can run at 20 Gbps line rate and a clock cycle time of 2.8 ns, reaching an aggregate switching bandwidth of 320 Gbps.

Index Terms—High-performance buffered crossbars, integrated scheduling design, unicast, multicast.

1 Introduction

The growing number of newly emerging applications such as teleconferencing, distance learning, IPTV, etc., on the Internet has created an increasing need for efficient multicast traffic support. In addition to point-to-point (unicast) communications, a network node (high-speed IP routers, Gigabit Ethernet switches, and Frame Relay switches) is also required to deal with point-to-multipoint (multicast) communications and the combination of the two. Contrary to traditional switch design where unicast and multicast traffic flows are treated separately, designing a switching algorithm capable of scheduling heterogenous, yet simultaneous, different traffic types is becoming increasingly important.

To date, little research has been done on the design of integrated algorithms that support both unicast and multicast traffic types. The scheduling algorithms presented are, in fact, a combination of earlier unicast and multicast algorithms unified in one integrated scheduler. The input queuing structure has also been a combination of unicast queuing structure and multicast queuing structure. The widely used unicast queuing structure is the virtual output queuing (VOQ) [1], since it avoids the head-of-line (HoL) blocking problem [2]. As for multicast traffic, a multicast packet (cell) can have more than one destination, known as its *fan-out set*. Consequently, a multicast queuing structure can vary from just one multicast FIFO queue per input to $2^N - 1$ queues per input, where N is the number of output ports of the switch. Depending on the input queuing structure, integrated scheduling algorithms have been proposed. They were mainly proposed for the input queued (IQ) crossbar-fabric-based switching architecture because of its scalability, low hardware requirements, and its *intrinsic multicast capabilities*. Most of these algorithms were based on input VOQs for unicast traffic and one FIFO queue for multicast traffic, such as Eslip [3] and others [4]. Other algorithms [5] used VOQs for unicast and k queues for multicast traffic, where $1 < k \ll 2^N - 1$. The major drawback of these algorithms lies in their inability to either achieve high performance or run at high speed. This is mainly due to their centralized design and the nature of the crossbar fabric switching architecture.

When small buffers are added inside the crossbar fabric chip of an IQ switch, the architecture is called Combined Input and Crossbar Queued (CICQ) switch [6]. The presence of internal buffers simplifies the scheduling and makes it distributed. Instead of one centralized and complex scheduler, a CICQ switch maintains one scheduler per input as well as one scheduler per output. These schedulers are therefore decoupled and can work independently in parallel, improving the switching performance. Substantial work has focused on designing unicast algorithms for the CICQ switching architecture [7], [8], [9], [10], [11], [12]. However, fewer results have appeared for multicast scheduling in CICQ switches [13], [14]. These algorithms, unicast and multicast, have been shown to have superior performance than all algorithms proposed for the IQ buffer-less switching architecture.

Despite the CICQ switches potential in solving the scalability and scheduling complexity issues faced by their buffer-less predecessors, the problem of scheduling integrated (unicast and multicast) traffic in CICQ switches has

• *The author is with the Computer Engineering Department, EEMCS, Delft University of Technology, 2628 CD Delft, The Netherlands. E-mail: lotfi@ce.et.tudelft.nl.*

Manuscript received 6 Mar. 2008; revised 26 Nov. 2008; accepted 8 Dec. 2008; published online 18 Dec. 2008.
Recommended for acceptance by B. Parhami.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-03-0089.
Digital Object Identifier no. 10.1109/TPDS.2008.262.

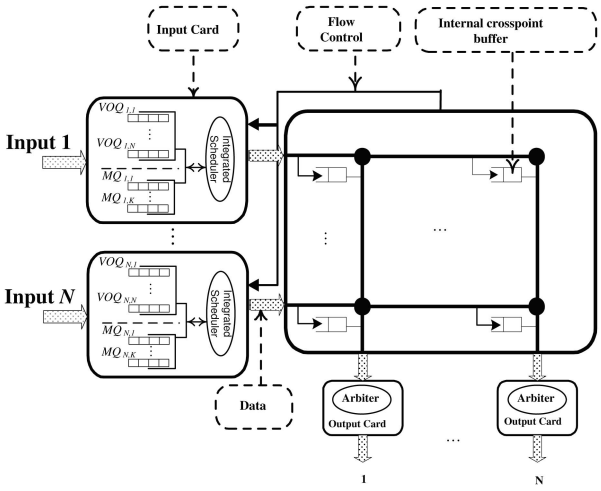

Fig. 1. The integrated CICQ switching architecture.

not been addressed. In this paper, we fill this gap and propose the following:

- An integrated CICQ switching architecture that supports concurrent unicast and multicast flows: The proposed architecture, shown in Fig. 1, is based on input VOQs for unicast traffic and k ($1 \leq k \ll 2^N - 1$) FIFO queues per input for multicast traffic. We devise a cell placement algorithm, named *Modulo*, which efficiently maps incoming traffic to the input, k, multicast queues. Our proposed placement scheme was shown to outperform all alternative schemes both in terms of performance and hardware cost.
- A simple round robin scheduling algorithm, termed Multicast and Unicast Round robin Scheduling (MURS), capable of arbitrating both traffic types simultaneously: Different variations of the MURS algorithm are also proposed, depending on the scheduling priority of each traffic type. The proposed MURS algorithm was shown, through simulation, to achieve high performance and outperform alternative algorithms under various traffic scenarios and combinations of unicast and multicast fractions. We further propose a possible hardware implemention for the MURS algorithm for a 16×16 buffered crossbar switch. It has been segmented into seven clock cycles with a clock cycle time of 2.8 ns. The hardware implementation results suggest that our proposed algorithm can sustain up to 20 Gbps line rate, allowing every switch line card to forward more than 47 million ATM cells per second.
- Simulation results showed that we can trade the size of the internal buffers for the number of input multicast queues. The reduction in the number of big input queues at the expense of adding small extra internal buffering is very important not only because of the cost savings in terms of buffering and power consumption, but also because it greatly reduces the complexity of the scheduler in terms of information exchange, resulting in faster and more scalable switching.

The remainder of this paper is organized as follows: Section 2 presents background knowledge and related work. In Section 3, we introduce the integrated CICQ switching architecture. We discuss the multicast queues management and propose an efficient multicast cell assignment scheme that is named *Modulo*. Section 4 introduces the proposed MURS algorithm, along with two variations: one for unicast priority scheduling and the other for multicast priority scheduling. Section 5 presents the performance study of our algorithms with a comparison to existing schemes. In Section 6, we propose a possible hardware implementation of our proposed scheme, along with its area and timing results. Finally, Section 7 concludes the paper.

2 Background and Related Work

The scheduling algorithm is a critical block for high-speed switches. When incoming traffic reaches the switch input cards, the scheduling algorithm resolves contentions, finds a conflict-free match between input-output pairs, and decides which inputs (outputs) are eligible to send (receive) data. Designing schedulers capable of keeping up with the scalability of the switch in line speed and/or port count is as important as challenging.

2.1 Background

The problem of packet scheduling has been extensively studied over the past two decades for IQ buffer-less crossbar-based switches. Most of the research work has focused either in a purely unicast or a purely multicast context. A plethora of unicast scheduling algorithms have been proposed. Irrespective of the incoming traffic, the input queuing structure influences the scheduling algorithm and the overall switch performance. When FIFO input queues are used, the HoL blocking problem severely limits the throughput of the system. This blocking can be prevented by adopting the VOQ structure [1]. Optimal scheduling algorithms have been proposed for IQ packet switches with VOQs [15]; however, they are too complex to run at high speed. As a result, many practical iterative algorithms have been proposed [16], [17].

Likewise, multicast traffic scheduling has been studied and many scheduling algorithms were proposed. When a multicast packet (cell) arrives at an input port, it can have one or more destination(s) indicated by its *fan-out set*. For an $N \times N$ switch, the fan-out of a cell can range from one to $2^N - 1$. Because of the importance of the input queuing structure, different queuing strategies were proposed for multicast traffic. The optimal solution is where $2^N - 1$ different queues are maintained at each input [18]. This structure completely avoids the HoL problem. However, this architecture and its scheduling algorithm are impractical even for a medium-sized switch. A single FIFO per input along with multicast scheduling algorithms were proposed [19]. However, the HoL limits the performance of these algorithms. As a compromise between using only one multicast FIFO per input or matching the whole number of fan-out configurations, other researchers propose to allocate a small number, k, of multicast queues per input [20], [21]. Because k is less than the fan-out set cardinality, how to distribute incoming traffic over the k queues is important as

it affects the scheduling performance. Specific cell placement schemes have been presented to address this issue [20], [21].

Similar to the work on IQ switches, CICQ switches have drawn a lot of interest recently due to the advantages they offer in reducing the scheduling complexity and scaling the switching performance. Obviously, these advantages do not come for free. For an $N \times N$ CICQ switch, N^2 small buffers are added inside the crossbar. Fortunately, VLSI density increases made it possible to embed enough memory inside the crossbar fabric chip. CICQ switches use distributed schedulers, one per input (input scheduler) and one per output (output scheduler). These schedulers can work independently in parallel and were shown to achieve good performance while being easily implementable in hardware. Many unicast scheduling algorithms were proposed. These algorithms can be classified into weight-based schemes [9], [10] and Round Robin (RR)-based schemes [7], [8]. Not much attention, however, has been dedicated to scheduling multicast traffic in CICQ switches. We first proposed a CICQ multicast switching architecture based on one FIFO queue per input along with a multicast round-robin-based scheduler [13]. This architecture was shown to exhibit better performance than all previous results. A CICQ switching architecture with k input multicast queues per input was proposed in [14]. Because a cell placement scheme is needed to enqueue the incoming data, the authors presented some cell placement schemes, such as CRRA and BRRA [14]. While these cell placement schemes ensure a fair and good distribution of the traffic over the input queues, they fail to prevent the packets out of sequence problem.

2.2 Related Work

Despite the substantial work advocated to either unicast or multicast scheduling, only little has been done on integrating unicast and multicast traffic. Except [22], where the architecture is a shared memory, the few other algorithms have been proposed for the IQ buffer-less switching architecture. In [4], the problem of integration of unicast and multicast has been addressed and its hardness has been derived. At each input, the queuing architecture was based on VOQs for unicast and one multicast queue for multicast. A practical algorithm was proposed that consists of scheduling multicast traffic first and leaving the unicast traffic for idle inputs (or outputs). While this solution achieves good performance, it leads to permanent starvation of unicast flows. The Eslip algorithm [3], based on round robin scheduling, attempts to overcome the starvation issue and maintain fairness among unicast and multicast flows. In more recent work [5], the input queuing structure used was based on VOQs for unicast and a small number, k, of multicast queues for multicast per input. The authors proposed integrated algorithms based on previous unicast and multicast scheduling algorithms. The integration was based on some priority metrics, such as time and/or multicast service ratio. Other researchers [23] have used VOQs for both unicast and multicast traffic together, where unicast packets are treated as multicast with a fan-out set of one. An iterative algorithm was proposed for this architecture [23] and was shown to have better performance than other unbuffered crossbar scheduling algorithms. The main drawback of the above proposed algorithms is that they perform many iterations in order to achieve good performance, limiting their scalability in port count and/or speed per port.

The CICQ switching architecture has shown superior performance over IQ switches, especially in terms of scheduling (unicast and/or multicast). In CICQ switches, no centralized scheduler nor many iterations are required. A scheduling cycle consists of three independent and parallel phases: input scheduling, output scheduling, and flow control [24]. To the best of our knowledge, no work has been done with respect to integrating unicast and multicast scheduling in CICQ switches. Motivated by the above, in the remainder of this paper, we propose an integrated CICQ-based switching architecture, along with its appropriate scheduling, which supports both unicast and multicast traffic simultaneously.

3 The Integrated CICQ Switching Architecture

We consider the CICQ switch model depicted in Fig. 1. Incoming variable-size packets are segmented into fixed-size units, called *cells*, upon their arrival to the input queues of the switch. Cells are reassembled back into packets before their departure from the output ports. Time is slotted such that every time slot is equal to a scheduling cycle (as defined in Section 2.2).

3.1 Reference Architecture

The proposed switch model consists of an $N \times N$ buffered crossbar fabric switch. This architecture differs from conventional CICQ switches [6] in its input queuing structure as well as its input scheduling. There are N input ports, each maintaining two types of queues: unicast traffic queues and multicast traffic queues. The VOQ structure is adopted for unicast queues and there are N VOQs per input, one per output. When a unicast cell, destined to output j, arrives at input i, it is placed in $VOQ_{i,j}$. A multicast cell can have a fan-out set, Φ, where $\{|\Phi| \text{ such that}, 1 \leq |\Phi| \leq N\}$. To cover all possible fan-out sets would require $2^N - 1$ multicast queues, one per fan-out set. This is clearly infeasible for a medium or large switching system. In our model, each input maintains a small number, k, of multicast FIFO queues per input, where $\{k|1 \leq k \ll 2^N - 1\}$. At each input, multicast queues are denoted by $MQ_{i,j}$, where $\{(i,j)|1 \leq i \leq N; 1 \leq j \leq k\}$. A *cell assignment policy* has to take place in order to map incoming cells to the multicast queues. This is discussed in Section 3.2.

The buffered crossbar fabric chip contains N^2 distributed crosspoint buffers, denoted by XP. A cell, coming from input i and destined to output j, is buffered in $XP_{i,j}$ while inside the crossbar chip. In addition to the input queuing structure, each input card contains an *integrated input scheduler*. The scheduler, at each input, examines the HoL cells of the *eligible* queues belonging to that input and selects one cell to be transmitted to the buffered crossbar fabric chip. An input VOQ is deemed eligible if it is not empty and its corresponding XP is not full. An input multicast queue, MQ, is considered eligible if it is not empty and at least one of its destination output ports corresponds to a nonfull XP. Each output contains an output buffer and an output scheduler. The output scheduler at output j examines the content of $XP_{i,j}\{i|1 \leq i \leq N\}$ and selects one cell to be transferred to the output port. Both input and

output scheduling are discussed in Section 4. A flow control mechanism is implemented to continuously communicate the state of the internal buffers to the input schedulers to prevent XPs overflow. The implementation of the flow control mechanism is key to the scheduler performance and the internal crossbar buffers sizing. In this paper, we use the following flow control. Every time slot, each input scheduler receives N flow control bits (from the buffered crossbar core), indicating the status of its corresponding XPs (bit value 0 if XP is full, 1 otherwise).

3.2 Multicast Cell Assignment

Before going into detail, we first explain why multicast cells require a cell assignment scheme (policy) in order to place incoming multicast cells into specific multicast queues (MQ) while waiting their turn to be scheduled. In general, the cardinality of the fan-out set, Φ, of a multicast cell can vary between $|\Phi_{\min}|$ and $|\Phi_{\max}|$. Since a multicast cell can have a minimum of 1 destination output port and a maximum of N destination output ports, we can set $|\Phi_{\min}| = 1$ and $|\Phi_{\max}| = N$. Therefore, the cardinality of the set of all fan-out sets, C_Φ, can be calculated as follows:

$$C_\Phi = |\wp(\Phi)| = \sum_{|\Phi_i|=1}^{N} \binom{N}{|\Phi_i|} = 2^N - 1, |\Phi_{\min}| \leq |\Phi_i| \leq |\Phi_{\max}|. \quad (1)$$

In order to completely avoid the HoL blocking problem, multicast cells having the same fan-out sets must be placed in the same *separate* multicast queue (MQ), which requires as many MQs as C_Φ. Maintaining such a big number of MQs is, however, impractical. The alternative is to use a small number, k, of MQs per input to accommodate the incoming multicast cells. Because k is smaller than C_Φ, cells with different fan-out sets have to be queued in the same MQ and every MQ receives cells with at most $\lceil \frac{C_\Phi}{k} \rceil$ different fan-out sets. Thus, a *cell placement scheme* is required to map incoming multicast cells into the k multicast queues.

Since, in our model (Fig. 1), the number of MQs, k, maintained at each input is much smaller than the number of all fan-out configurations, C_Φ, a cell assignment policy is required in order to map incoming cells to the MQs. This has a significant effect on the scheduling performance. Previous work [21] has outlined some criteria in designing such a policy: 1) The heads of the MQs should be *diverse* in order to span a large number of the outputs. This would ensure more scheduling opportunities and work conservation. 2) Cells with the same, or similar, fan-out sets should be stored in the same queue, to reduce HoL blocking and prevent the out-of-sequence delivery problem. Many cell assignment schemes existed, such as Majority [21], Minimum Distance Queue (MDQ), and Load Balanced Queuing (LBQ) [20]. The MDQ scheme assigns a representative fan-out set per MQ. Each incoming packet is inserted in the MQ with the representative having the minimum hamming distance from the packet's fan-out set. The LBQ scheme assigns packets to MQs based on either sorting the packets fan-out sets or fan-out values. The previous two cell assignment algorithms are inspired by the Majority algorithm.

In the majority scheme, each MQ is associated with a bit mask that corresponds to a subset of outputs. The mask is created by forming a balanced partition of the set of outputs whose cardinality is equal to the number of MQs. For example, if an 8×8 switch has 2 MQs per input, the partition will result in every MQ's mask equalling $\frac{8}{2} = 4$. The drawback of this partition is that the number of bits set for each mask decreases with increasing numbers of MQs. As a result, this assignment does not adequately capture the multicast destination sets of packets. The solution to the bit mask partition was to allow the bit masks to overlap. In addition to the original balanced partition, the remaining bits of every mask are set at random. Because masks overlap, Majority has to deal with cases when a packet has the maximum match with more than one mask. Majority resolves this by associating multiple sets of masks for each MQ and resolves ties with multiple levels of comparisons, with the final comparison breaking ties statically.

Unfortunately, in addition to its complex and time-consuming procedure, Majority suffers a major problem by causing MQs to be unbalanced in the number of packets they cater for, causing part of these MQs to become underutilized. This problem arises from the statical tie breaking mechanism of Majority. Because ties are broken statically, packets with small numbers of fan-out sets will always be inserted in the same MQ. This results in the same MQs receiving more packets than others and severely limits the performance of Majority, and defeats the purpose and advantage of increasing the number of MQs per input port (Figs. 7 and 8 illustrate this effect).

3.3 The Modulo Cell Assignment Algorithm

Our queuing structure implements a simple and efficient cell assignment scheme that does not require any sorting. Our cell assignment scheme, named *Modulo*, works as follows:

Modulo:

- For each input, i, do
- If an incoming multicast cell, c, has a fan-out set Φ_c
  - Insert c in $MQ_{i,j}$, where $j = |\Phi_c| mod(k)$.

In addition to its simplicity, especially if the number of MQs, k, is a power of two, *Modulo* meets all previously mentioned criteria for efficient cell assignment. To better understand this, let us consider an example. Assume we have an 8×8 switch and two multicast queues per input ($k = 2$). At each input, i, *Modulo* places cells with even fan-out sets in $MQ_{i,0}$ and the cells with odd fan-out sets in $MQ_{i,1}$. This way, the heads of the MQs can span large numbers of destinations[1] (i.e., the fan-out set cardinality of the HoL cell of $MQ_{i,0} = 6$ and that of the HoL cell of $MQ_{i,1} = 3$). Moreover, cells with the same fan-out sets are ensured to be queued in the same MQ, avoiding the out-of-sequence problem. Additionally, our scheme exhibits one further important property as a fair scheme, in the sense that it gives equal opportunities to the cells to advance to the head of the queues irrespective of their number of destinations. This is important as there are scheduling algorithms that use the fan-out set as the weight for priority

1. This is assuming uniform traffic. When the traffic is nonuniform or bursty, the HoL fan-out sets may overlap to a large degree.

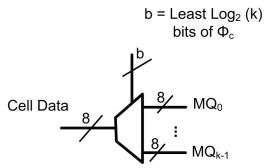

Fig. 2. The Modulo cell placement scheme.

scheduling and, unless the cells fan-out sets per MQ are diverse, MQ starvation (unfairness) can occur.

Finally, the *Modulo* scheme has simple hardware requirement allowing it to place cells in the MQ at very high speed. Assuming there are k MQs per input and that each incoming cell has a fan-out set value $|\Phi_c|$, our scheme can be implemented as DeMux with its input consisting of the cell data, its select bit(s) consist(s) of the least $\log(k)$ bits of $|\Phi_c|$ and its outputs consist of MQ_j, where $\{j|0 \leq j < k\}$, as depicted in Fig. 2. We implemented the *Modulo* scheme in reconfigurable logic, using the Xilinx Virtex IV [25] as the target device and the Xilinx ISE platform 7.1 design flow platform. The design was simulated for different numbers of MQs (k = 2, 3, 4, and 8, respectively) and the post place and route results in terms of area and timing are depicted in Table 1. We can see from the table that the delay of the scheme is very small irrespective of the number of MQs used per input. When k = 2, the *Modulo* scheme checks the least significant bit of the cell's fan-out value; if it is 0, the cell will be placed in MQ_0; else it will be placed in MQ_1. When $k = 8$, however, we can see that the delay is higher. It is of note that when setting k = 3 or 4 results in the same delay.[2] As a result, it is better to choose a number of MQs, k, that is a power of 2 as it results not only in a full DeMux, but also in balanced MQs in the number of fan-out sets per queue. This is because, generally, the switch sizes are a power of 2 and using a small number k of MQs per input (which is also a power of 2) will result in balanced queues in the number of fan-out sets that they can accommodate. Next, we need to devise the appropriate algorithm to schedule the transfer of cells from the input MQs to the output ports. The next section describes our proposed scheduling algorithm.

4 The MURS Integrated Scheduler

This section introduces the proposed integrated scheduling algorithms, Multicast and Unicast Round robin Scheduling (MURS). Because the input queuing structure consists of two types of queues and two types of traffic (unicast and multicast), extra focus has to be placed on the input scheduler at each input. The input scheduler is not only required to *select* cells to be transmitted to the fabric chip but also needs to decide *when* to choose a cell and *from which* set of queues (VOQs or MQs). This is called the *integration* phase of the scheduler. The integration phase of the scheduler determines the scheduling priority of each traffic type. The selection policy of our scheme is based on round robin because of its fairness and simple hardware implementation. The selection

2. The reason for this is because the select bits b are equal to 2 resulting in a 2-to-4 DeMux, irrespective of whether k is equal to 3 or 4. This results in the same delay and different area due to the partial use of the DeMux.

TABLE 1
Implementation Results for Different Numbers of MQs

Number (k) of MQs	Area		Delay (ns)
	Slices	Equivalent Gate Count	
k = 2	4	248	1.5
k = 3	13	432	1.9
k = 4	22	576	1.9
k = 8	39	1032	2.4

policy is fixed and independent of the integration phase. Before discussing the integration policy, we first describe the selection policy since it is fixed. The specification of the selection policy is as follows:

Selection Phase:
Select_Queue (Queue_type, Pointer_type):
N = number of queues in Queue_type;
i = current input;
- Starting from the *Pointer_type* index, select the first eligible queue $EQ_{i,j}$ and send its HoL cell* to the internal buffer $XP_{i,j}$.
- Move *Pointer_type* to the location $(j+1)(modN)$.

The integration phase of the scheduler decides the priority of the scheduling of cells at the queue type level. Each input scheduler maintains two priority pointers: a unicast pointer (UP) and a multicast pointer (MP). If the set of VOQs (MQs) is chosen to select a cell from, the round robin pointer will be based on UP (MP). Note that we consider *fan-out splitting* when serving multicast flows [19]. The integration phase is responsible for deciding which set of queues to choose from. As the traffic can be unicast, multicast, or a mix of both, we derived three integration policies. The first is called MURS_uf (unicast first) and always gives priority to unicast traffic. The second, MURS_mix, is designed to be a fair policy and treats both traffic types equally. MURS_mix gives priority to unicast traffic during even time slots, while multicast traffic is favored during odd time slots. The third integration policy is named MURS_mf (multicast first) and always gives priority to multicast traffic. Each integration policy corresponds to an input scheduling algorithm.

MURS_uf always gives priority to unicast flows over multicast flows. Therefore, in the presence of mixed unicast and multicast traffic, MUR_uf will always favor the VOQs set of queues to receive service and leave remaining idle connections to multicast flows. As a result, this scheme will produce more one-to-one connections than one-to-many connections. This causes performance degradation under heavy loads since when a unicast cell is chosen to be sent from an input port containing multicast cells, only one cell (*copy*) will be transmitted to the buffered crossbar fabric. The specification of the MURS_uf algorithm is as follows:

MURS_uf:
*/*Always Unicast traffic first (prioritized)*/*:
- Select_Queue(VOQs , UP);
- If no queue was selected
  - Select_Queue(MQs , MP);

*If the cell is multicast, then only copies destined to c outputs are sent, where $c|c \in \{1, \ldots, N\}$ and $XP_{i,c}$ is available. Other copies will have to compete in later time slots.

If, instead of MURS_uf, we allow preference to multicast flows in the presence of unicast flows at the same input, this would result in more cells (*copies*) being transmitted to the buffered crossbar. As a result, the performance can be greatly scaled up. This is exactly what MURS_mf algorithm achieves, by favoring multicast flows over unicast flows. Despite the performance difference, there are similarities between MURS_uf and MURS_mf. They both have the same performance when the traffic is either purely unicast or purely multicast. Additionally, both schemes are *unfair*. Each tries to monopolize the switch bandwidth to its preferred traffic flows and this is undesirable. The specification of the MURS_mf algorithm is as follows:

MURS_mf:
*/*Always Multicast traffic first (prioritized)*/*:
- Select_Queue(MQs, MP);
- If no queue was selected
  - Select_Queue(VOQs , UP);

As a compromise between MURS_uf and MURS_mf, we propose MURS_mix. The scheduling priority of each traffic type is time slot dependent. The specification of the MURS_mix algorithm is as follows:

MURS_mix:
*/*Equal Priority*/*:
- If current time slot is even */*Unicast is served first*/*
  - Select_Queue(VOQs, UP);
  - If no queue was selected
    - Select_Queue(MQs, MP);
- Else */*Multicast is served first*/*
  - Select_Queue(MQs, MP);
  - If no queue was selected
    - Select_Queue(VOQs , UP);

In addition to its fairness in the presence of different traffic types, the MURS_mix algorithm exhibits the same performance as the other two algorithms when the traffic is all unicast or all multicast. The properties of MURS_mix makes it a good candidate for being an optimal integrated scheme because: 1) It is fair and starvation free both on the traffic level as well as the flow level. In the presence of different traffic types, MURS_mix provides equal chances (even and odd time slots) to heterogeneous traffic types to be served. At the flow level, the round robin scheduling mechanism ensures fairness to flows belonging to different queues (whether unicast or multicast) and schedules them with the same likelihood. It is worth noting that MURS_mix is not a max-min fair algorithm. Although unicast flows consume less internal switching bandwidth than multicast flows, both traffic types share equal scheduling priority. This is intended in order to maintain the same equal scheduling priority at the traffic type level (whether unicast type or multicast type). 2) MURS_mix requires simple hardware allowing it to run at high speed. 3) Finally, MURS_mix shows enhanced performance in terms of high throughput and low cell latency by comparison to existing algorithms. This is illustrated in Section 5.

For all the three input schedulers, we used the same output scheduling algorithm that we previously proposed [26] and has the following specification:

Output Scheduling (OS):
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 the pointer index, select the first nonempty crosspoint buffer (XP) and send its queued cell to the output line card.

The output scheduler is based on a *static* round robin pointer movement, wherein all the output arbiters share the same pointer. The pointer is initialized to a random position and is incremented by 1 mod(N) every scheduling cycle. The pointer setting is very important and has a twofold advantage. First, the synchronous move of the output pointer ensures that at least one complete multicast cell is discharged every scheduling cycle (since all pointers always point to the same XP). Second, while our scheme adopts a fan-out splitting discipline resulting in higher throughput, it also closely resembles a nonfan-out splitting discipline (by guaranteeing the complete discharge of a multicast cell every time slot) which results in optimized use of internal bandwidth on the serial links between the input line cards and the buffered crossbar fabric core. If we use static round robin for the the MURS_mix and combine it with the OS at the output scheduling, no HoL cell (wether unicast or multicast) will be starved. Furthermore, every HoL cell is guaranteed to be served in no more than $N(N+k)$ time slots from the time it reaches the head of its queue. Therefore, by considering the input/output pointer static movement, every HoL cell is guaranteed to be transferred from its input queue to its destination output port in no more than $N(N+k+1)$ time slots.

5 Performance Results

This section presents the simulation study of two CICQ switching systems of 8×8 and 16×16 and employing the MURS set of algorithms. The experimental results are structured in four parts. In the first part of the experiments, we compare the performance of MURS_mix to the Eslip algorithm which uses a bufferless crossbar switch [3]. We choose the Eslip algorithm for comparison because it is one of the first and few proposed integrated schedulers for bufferless crossbars, it is practical, and is being deployed on commercial switching products [3]. The second set of experiments studies the performance of our set of algorithms under different settings of MQs and traffic arrival. The third section of the experiments analyzes the effect of varying both the number of MQs and the size of the internal buffers. Additionally, we observe the stability of the input queues under different traffic, input queuing, and internal buffer size settings. The last part of the experiments studies the performance of the MURS algorithms under nonuniform unbalanced traffic conditions. The performance metrics studied here are the average cell latency and throughput. Simulations run for 1 million time slots and

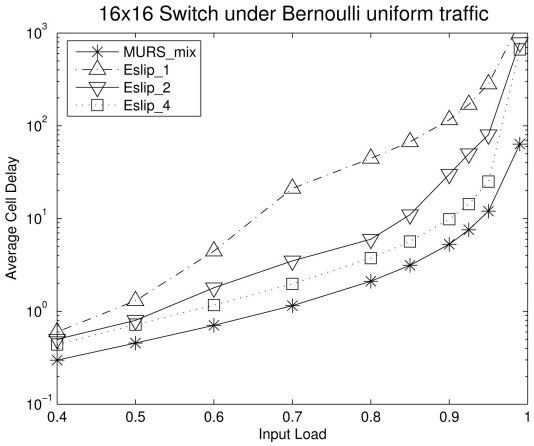

Fig. 3. Average cell delay of MURS_mix and Eslip under Bernoulli uniform unicast traffic ($f_m = 0$).

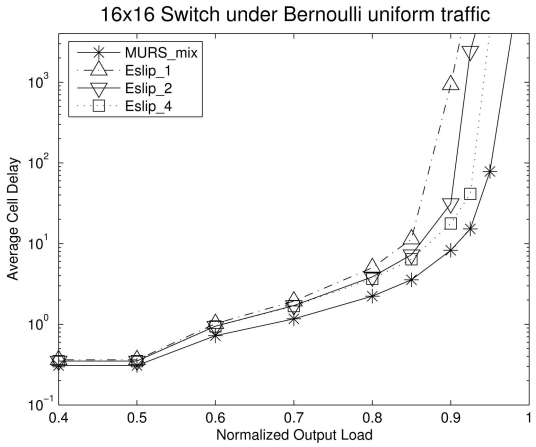

Fig. 4. Average cell delay of MURS_mix and Eslip under Bernoulli uniform mixed traffic ($f_m = 0.5$).

statistics are gathered when fourth of the total simulation length has elapsed.

We studied the performance of our set of algorithms under the Bernoulli uniform, bursty uniform and nonuniform unbalanced traffic scenarios. Arriving cells can be either unicast or multicast. Cells arrive with a rate denoted by λ. Since the traffic is uniform, λ is the input load of the switch. The departure rate is denoted by μ. Similarly, μ is the output load of the switch. We consider admissible traffic, no input or output is oversubscribed. Because the traffic is a combination of unicast and multicast flows, the input load consists of a multicast fraction (f_m) and a unicast fraction (f_u), where $\{(f_m, f_u)|f_m = 1 - f_u\}$. The fan-out set, Φ, of multicast cells has cardinality (fan-out number) $|\Phi|$ which is uniformly distributed between 1 and N (depending on the switch size, N can be 8 or 16) and all outputs have equal chances to be the destination of a multicast cell. Based on the above, the relationship between the switch input and output loads is expressed by (2):

$$\mu = \lambda(f_u + |\Phi| f_m). \quad (2)$$

In our simulation, we averaged the cells fan-out set[3] to be $|\Phi| = 8$. Following our settings and substituting f_u with f_m, we have

$$\mu = \lambda(1 + 7f_m). \quad (3)$$

For example, if we set f_m to be 0 in (3), the traffic is all unicast. When we set it to 1, the traffic becomes pure multicast. Whereas, if we fix μ to 1, for example (switch fully loaded), we can vary f_m and see its effect on the throughput. When $f_m = 0.5$, the incoming traffic is evenly distributed between unicast and multicast flows.

5.1 MURS_mix versus Eslip

Because Eslip is based only on a single multicast queue per input, we used the same settings with MURS_mix by using just one MQ ($k = 1$), with a crosspoint buffer (XP) of size one cell, for fair comparison. Note that Eslip_i refers to Eslip with i iteration(s). In Fig. 3, we compare the average delay performance of MURS_mix and Eslip under Bernoulli uniform traffic with all cells being unicast. As depicted in Fig. 3, the performance of MURS_mix is always higher than Eslip irrespective of the number of iterations performed by the latter. Fig. 4 depicts the average cell delay of the two algorithms when the input traffic is evenly distributed between unicast and multicast flows ($f_m = 0.5$). Again, MURS_mix has a shorter average delay than Eslip. Fig. 5 depicts the delay performance of MURS_mix and Eslip when the incoming traffic is all multicast, we can see that MURS_mix still achieves better delay than Eslip. As we can see from the previous three figures, MURS_mix always achieves a far shorter delay than Eslip irrespective of the incoming traffic type.

We wanted to compare the performance of MURS and Eslip for different mixed traffic settings. However, checking all possibilities of mixed traffic requires tuning f_m from 0 to 1 and observing the throughput. To this end, we fixed the output load, μ, to be 100 percent (fully loaded system) and recorded the throughput of each algorithm as f_m varies from 0 to 1. Fig. 6 compares the maximum achievable throughput of MURS_mix and Eslip_4 under different switch sizes (8×8 and 16×16). MURS_mix achieves higher

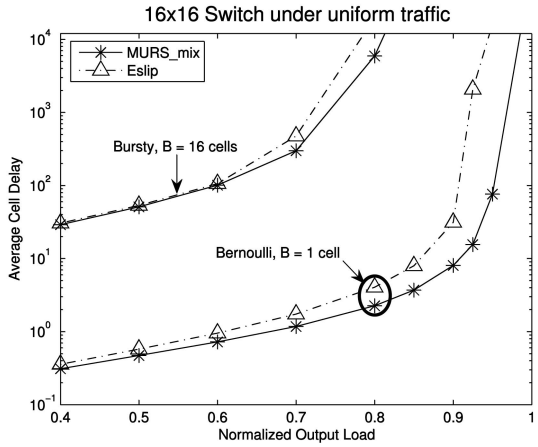

Fig. 5. Average cell delay of MURS_mix and Eslip under Bernoulli uniform multicast traffic ($f_m = 1$).

3. Note that when the size of the switch is 8×8, the average fan-out size becomes 4 and (3) becomes: $\mu = \lambda(1 + 3f_m)$.

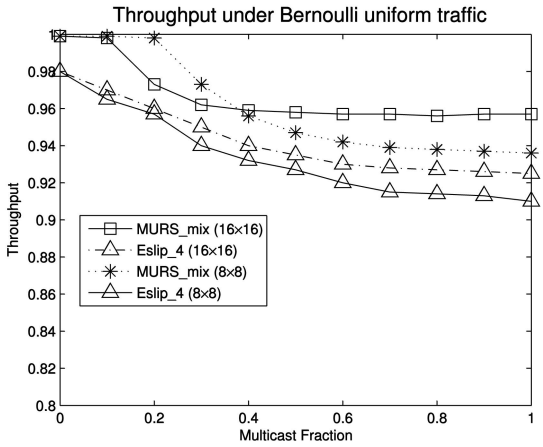

Fig. 6. Throughput performance of MURS_mix and Eslip_4 under different switch sizes and different multicast fractions.

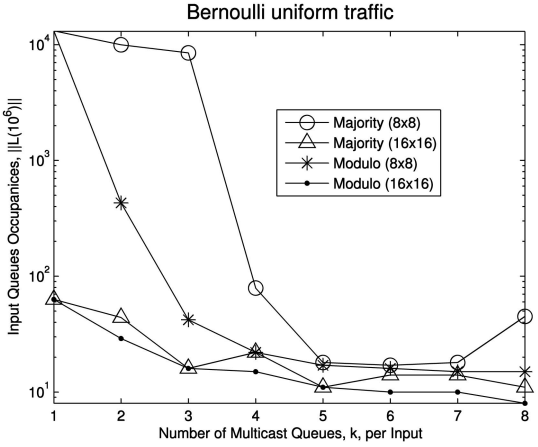

Fig. 8. Input queues occupancies of Modulo and Majority under Bernoulli uniform traffic.

throughout than Eslip irrespective of the switch size and/or the multicast fraction of the incoming traffic. Note that while each of our algorithms have a smaller delay than Eslip, we chose MURS_mix because it is more analogous to Eslip in the sense that is does not prioritize one traffic type over another.

5.2 The Effect of MQs Number, k

The remainder of the simulation is conducted for multicast queues set MQ per input equal to or greater than one ($k \geq 1$). Note that in this section, the internal buffer size is kept to one cell per XP. We first studied the performance of the Modulo cell assignment algorithm. We compared the Modulo and Majority cell placement schemes in terms of throughput and input queues occupancies (see (4) in Section 5.3) for two switch sizes of 8×8 and 16×16, both employing the MURS_mix algorithm under uniform traffic. We varied the number of MQs per input and observed the maximum throughput achieved by each of the cell placement algorithms. We can see from Fig. 7 that the throughput of Modulo increases proportionally with the number of MQs. However, Majority tends to have even lower throughput with increasing number of MQs. This is attributed to the unbalanced MQs effect by its assignment policy and statical tie breaking mechanism. Fig. 8 depicts the MQs occupancies for each algorithm just before saturation (at 95 percent input load). Again, Modulo outperforms Majority irrespective of the switch size or number of MQs. Since the input traffic is uniform, by applying Little's Law [27], we can directly deduct the average cell delay under each scheme from Fig. 8. Since the input load is 95 percent, the values of Fig. 8 are similar to the average cell delay. This delay does not include the internal buffers delay, which is bound by the number of input ports N, as discussed in Section 4.

We study the average cell delay performance of each of our algorithms for k = 1, 2, and 4, respectively, and evenly distributed traffic over unicast and multicast ($f_m = 0.5$). Fig. 9 depicts the average delay for an 8×8 switch and Fig. 10 shows the average cell delay for a 16×16 switch. As expected, the MURS_mf scheme has the best delay irrespective of the arrival traffic and the switch size. This is because it gives priority to multicast flows over unicast flows resulting in more connections per scheduling cycle. This result holds irrespective of the number of MQs used per input. MURS_uf, however, has the worst delay because

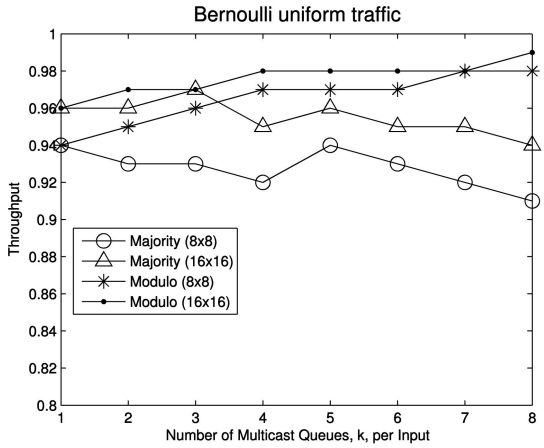

Fig. 7. Throughput comparison between Modulo and Majority cell placement schemes under Bernoulli uniform traffic.

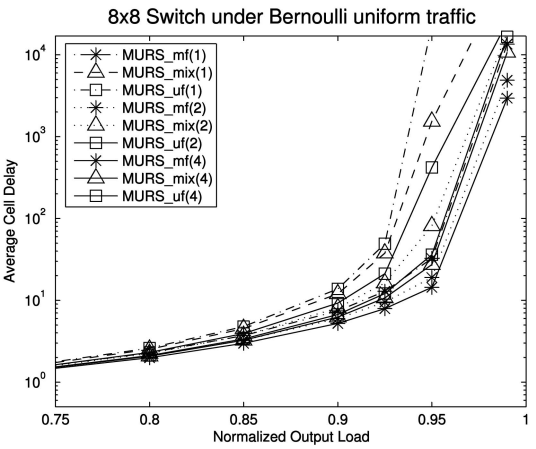

Fig. 9. Average cell delay of an 8×8 CICQ switch running MURS with different numbers of MQs per input and mixed input traffic ($f_m = 0.5$).

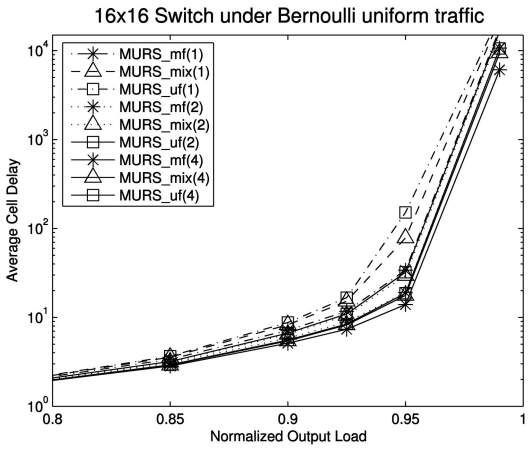

Fig. 10. Average cell delay of a 16×16 CICQ switch running MURS with different numbers of MQs, k, per input and mixed input traffic ($f_m = 0.5$).

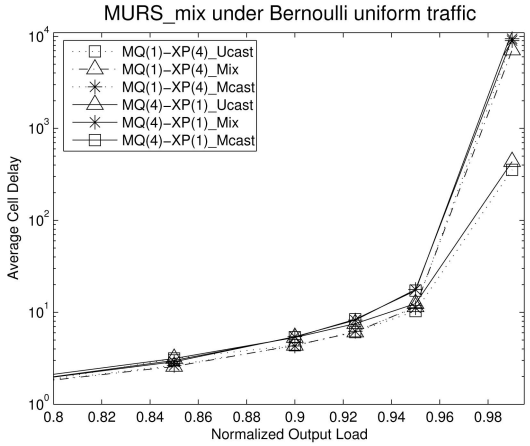

Fig. 12. Average cell delay of MURS_mix as a function of the number of MQs, the XP size, and input traffic combinations.

it prioritizes unicast over multicast, resulting in fewer cells transferred to each output per scheduling cycle. MURS_mix has a moderate average delay because it treats both traffic types with the same priority. Overall, MURS_mix is the best choice due to its fairness. Fig. 11 depicts the average cell delay of MURS_mix with varying switch sizes and different numbers of MQs. We can see that the average cell delay improves with increasing numbers of MQs and this is due to the role of the MQs in reducing the effect of the HoL blocking problem.

5.3 The Number of MQs versus the XP Size

Due to the importance of the internal buffers in simplifying the scheduling, we tested our algorithms under different input and internal buffer scenarios for a 16×16 switch. Fig. 12 depicts the average delay performance of the MURS_mix algorithm under three different scenarios. Incoming traffic is either all unicast ($f_m = 0$), or a mix ($f_m = 0.5$) or all multicast ($f_m = 1$). We varied the number of input multicast queues per input as well as the size of the internal buffers (XP) and studied their effect under each traffic scenario. For example, "MQ(1)-XP(4)_Ucast" corresponds to the MURS_mix algorithm with one multicast queue per input (MQ=1), four cells per internal buffer ($\mathrm{XP} = 4$) and incoming traffic consisting of unicast cells only. "MQ(4)-XP(1)_Mix" corresponds to MURS_mix with four MQs ($k = 4$) per input, one cell per XP, and a mixed incoming traffic over unicast and multicast flows ($f_m = 0.5$).

The plots in Fig. 12 show that the average delay of MURS_mix is shorter when only one multicast is used per input port (instead of four) with an internal buffer size of four cells per XP (instead of one). It is apparent that the delay of "MQ(1)-XP(4)_Ucast" should be shorter than that of "MQ(4)-XP(1)_Ucast." The reason for this is because incoming traffic is all unicast and therefore varying the number of MQs does not affect the delay. However, when the incoming traffic is mixed ("MQ(1)-XP(4)_mix") and at 99 percent input load, the average cell delay of MURS_mix is 25 percent shorter than the cell delay when using "MQ(4)-XP(1)_mix." This is attributed to the important effect of the internal buffers in absorbing the HoL blocking problem, resulting in shorter delay. The same results are observed when we also use MURS_mf, as depicted in Fig. 13.

Because the trade-off between the input multicast queues and the internal buffers is not straightforward (on-chip memory versus off-chip memory), we studied the stability of the input queues under the same settings as above. We used the L^2 norm vector representing the occupancy of all

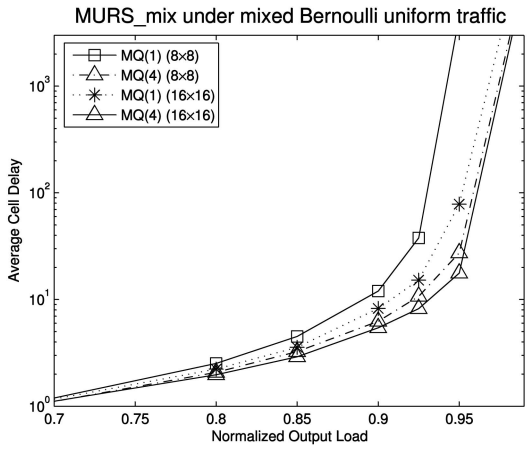

Fig. 11. Average delay of MURS_mix with different switch sizes and different MQ numbers.

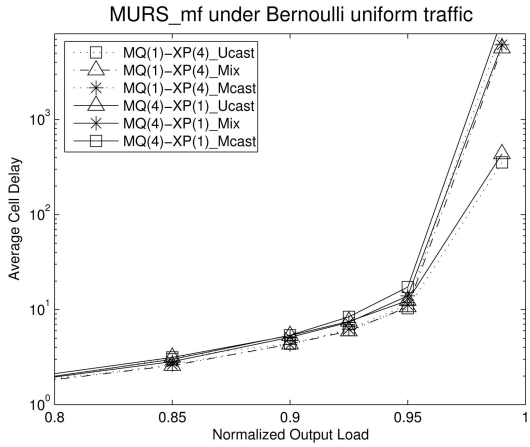

Fig. 13. Average cell delay of MURS_mf as a function of the number of MQs, the XP size, and input traffic combinations.

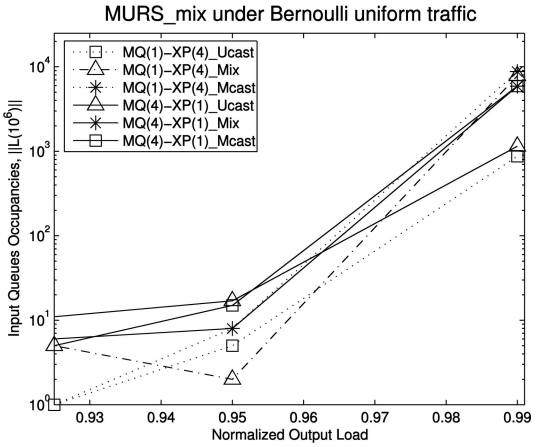

Fig. 14. Input queues occupancies of MURS_mix as a function of the number of MQs, the XP size, and input traffic combinations.

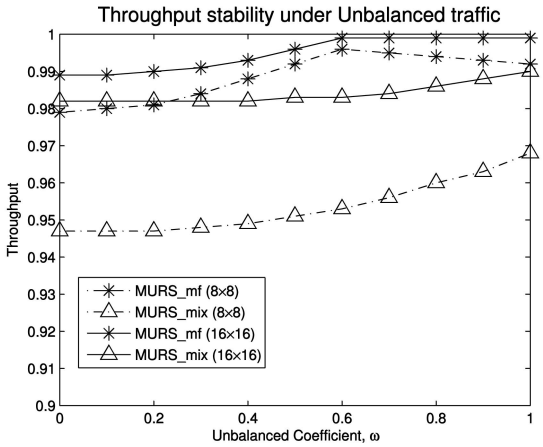

Fig. 16. Throughput stability of MURS algorithm under unbalanced traffic and different switch sizes.

input queues. Let $VOQ_{i,j}(n)$ be the number of unicast cells queued in $VOQ_{i,j}$ at time slot n and $MQ_{i,l}(n)$ be the number of multicast cells queued in $MQ_{i,l}$ at time slot n. The L^2 norm vector at time slot n is denoted by $\|L^2\|$ and defined in (4) as follows:

$$\|L(n)\| = \sqrt{\sum_{i=1}^{n}\Big(\sum_{j=1}^{n} VOQ_{i,j}(n)^2 + \sum_{l=1}^{k} MQ_{i,l}(n)^2\Big)}. \quad (4)$$

In addition to representing the occupancy of the input queues, the L^2 norm vector can be used to analyze the distribution of cells over the input queues as well as the input buffer requirement per input port.

As depicted in Fig. 14, under either mix or all multicast traffic, the input queues occupancy is smaller when we use only one multicast queue per input and an internal buffer size of four cells compared with employing four multicast queues per input and internal buffer size of one cell. Observe that in both Figs. 14 and 15, and at 99 percent input load and for both switch sizes, the L^2 norm of MQ(1)-XP(4) is approximately 1.5 times greater than that of MQ(4)-XP(1). Meaning that, $(L_{14})^2 = \frac{3}{2}(L_{41})^2$, where $(L_{14})^2$ refers to the L^2 norm of MQ(1)-XP(4). However, the total number of cells per input port is less with MQ(1)-XP(4) than with MQ(4)-XP(1). To better see this, let us assume that the occupancy of the unicast queues is the same for both cases (MQ(1)-XP(4) and MQ(4)-XP(1)) since the number of VOQs remains the same. Let a be the total number of cells (MQ occupancy) in the case of MQ(1)-XP(4) and b denote the average occupancy of each of the four MQs in the case of MQ(4)-XP(1). Then, from (4), we have

$$\sqrt{a^2} = \sqrt{\frac{3}{2}4b^2},$$
$$a = b\sqrt{6}.$$

Meaning, at 99 percent input load, the total number of cells—using just one FIFO queue per input port and an XP size of four cells—is equal to $\frac{\sqrt{6}}{4} \approx 60\%$ of the total number of cells when we use four FIFO queues per input port and XP size of one cell. This result conforms to the lower average cell delay (Figs. 12 and 13) of MURS with MQ(1)-XP(4) instead of MQ(4)-XP(1). This result can help decide on the number of MQs per input, which has a significant impact on the hardware implementation of the MURS input scheduler (see Section 6).

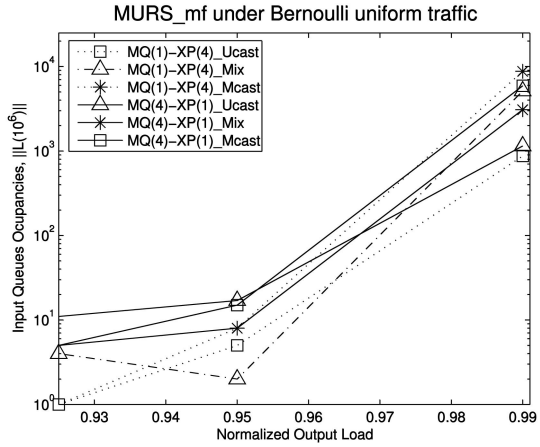

Fig. 15. Input queues occupancies of MURS_mf as a function of the number of MQs, the XP size, and input traffic combinations.

5.4 Unbalanced Traffic

We analyzed the performance of the MURS scheduling algorithms under nonuniform unbalanced traffic conditions. The unbalanced traffic is defined by using an unbalanced probability, ω. For an $N \times N$ switch, the traffic load at each input port is defined by ρ. Then, for each input port s and output port d, the traffic load, $\rho_{s,d}$, is given by

$$\rho_{s,d} = \begin{cases} \rho(\omega + \frac{1-\omega}{N}), & \text{if } s = d, \\ \rho\frac{1-\omega}{N}, & \text{otherwise.} \end{cases}$$

In this section, ρ is generated according to (3) with a multicast fraction, *fm* = 0.5. Meaning that the input unbalanced traffic is equally divided between unicast and multicast flows. Note that when $\omega = 0$, the load is uniform over all outputs and when $\omega = 1$, the traffic load is totally unbalanced (only on the diagonal).

Fig. 16 illustrates the throughput stability of MURS_mix and MURS_mf under unbalanced traffic conditions. The

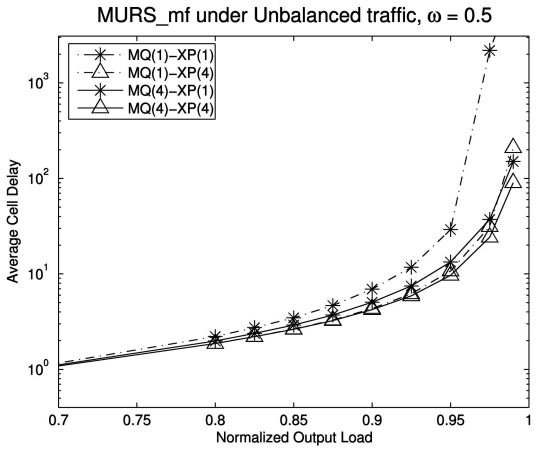

Fig. 17. Average cell delay of a 16×16 switch running MURS_mf under unbalanced traffic, with unbalanced coefficient ω = 0.5 and different queuing scenarios.

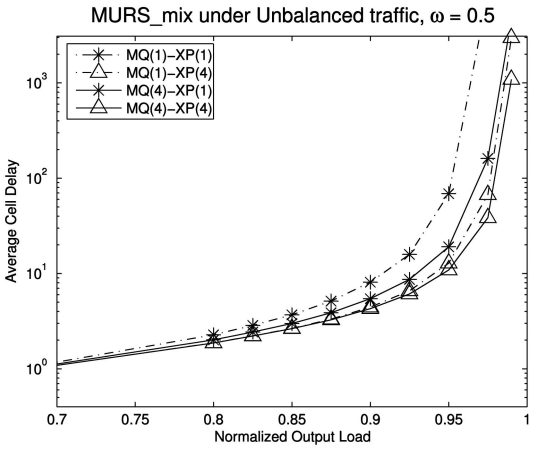

Fig. 18. Average cell delay of a 16×16 switch running MURS_mix under unbalanced traffic, with unbalanced coefficient ω = 0.5 and different queuing scenarios.

algorithms are tested for two switch sizes of 8×8 and 16×16 with 1 MQ per input and XP size of one cell. Both algorithms sustain a high throughput (≥ 95 percent), irrespective of the unbalanced coefficient, ω. MURS_mf maintains a slightly higher throughput than MURS_mix across all values of ω. Note that MURS_mf reaches a throughput peak at $\omega = 0.6$ and either stays constant (for 16×16 switch) or decreases (for 8×8 switch). The reason for this is attributed to its unfair scheduling that favors multicast flows over unicast flows, leading to performance degradation with highly unbalanced traffic conditions. On the other hand, the throughput performance of MURS_mix increases with increasing ω for both switch sizes.

We want to study the cell delay performance of the MURS algorithms under unbalanced traffic. In our experiments, we set ω to be 0.5 because this value corresponds to a hot-spot pattern with half the load destined to one output and the other half being equally shared among the rest of the outputs. Fig. 17 shows the average cell delay of a 16×16 switch employing the MURS_mf algorithm under unbalanced traffic with varying number of MQs and XP sizes. We can see that the worst delay performance corresponds to the scenario where only one MQ per input is used with internal XP size of one cell. On the other hand, using four MQs per input port and XP size of four cells results in the best average overall delay. When we exchange the number of MQs for the size of the XPs, we get similar switching delay. This conforms to the results obtained in Section 5.3. The same trend is observed with the MURS_mix algorithm, as depicted in Fig. 18. We can see that using one MQ per input port of the switch with internal XP size of four cells provides slightly better switching delay than employing four MQs per input and XP size of one cell.

Using just one MQ per input port instead of four MQs at the expense of a little increase in the size of the internal buffers results in significant reduction in the hardware complexity of the input integrated scheduler. This is because the scheduler needs to maintain the state of the fan-out sets of the HoL cells of every MQ and using just one MQ would translate in reduced information exchange, and consequently, in a shorter scheduling cycle time. For these reasons, we decided to implement the MURS scheduler with one MQ per input and XP size of four cells, as will be discussed in the following section.

6 Hardware Implementation

This section presents the hardware implementation of the MURS scheduling algorithm for a 16×16 CICQ switch with one MQ per input and XP size of four cells. Fig. 19 depicts the schematic diagram of the algorithm and the scheduling process. The design can be divided into three main blocks, as follows:

- **Unicast block:** This block is responsible for handling unicast traffic and consists of a 16-bit vector called Virtual Output Queue (VOQ) that contains the state of each VOQ in a line card. A 16-bit vector named the Eligible VOQ (EVOQ) is used for the index of the VOQs eligible for scheduling. The EVOQ is obtained by ANDing the VOQ vector with the Empty internal crosspoint, XP (EXP). A component named Masked

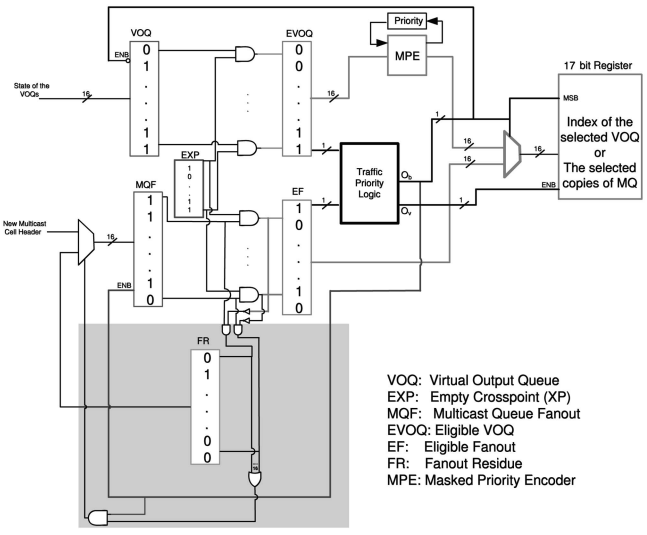

Fig. 19. The MURS input scheduling algorithm.

Priority Encoder (MPE) that is responsible for selecting the next index from the EVOQ vector in a round robin fashion. In our design, we used the MPE proposed in [28].

- **Multicast block:** This block is responsible for the multicast traffic and contains the following components. A 16-bit vector, MQF (Multicast Queue Fanout), which contains the fan-out set of the HoL cell of the MQ. A 16-bit vector, EF (Eligible Fanout), which contains the subset of outputs that the multicast cell can be sent to. The EF is the result of a logic AND of the MQF and the EXP vectors. A vector called FR (Fanout Residue) that contains the subset of unreachable output ports of the HoL cell of the MQ. This vector is obtained by ANDing the MQF and the logic inverse of the EF vector.
- **Traffic priority block:** This block manages the scheduling priority of unicast and multicast cells over time. It is designed as a state machine and works as follows: It takes as input two bits, the first bit (U_b) is the logic OR of the EVOQ vector bits and the second bit (M_b) is the logic OR of the EF vector bits. There is an internal bit (P_b) that determines which traffic is prioritized during the current scheduling cycle.[4] The value of P_b is inverted every scheduling cycle.[5] There are two output bits of the traffic priority block denoted by O_b and O_v (see Fig. 19). The value of each of them is obtained as follows: $O_b = (P_b \wedge M_b) \vee \overline{U_b}$; $O_v = M_b \vee U_b$.

  The upper output bit of the traffic priority block (O_b) is used as the select bit of the 2-to-1 Mux that decides which traffic type cell is chosen. The other output bit, O_v, indicates whether or not the output O_b is valid. The O_b bit along with the output of the the MPE (first block) and the content of the EF (second block) are used as the select bit and the two inputs of the Mux. Finally, the output of the Mux is forwarded, along with the select bit to a 17-bit register that contains the scheduling decision. This decision register is 17 bits wide with the select bit being its most significant bit (MSB). If the MSB bit is 1, then we know that the content of the register (16 bits) represents the reachable destination ports of the HoL multicast cell. Otherwise, the content of the register represents the index of the VOQ containing the selected unicast cell.

It is important to note that as soon as the output of the traffic priority block is computed, the following update process takes place. If $O_b = 1$, then we know that a multicast cell will be scheduled and the content of the VOQs will not need to change (VOQ ENB set to 0). Therefore, the content of the MQF must be updated (ENB $= 1$) with a new value and this depends on the content of the FR vector (the lower shaded area of Fig. 19). The bits of the FR vector are ORed, and if the result is 1, then the input Mux (see lower left side of Fig. 19) will forward the content of the FR vector to the MQF vector as its new content. Otherwise, the result is 0 meaning the whole multicast cell was completely scheduled, and therefore, a new multicast cell fan-out will be forwarded to the MQF vector. If, however, $O_b = 0$, meaning a unicast cell is chosen, the VOQ vector will be updated while the MQF remains unchanged.

As for the output arbiter, as mentioned previously, it consists of a round robin scheduling mechanism based on a priority encoder. In our design, we employed the MPE design proposed in [28]. It has been segmented into three cycles. We employed the Xilinx Virtex IV platform and implemented our algorithm. The target device of our design was the Xilinx Virtex IV FX family and the results are obtained after place and route. Table 2 depicts the area, in number of slices, and delay, in nanoseconds, results of our design. The input arbiter has a clock cycle time of 2.8 ns and was segmented into seven cycles, resulting in a delay of 19.6 ns. Assuming fixed-sized ATM-like cells of 53 bytes each and a 20 Gbps line speed, the arrival rate would be one cell every 21.2 ns. This is enough for the input scheduler to perform its arbitration (19.6 ns). The critical path of the design is the MPE block. The output arbiter has been segmented into three cycles of 3.4 ns each. The area results are 232 slices for the input arbiter and 107 for the output arbiter.

TABLE 2
Hardware Implementation Results

Module	Area (slices)	Delay (ns)
Input Scheduler (MURS)	232	19.6
Output Scheduler (OS)	107	10.2

7 Conclusion

Combined Input and Crossbar Queued (CICQ) switches have been known to outperform IQ switches due to the simplicity of their scheduling. The problem of integrating unicast and multicast traffic scheduling has been studied for IQ switches only. In this paper, we proposed a CICQ switching architecture able to support both traffic types along with an efficient cell placement scheme. We presented a simple set of integrated scheduling algorithms, named MURS, that can schedule concurrent unicast and multicast traffic flows. In particular, the MURS_mix algorithm has been shown to exhibit very good performance and outperform previous algorithms. Simulation results suggested that a profitable trade-off between the number of input multicast queues and the size of the internal buffers is possible. We presented a hardware implementation of the algorithm for a 16×16 buffered crossbar switch using the Xilinx reconfigurable logic platform. The implementation results suggested that MURS can sustain a 20 Gbps line rate, reaching an aggregate switching bandwidth of 320 Gbps for our target switching system.

4. MURS has been segmented into seven clock cycles, which equals a scheduling cycle.

5. Inverting P_b every scheduling cycle results in MURS_mix. Setting P_b to always 1 results in MURS_mf being implemented and when it is set to always 0, it results in MURS_uf.

References

[1] N. McKeown, "Scheduling Algorithms for Input-Queued Cell Switches," PhD dissertation, Univ. of California at Berkeley, May 1995.

[2] M. Karol, M. Hluchyj, and S. Morgan, "Input Versus Output Queuing on a Space-Division Packet Switch," *IEEE Trans. Comm.*, vol. 35, no. 9, pp. 1337-1356, Dec. 1987.
[3] N. McKeown, "A Fast Switched Backplane for a Gigabit Switched Router," *Business Comm. Rev.*, vol. 27, no. 12, pp. 1-17, 1997.
[4] M. Andrews, S. Khanna, and K. Kumaran, "Integrated Scheduling of Unicast and Multicast Traffic in an Input-Queued Switch," *Proc. IEEE INFOCOM '99*, pp. 1144-1151, 1999.
[5] M. Song and W. Zhu, "Integrated Queuing and Scheduling for Unicast and Multicast Traffic in Input-Queued Packet Switches," *Proc. IASTED Int'l Conf. Comm. and Computer Networks (CCN '04)*, Nov. 2004.
[6] M. Nabeshima, "Performance Evaluation of Combined Input- and Crosspoint-Queued Switch," *Proc. IEICE Trans. Comm.*, vol. B83-B, no. 3, pp. 737-741, Mar. 2000.
[7] R. Rojas-Cessa, Z.J.E. Oki, and H.J. Chao, "CIXB-1: Combined Input One-Cell-Crosspoint Buffered Switch," *Proc. IEEE Workshop High Performance Switching and Routing (HPSR)*, pp. 324-329, 2001.
[8] K. Yoshigoe and K.J. Christensen, "A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar," *Proc. IEEE Workshop High-Performance Switching and Routing*, pp. 271-275, 2001.
[9] T. Javadi, R. Magill, and T. Hrabik, "A High-Throughput Algorithm for Buffered Crossbar Switch Fabric," *Proc. IEEE Int'l Conf. Comm. (ICC '01)*, pp. 1581-1591, June 2001.
[10] L. Mhamdi and M. Hamdi, "MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches," *IEEE Comm. Letters*, vol. 7, no. 9, pp. 451-453, Sept. 2003.
[11] X. Zhang and L.N. Bhuyan, "An Efficient Scheduling Algorithm for Combined Input-Crosspoint-Queued (CICQ) Switches," *Proc. IEEE GLOBECOM '04*, pp. 1168-1173, Nov. 2004.
[12] X. Zhang, S.R. Mohanty, and L.N. Bhuyan, "Adaptive Max-Min Fair Scheduling in Buffered Crossbar Switches Without Speedup," *Proc. IEEE INFOCOM '07*, pp. 454-462, May 2007.
[13] L. Mhamdi and M. Hamdi, "Scheduling Multicast Traffic in Internally Buffered Crossbar Switches," *Proc. IEEE Int'l Conf. Comm. (ICC '04)*, pp. 1103-1107, June 2004.
[14] S. Sun, S. He, Y. Zheng, and W. Gao, "Multicast Scheduling in Buffered Crossbar Switches with Multiple Input Queues," *Proc. IEEE Workshop on High Performance Switching and Routing (HPSR '05)*, pp. 73-77, May 2005.
[15] A. Mekkittikul, "Scheduling Non-Uniform Traffic in High Speed Packet Switches and Routers," PhD dissertation, Stanford Univ., Nov. 1998.
[16] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High Speed Switch Scheduling for Local Area Networks," *ACM Trans. Computer Systems*, pp. 319-352, 1993.
[17] N. McKeown, "iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE Trans. Networking*, vol. 7, no. 2, pp. 188-201, Apr. 1999.
[18] M.A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Optimal Multicast Scheduling in Input-Queued Switches," *Proc. IEEE Int'l Conf. Comm. (ICC)*, 2001.
[19] B. Prabhakar, N. McKeown, and R. Ahuja, "Multicast Scheduling for Input-Queued Switches," *IEEE J. Selected Areas in Comm.*, vol. 15, no. 5, pp. 855-866, June 1997.
[20] A. Bianco, P. Giaccone, E. Leonardi, F. Neri, and C. Piglione, "On the Number of Input Queues to Efficiently Support Multicast Traffic in Input Queued Switches," *Proc. IEEE Workshop on High Performance Switching and Routing (HPSR '03)*, pp. 111-116, June 2003.
[21] S. Gupta and A. Aziz, "Multicast Scheduling for Switches with Multiple Input-Queues," *Proc. Ann. IEEE Symp. High-Performance Interconnects (Hot Interconnects)*, pp. 28-33, 2002.
[22] C. Minkenberg, "Integrating Unicast and Multicast Traffic Scheduling in a Combined Input- and Output-Queued Packet-Switching System," *Proc. Int'l Conf. Computer Comm. and Networks (ICCCN '00)*, pp. 127-234, 2000.
[23] D. Pan and Y. Yang, "FIFO-Based Multicast Scheduling Algorithm for Virtual Output Queued Packet Switches," *IEEE Trans. Computers*, vol. 54, no. 10, pp. 1283-1297, Oct. 2005.
[24] L. Mhamdi, M. Hamdi, C. Kachris, S. Wong, and S. Vassiliadis, "High-Performance Switching Based on Buffered Crossbar Fabrics," *Computer Networks*, vol. 50, no. 13, pp. 2271-2285, Sept. 2006.
[25] Xilinx, Inc., "Virtex-4 Family Overview," http://www.xilinx.com, Mar. 2005.
[26] L. Mhamdi, G.N. Gaydadjiev, and S. Vassiliadis, "Efficient Multicast Support in High-Speed Packet Switches," *J. Networks*, vol. 2, no. 3, pp. 28-35, June 2007.
[27] J.D.C. Little, "A Proof of the Queuing Formula $L = \lambda W$," *Operations Research*, vol. 9, pp. 383-387, 1961.
[28] K. Yoshigoe, K. Christensen, and A. Jacob, "The RR/RR CICQ Switch: Hardware Design for 10-Gbps Link Speed," *Proc. IEEE Int'l Performance, Computing, and Comm. Conf.*, pp. 481-485, Apr. 2003.

Lotfi Mhamdi (S'03-M'07) received the MPhil degree in computer science from the Hong Kong University of Science and Technology (HKUST), in 2002, and the PhD degree in computer engineering from Delft University of Technology (TU Delft), The Netherlands, in 2007. He is currently with the Computer Engineering Laboratory, TU Delft. His research work spans the area of high-speed networks, including the design, analysis, scheduling, and management of high-performance switches and Internet routers. He served as a technical program committee member in various conferences, including the IEEE International Conference on Communications (ICC), the IEEE Workshop on High Performance Switching and Routing (HPSR), and the ACM/IEEE International Symposium on Networks-on-Chip (NoCS). He served as the publicity chair of the International Conference on Design and Technology of Integrated Systems in nanoscale era (DTIS). He is a member of the IEEE.