Integrating Uni- and Multicast Scheduling in Buffered Crossbar Switches

Lotfi Mhamdi, Stamatis Vassiliadis
Computer Engineering Lab., TU Delft, The Netherlands
E-mail: {lotfi, stamatis}@ce.et.tudelft.nl

***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 while keeping simple hardware requirements. Simulation results suggested that we can trade the size of the internal buffers for the number of input multicast queues.**

***Index Terms*— Integrated scheduling, Buffered crossbar fabric.**

I. Introduction

The growing number of newly emerging applications such as teleconferencing, distance learning, IPTV etc. on the Internet has resulted in an increasing proportion of multicast traffic. In addition to point-to-point (unicast) communications, a network node (high speed IP routers and ATM switches) is also required to deal with point-to-multipoint (multicast) communications and the combination of the two. Contrarily 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 queueing structure has also been a combination of unicast queuing structure and multicast queuing structure. The widely used unicast queuing structure is the virtual output queueing (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 *fanout 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 [3] [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 to the nature of the crossbar fabric switching architecture.

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

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]. However, fewer results have appeared for multicast scheduling in CICQ switches [11] [12]. 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 not been addressed. In this paper, we fill this gap and propose the followings:

- An integrated CICQ switching architecture that supports concurrent unicast and multicast flows. The proposed architecture, 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. An efficient cell assignment scheme is proposed to place multicast cells in the k queues.
- A simple round robin scheduling algorithm, called Multicast and Unicast Round robin Scheduling (MURS), that is capable of arbitrating both traffic types simultaneously. MURS was shown, through simulation, to achieve high performance and outperform alternative algorithms. Simulation results showed that we can trade the size of the internal buffers for the number of input multicast queues.

The remainder of this article is organized as follows: Section II presents background knowledge and related work.

In Section III, we introduce the integrated CICQ switching architecture. We discuss the multicast queues management and propose an efficient multicast cell assignment scheme. We then introduce the proposed MURS algorithm, along with two variations: one for unicast priority scheduling and the other for multicast priority scheduling. Section IV presents the performance study of our algorithms with a comparison to existing schemes. Finally, Section V concludes the paper.

II. 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 ports count is as important as challenging.

A. 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 over all 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 [13], however they are too complex to run at high speed. As a result, many practical iterative algorithms have been proposed [14] [15].

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 *fanout set*. For an $N \times N$ switch, the fanout of a cell can range from one to $2^N - 1$. Because of the importance of the input queuing structure, different queueing strategies were proposed for multicast traffic. The optimal solution is where $2^N - 1$ different queues are maintained at each input [16]. 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 [17]. 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 fanout configurations, other researchers propose to allocate a small number, k, of multicast queues per input [18] [19]. Because k is smaller than the fanout 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 [18] [19].

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 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 weighted 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 [11]. 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 [12]. Because a cell placement scheme is needed to enqueue the incoming data, the authors presented some cell placement schemes, such as CRRA and BRRA [12]. 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.

B. 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 [20], 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. In more recent work [5], the input queueing 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. These algorithms perform many iterations in order to achieve good performance, limiting their scalability in port counts 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 [21]. 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 article, we propose an integrated CICQ based switching architecture,

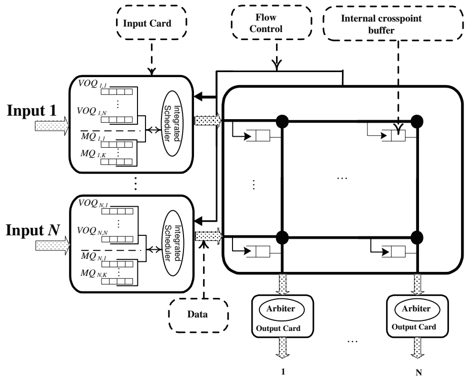


Fig. 1. The Integrated CICQ Switching Architecture

along with its appropriate scheduling, that supports both unicast and multicast traffic simultaneously.

III. 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 II-B).

A. Reference Architecture

The proposed switch model consists of an $N \times N$ buffered crossbar fabric. 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 fanout set, Φ, where $\{\Phi \mid 1 \leq \Phi \leq N\}$. To cover all possible fanout sets would require $2^N - 1$ multicast queues, one per fanout 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 $\{k \mid 1 \leq k \ll 2^N - 1\}$. At each input, multicast queues are denoted by $MQ_{i,j}$ where $\{(i,j) \mid 1 \leq i \leq N; 1 \leq j \leq k\}$ A *cell assignment policy* has to take place in order to map incoming cells to the multicast queues. This is discussed in Section III-B.

The buffered crossbar fabric chip contains N^2 distributed cross-point buffers, denoted 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 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 non full XP. Each output contains an output buffer and an output scheduler. The output scheduler at output j examines the content of $XP_{i,j}$ $\{i \mid 1 \leq i \leq N\}$ and selects one cell to be transferred to the output port. Both input and output scheduling are discussed in Section III-C. A flow control mechanism is implemented to continuously communicate the state of the internal buffers to the input schedulers to prevent XPs overflow.

B. Multicast Cell Assignment

Since the number of MQs, k, maintained at each input is much smaller than the number of all fanout configurations, 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 [19] has pointed out some criteria in designing such a policy: (i) 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. (ii) Cells with the same, or similar, fanout sets should be stored in the same queue, to reduce HoL blocking and prevent out of sequence delivery problem. Many cell assignment schemes existed, such as majority [19], MDQ [18]. While they succeed in meeting some, or all, of the aforementioned criteria, their implementation can prove not cost effective.

Our queuing structure implements a simple and efficient cell assignment scheme. Our scheme works as follows: At every input, i, each incoming multicast cell with fanout set, Φ, is assigned to $MQ_{i,j}$ where $\{j \mid j = \Phi \; mod(k)\}$. In addition to its simplicity, especially if the number of MQs, k, is a power of two, our scheme meets all previously mentioned criteria for efficient cell assignment. To better understand this, let's consider an example. Assume we have an 8×8 switch and 2 multicast queues per input ($k = 2$). At each input, i, the cell assignment scheme will place cells with even fanout sets in $MQ_{i,0}$ and those with odd fanout sets in $MQ_{i,1}$. This way, the heads of the MQs can span large numbers of destinations (i.,e. HoL of $MQ_{i,0} = 6$ and HoL of $MQ_{i,1} = 3$). Moreover, cells with the same fanout sets are ensured to be queued in the same MQ, avoiding the out of sequence problem. Additionally, our scheme is a fair scheme in the sense that is gives equal opportunities to the cells to advance to the head of the queue irrespective of their number of destinations. This is important as there are scheduling algorithms that use the fanout set as the weight for priority scheduling and, unless the cells fanout sets per MQ are diverse, MQ starvation (unfairness) can occur.

C. Integrated Scheduling

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 care has to be taken of the input scheduler at each input. The input scheduler, not only is required to *select* cells to be transmitted to the fabric chip, but also needs to decide *when* to pick a cell *from which* set of queues (VOQs or MQ). This is called the *integration* phase of the scheduler. The selection policy of our scheme is based on round robin because of its fairness and simple hardware implementation. The selection policy is fixed and independent of the integration phase. Each input scheduler maintains two priority pointers: a unicast pointer (UP) and a multicast pointer (MP). If the $VOQs$ (MQ) set of queues is chosen to select a cell from, the round robin pointer will be based on UP (MP). Please note that we consider *fanout splitting* when serving multicast flows [17]. The integration phase is responsible for making the decision of which set of queues to chose from. As the traffic can be unicast, multicast or a mix of the two, 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. The specification of the selection and integration policies are as follows:

- *Select_Queue*(Queue_type , Pointer_type)

```
    {
  . $N$ = number of queues in Queue_type;  $i$ = current input
  . Starting from the Pointer_type's location, select the first eligible
    queue $EQ_{i,j}$ and send its HoL cell[1] to the internal buffer ($XP_{i,j}$).
  . Move Pointer_type to the location $(j+1)\ (mod\ N)$.
    }
```

- Each integration policy corresponds to an input scheduling (IS):
  - MURS_uf /*Always Unicast traffic first (prioritized)*/
  - MURS_mix /*Priority is time slot dependent */
  - MURS_mf /*Always Multicast traffic first (prioritized)*/

```
– MURS_uf    /*Always Unicast traffic first (prioritized)*/
      {
    . Select_Queue($VOQs$ , $UP$);
    . If no queue was selected
    . Select_Queue($MQ$ , $MP$);
      }
– MURS_mix    /*Priority is time slot dependent */
      {
    . If current time slot is even    /*Unicast is served first*/
      Select_Queue($VOQs$ , $UP$);
      If no queue was selected
      Select_Queue($MQ$ , $MP$);
    . Else                            /*Multicast is served first*/
      Select_Queue($MQ$ , $MP$);
      If no queue was selected
      Select_Queue($VOQs$ , $UP$);
      }
– MURS_mf    /*Always Multicast traffic first (prioritized)*/
      {
    . Select_Queue($MQ$ , $MP$);
    . If no queue was selected
    . Select_Queue($VOQs$ , $UP$);
      }
```

Irrespective of the input scheduling algorithm, we used the same output scheduling algorithm. It is a simple round robin scheduling, with the following specification:

- Output Scheduling (OS)

```
  . All outputs share the same output pointer and it is incremented,
    each time slot, by one $mod\ (N)$.
  . Starting from the pointer's index, each output, $j$, selects the first
    non empty internal buffer, $XP_{i,j}$, and sends its HoL cell to the
    output port.
```

The input scheduling, MURS, plays an important role in the overall performance of the system. The three variations we proposed above are different and would have different performance especially in the presence of concurrent unicast and multicast traffic. 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 VOQ 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 because 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. Whereas, if we give 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 greatly scale up. This is exactly what MURS_mf algorithm does, 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 of them tries to monopolize the switch bandwidth to its preferred traffic flows and this is undesirable.

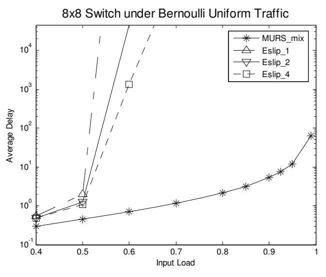
As a compromise between MURS_uf and MURS_mf, we propose MURS_mix. In addition to its fairness in the presence of different traffic types, the MURS_mix algorithm exhibits the same performance as the other two algorithms when the traffic is all unicast or all multicast. The properties of MURS_mix makes it a good candidate for being an optimal integrated scheme because: (i) 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 gives equal chances (even and odd time slots) to heterogenous 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. (ii) MURS_mix requires simple hardware allowing it to run at high speed. (iii) Finally, MURS_mix shows enhanced performance in terms of high throughput and low cell latency by comparison to existing algorithms. This is shown in the following section.

IV. Performance Results

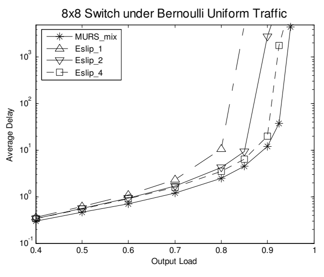
This section presents the simulation study of an $8x8$ CICQ switching system employing the MURS set of algorithms. We compared the performance of MURS_mix to the Eslip algorithm which uses buffer-less crossbar switch [3]. Simulations run for 1 million time slots and statistics are gathered when tenth of the total simulation length has elapsed. We analyzed the delay and throughput performance of the algorithms with different MQ numbers, k. Additionally, we observed the stability of the input queues under different traffic, input queueing and internal buffer size settings.

Incoming traffic is generated according to a uniform i.i.d Bernoulli process and a bursty uniform process, respectively. 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

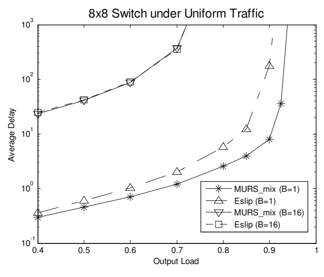
[1]If the cell is multicast, then only copies destined to c outputs are sent, where $\{c \mid c \in \{1, ..., N\}\ and\ XP_{i,c}\ is\ not\ full\}$. Other copies will have to compete in later time slots.



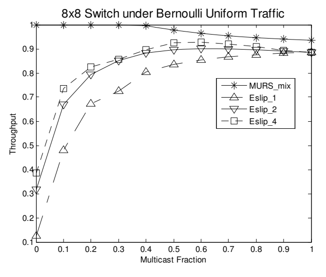
(a)



(b)



(c)



(d)

Fig. 2. Delay Performance of MURS_mix and Eslip with different multicast fractions, f_m. (a): $f_m = 0$, corresponding to unicast traffic only. (b): $f_m = 0.5$, resulting in evenly distributed input traffic over unicast and multicast flows. (c): $f_m = 1$ corresponding to multicast traffic only, with uniform Bernoulli and uniform Bursty arrivals (burst length, B=16). (d): Maximum Throughput as a function of Multicast Fraction (f_m ranging from 0 to 1).

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 fanout set, Φ, of multicast cells is uniformly distributed between 2 and 8 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 Equation (1).

$$\mu = \lambda(f_u + \Phi f_m). \quad (1)$$

In our simulation, we averaged the cells fanout set to be $\Phi = 4$. Following our settings and substituting f_u with f_m, we have:

$$\mu = \lambda(1 + 3f_m). \quad (2)$$

For example, if we set f_m to be 0 in Equation (2), the traffic is all unicast. When we set it to 1, the traffic becomes all 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.

In Fig. 2, we compare the average delay performance of MURS_mix and Eslip. Because Eslip is based only on one multicast queue per input, we used the same settings with MURS_mix ($k = 1$) for fair comparison. Please note that Eslip_i refers to Eslip with i iteration(s). As depicted in Fig. 2, irrespective of the incoming traffic, MURS_mix always achieves far shorter delay than Eslip when the traffic is either all unicast (Fig. 2-(a)) or all multicast (Fig. 2-(c)). When each traffic type shares half the input load ((Fig. 2-(b)) MURS_mix still achieves better performance than Eslip. However, this corresponds to only one setting of a mixed traffic.

Checking all possibilities of mixed traffic requires tuning f_m from 0 to 1 and observe the throughput. To this end, as depicted in Fig. 2-(d), we fixed the output load, μ, to be 100% (fully loaded system) and recorded the throughput of each algorithm as f_m varies from 0 to 1. Again, MURS_mix is always superior to Eslip by an order of magnitude. MURS_mix reaches its lowest performance when $f_m = 1$. This is because the traffic is all multicast and only a single multicast queue is used per input. Please note that, while all our set of algorithms have better 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 an other.

The remainder of the simulation is done for multicast queue set, MQ per input, equal to or bigger than one ($k \geq 1$). Fig. 3 shows the average cell delay performance of each of our algorithms for k = 1, 2 and 4 respectively and evenly

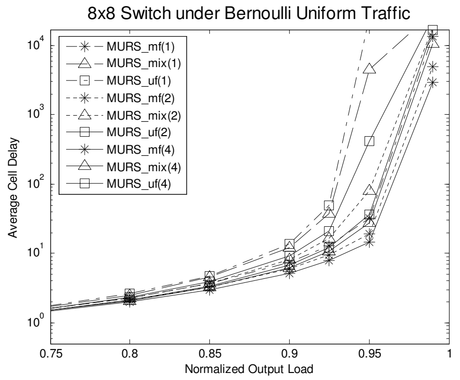


Fig. 3. Average Delay of MURS with Different Numbers of Multicast Queues per Input and Mixed Input Traffic ($f_m = 0.5$).

distributed traffic over unicast and multicast ($f_m = 0.5$). As expected, the MURS_mf scheme has the best delay. This is because it gives priority to multicast flows over unicast flows resulting in more connections per scheduling cycle. This result holds independently of the number of MQ used per input. MURS_uf, however, has the worst delay because it prioritizes unicast over multicast. MURS_mix has a medium average delay because it treats both traffic types with the same priority. On overall MURS_mix is the best due to its fairness.

Due to the importance of the internal buffers in simplifying the scheduling, we simulated our algorithms under different internal buffer sizes. Fig. 4-(a) depicts the average delay performance of the MURS_mix algorithm under three different scenarios. Incoming traffic is either all unicast ($f_m = 0$), or 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 and studied their effect under each traffic scenario. For example, "Q_XP(14)_Ucast" corresponds to the MURS_mix algorithm with 1 multicast queue per input ($k = 1$), 4 cells per internal buffer ($XP = 4$) and incoming traffic consisting of unicast cells only. "Q_XP(41)_Mix" corresponds to MURS_mix with 4 MQs ($k = 4$) per input, 1 cell per XP and a mixed incoming traffic over unicast and multicast flows ($f_m = 0.5$). Simulation results show that, irrespective of the incoming traffic type, the performance of MURS is the same when we decrease the input multicast buffers by 75% at the expense of increasing the size of each internal buffer to accommodate 4 cells instead of 1. This result is the same when we also use MURS_mf, as depicted in Fig. 4-(b).

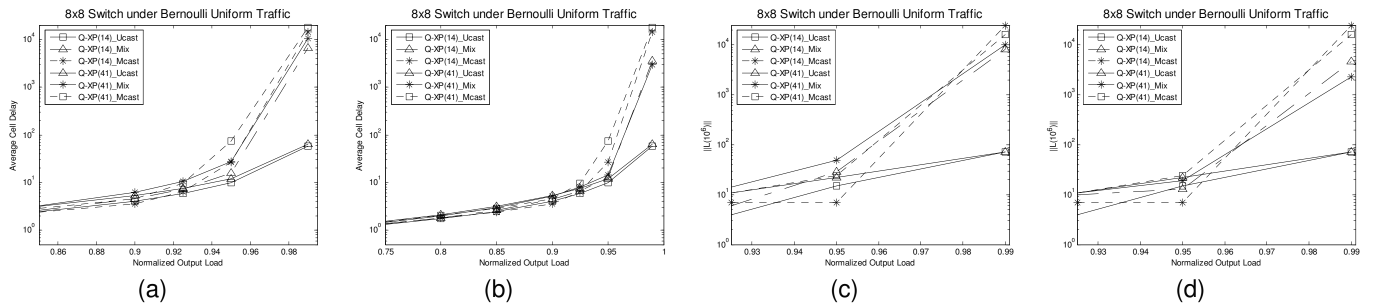


Fig. 4. Delay and Stability Performance with Different Numbers of Input Multicast Queues, Different Internal Buffer Sizes and Different Traffic Scenarios. (a): Average Delay of the MURS_mix Algorithm. (b): Average Delay of the MURS_mf Algorithm. (c): The $l - two$ Norm Vector at Time = 1 million time slots, MURS_mix is employed. (d): The $l - two$ Norm Vector at time = 1 million time slots, MURS_mf is employed.

Because the trade off between the input multicast queues and the internal buffers is not straightforward (on chip memory vs. off chip memory), we studied the stability of the input queues under the same settings as above. We used the $l - two$ norm vector representing the occupancy of all 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 $l - two$ norm vector at time slot n is denoted by $\|L(n)\|$ and defined as follows:

$$\|L(n)\| = \sqrt{\sum_{i=1}^{n}\left(\sum_{j=1}^{n} VOQ_{i,j}(n)^2 + \sum_{l=1}^{k} MQ_{i,l}(n)^2\right)}$$

In addition to representing the occupancy of the input queues, the $l - two$ 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. 4-(c) and (d) respectively, the input queues occupancies remain the same whether we use only 1 multicast queue per input and internal buffer size of 4 cells or 4 multicast queues per input and internal buffer size of 1 cell. This is very important because of the gain we can achieve by this trade off, due the small amount of internal buffers compared to the size (and number) of multicast input queues. By considering that a switch should hold up to $100ms$ worth of packets [22] and if we consider a switch running $10\ Gbps$ port speed, the buffer requirement per port would be $125\ MB$. Assuming $64\ B$ cell size and our 8×8 CICQ switch, every input queue is approximately $10.4\ MB$. In our case, reducing the multicast queues set by 75% (using just 1 MQ per input instead of 4) corresponds to increasing the size of each internal buffer to hold 4 cells. This results in saving $250MB$ of off chip memory, at the expense of adding approximately $12.3KB$ of internal buffers while keeping the same overall input buffer requirement as well as achieving the same cell delay. This translates into a considerable saving while maintaining the same high performance of the system.

V. 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. 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.

References

[1] McKeown. N., *Scheduling algorithms for input-queued cell switches*, Ph.D. thesis, University 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. on Commun.*, vol. 35, no. 09, pp. 1337–1356, Dec. 1987.

[3] N. McKeown, "A fast switched backplane for a gigabit switched router," *Business Commun. Rev.*, vol. 27, no. 12, 1997.

[4] M. Andrews, S. Khanna, and K Kumaran, "Integrated Scheduling of Unicast and Multicast Traffic in an Input-Queued Switch," *IEEE INFOCOM*, pp. 1144–1151, 1999.

[5] M. Song and W. Zhu, "Integrated Queuing and Scheduling for Unicast and Multicast Traffic in Input-Queued Packet Switches," *IASTED Int. Conference on Communication and Computer Networks (CCN 2004)*, Nov. 2004.

[6] M. Nabeshima, "Performance Evaluation of Combined Input-and Crosspoint-Queued Switch," *IEICE Trans. On Commun.*, vol. B83-B, no. 3, March. 2000.

[7] R. Rojas-Cessa, Z. Jing E. Oki, and H. J. Chao, "CIXB-1: Combined Input One-Cell-Crosspoint Buffered Switch," *IEEE HPSR*, pp. 324–329, 2001.

[8] K. Yoshigoe and K. J. Christensen, "A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar," *IEEE orkshop on 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," *IEEE ICC*, pp. 1581–1591, June 2001.

[10] L. Mhamdi and M. Hamdi, "MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches," *IEEE Communications Letters*, vol. 07, no. 09, pp. 451–453, Spet. 2003.

[11] L. Mhamdi and M. Hamdi, "Scheduling Multicast Traffic in Internally Buffered Crossbar Switches," *IEEE ICC*, pp. 1103–1107, June 2004.

[12] S. Sun and S. He and Y. Zheng and W. Gao, "Multicast Scheduling in Buffered Crossbar Switches with Multiple Input Queues," *IEEE HPSR*, pp. 73–77, May 2005.

[13] A. Mekkittikul, *Scheduling Non-Uniform Traffic In High Speed Packet Switches and Routers*, Ph.D. thesis, Stanford University, Nov 1998.

[14] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High Speed Switch Scheduling for Local Area Networks," *ACM Transactions on Computer Systems*, pp. 319–352, 1993.

[15] N. McKeown, "iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE Trans. On Networking*, vol. 07, no. 02, pp. 188–201, Apr. 1999.

[16] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Optimal Multicast Scheduling in Input-Queued Switches," *IEEE ICC*, 2001.

[17] B. Prabhakar, N. McKeown, and R. Ahuja, "Multicast Scheduling for Input-Queued Switches," *IEEE JSAC*, pp. 2021–2027, June 1997.

[18] A. Bianco and E.and Neri F.and Piglione C. Giaccone, P.and Leonardi, "On the Number of Input Queues to Efficiently Support Multicast Traffic in Input Queued Switches," *IEEE HPSR*, pp. 111–116, June 2003.

[19] S. Gupta and A. Aziz, "Multicast Scheduling for Switches with Multiple Input-Queues," *Proc. of Hot Interconnects*, pp. 28–33, 2002.

[20] C. Minkenberg, "Integrating Unicast and Multicast Traffic Scheduling in A Combined Input- and Output-Queued Packet-Switching System," *ICCCN*, pp. 127–234, 2000.

[21] L. Mhamdi, M. Hamdi, C. Kachris, S. Wong, and S. Vassiliadis, "High-Performance Switching Based on Buffered Crossbar Fabrics," *Computer Networks Journal, To appear in 2006*.

[22] H. J. Chao, "Next Generation Routers," *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1518–1558, Sept. 2002.