Multi-chip multicast schedulers in input queued switches

# Multi-chip Multicast Schedulers in Input-Queued Switches

Andrea Bianco [#], Alessandra Scicchitano [*]

[#]*Dip. di Elettronica, Politecnico di Torino*
*Corso Duca degli Abruzzi 24, 10129 Torino, Italy*
andrea.bianco@polito.it

[*]*Dip. di Elettronica, Informatica e Sistemistica, Università della Calabria*
*Via P.Bucci, 87036 Arcavatava di Rende, Cosenza, Italy*
ascicchitano@deis.unical.it

***Abstract*— Multi-chip scheduler implementation in IQ switches are suited to reduce the hardware complexity in very large, high-speed, switches. However, this implies introducing a RTTs (Round Trip Time) among input and output selectors used to determine a matching due to inter-chip latency. This delay requires modifications to scheduling algorithms to allow a fully distributed implementation while keeping good performance. We propose a novel multicast scheduler, named IMRR, an extension of a previously proposed multicast scheduling algorithm named mRRM, making it suitable to a multi-chip implementation, and examine its performance by simulation.**

## I. Introduction and Related Work

Despite the fact that synchronous slotted IQ (Input-Queued) switches have been proposed as an innovative architecture for high-speed switches many years ago, the interest of the research community in this type of architectures is still significant. Indeed, IQ switches are suited for several application domains, such as traditional routers/switches, SANs (Storage Area Networks), and HPC (High-Performance Computing) interconnects; in most of these application domains, a large number of ports and high line rates are dominant. Although multicast traffic has not yet became a significant portion of the current Internet traffic, its support is anyhow fundamental in routers/switches. Indeed, broadcast traffic is important in Ethernet switches both to support applications that rely on LAN broadcast capability, e.g., ARP, as well as to internally distribute forwarding tables; in SANs, multicast enables data replication, possibly on physically separated site, to improve reliability.

To obtain performance close to output queued switches, IQ switches rely on a VOQ (Virtual Output Queued) architecture at inputs for unicast traffic, as shown in Fig.1: in an $N \times N$ switch, data, named cells in this paper since we consider synchronous slotted operation, are stored at each input port in $N$ separate queues, depending on cell destination. The optimal queue architecture for multicast traffic [1], that completely avoids HoL blocking for multicast flows, requires a number of queues which grows exponentially as $2^N$ with the number $N$ of inputs/outputs, and a complex packet re-enqueuing scheme. Being this solution unpractical and due to the fact that currently multicast traffic is a small portion of data traffic, most researchers have proposed the use of a single FIFO queue for multicast traffic [2], [3]. Other studies propose an architecture with $K < N$ queues and show some performance benefits due to the reduced HoL blocking [4], [5], [6].

The most challenging problem in IQ architectures is the definition of a proper scheduling algorithm: to transfer fixed-size cells from VOQs to output ports, at every time slot, a matching between inputs and outputs must be determined to avoid output and input contention, since no buffers are available at output ports and no internal speedup is available in the switching fabric and in memory access speed. The task of the centralized scheduler is to collect information from input queues, to determine a matching and to configure the switching fabric in each time slot to connect input VOQs to outputs. Increasing high rates imply shorter time slots, thus requiring simple scheduling algorithms, especially in switches with a large number of ports. Several heuristic algorithms have been proposed for unicast [7], [8], [9], [10], [11] and for multicast traffic [2], [3], [4], [5], [6].

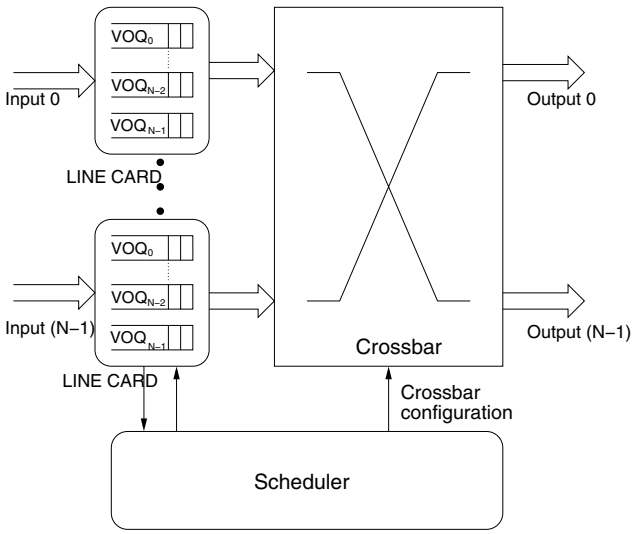


Fig. 1. IQ architecture with a centralized scheduler and VOQ architecture at inputs for unicast traffic.

A very popular solution to determine a heuristic matching is to devise parallel, iterative matching algorithms based on

a three-phase (request-grant-accept) [9], [10] or two-phase (request-grant) [2], [3], [11] scheme. In three-phase schemes, in the request phase, inputs issue requests to outputs. In the grant phase, outputs solve request contentions independently, by choosing a single request to grant. Finally, inputs solve contentions independently by accepting one among multiple received grants. In two phase schemes, each input issues a single request only to a given output. As such, no accept phase is needed, since no grant contention may arise at inputs. However, a proper request must be chosen at each input and request contention must be solved as in three-phase schemes. Iterations are in general fundamental to improve the matching and obtain good performance, especially for unicast traffic. As such, the three-step or two-step procedure is repeated several times within a time slot.

Request and grant contentions in three-phase schemes, as well as the choice of the request to send and request contentions in two-phase schemes, are typically solved using $N$ input and $N$ output selectors (labeled IS and OS respectively in the remainder of the paper), exploiting a round-robin mechanism based on pointers kept at inputs and outputs: the input (output) selector chooses the first eligible output (input) in a round robin fashion, starting from the position indicated by the pointer.

The centralized single-chip implementation of scheduling algorithms is largely dominant; however, scalability problems may arise for very large high-speed switches [13]. When looking at multi-chip implementations, device separation implies that decision taken by input/output selectors belonging to different devices are known only after an inter-chip latency, named RTT (Round Trip Time) in the remainder of the paper. RTTs may be significant with respect to the time slot. Indeed, at high speed the time slot is rather short (13ns at 40Gbit/s for a 64bytes packet) if compared with inter-chip communication latencies which include propagation delays, data serialization and pin sharing which may be required to overcome the I/O pin count limit. As such, as shown in Fig. 2, information critical i) to determine the matching, ii) to update the pointers and iii) to issue new requests, is delayed by the inter-chip latency, causing performance degradation.

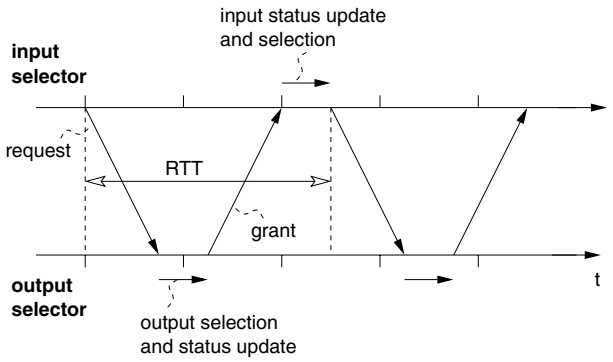


Fig. 2. Round trip time between input and output selectors.

Scheduling algorithms to deal with the inter-chip latency issue were previously proposed for unicast traffic [13], [14]. To the best of our knowledge, no studies were proposed to solve the scheduling problem for multicast traffic. Although one possible way to support multicast traffic is to replicate multicast cells at inputs and treat them like unicast cells, we disregard this approach in this paper, if not for comparison purpose. Indeed, a speed increase at input ports would be required to support data replication, and a poor bandwidth utilization of the switching fabric [1] can be obtained even in very simple scenarios; indeed, a broadcast flow at line speed incoming in a single input cannot be supported efficiently due to the no-speedup requirement of IQ switches, which limits the throughput to $1/N$ in an $N \times N$ IQ switch. The interesting multi-hop approach proposed in [12], which permits to distribute input load to several inputs and obtain high throughput efficiency, has a major drawback in terms of large delays at low-medium loads, a problem which is further exacerbated by the RTT induced by the inter chip latency penalty in multi-chip implementations.

We propose extensions/modifications/improvements to previously proposed multicast scheduling algorithms to cope with performance degradation induced by RTTs, without increasing too much the scheduler complexity. More precisely, we introduce a novel scheduler, named IMRR (Improved Multicast Round Robin), which is an extension of a previously proposed scheduler named mRRM (Multicast Round Robin) [2] We show by simulation that IMRR provides good performance for uniform multicast traffic and for multicast traffic pattern known to be difficult to schedule.

The remainder of the paper is organized as follows: Section II states the problems, challenges and solutions for multi-chip scheduler implementation. Section III describes previously proposed scheduling algorithms and the newly proposed IMRR scheme. Section IV presents performance results obtained by simulation. Finally, Section V ends the paper and suggests possible future research directions.

## II. Multi-chip implementation

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

Performance degradation and loss of fairness were already shown to be a possible problem in this multi-chip scenario.

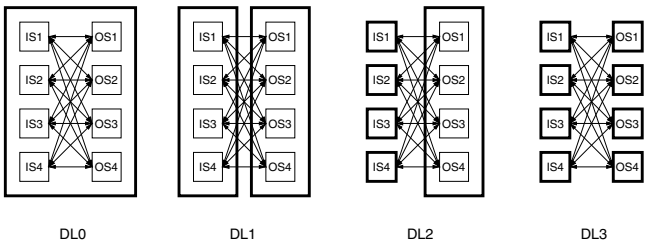


Fig. 3. Distribution levels of a centralized multi-chip scheduler: DL0 is a monolithic, single-chip implementation, DL3 is a fully distributed multi-chip implementation.

Different levels of selectors distribution could be envisioned, which yield to a different number of physically separated devices, as shown in Fig. 3. The first obvious solution to reduce the scheduler complexity, labeled distribution level DL1, is to implement the scheduler in two separate devices, each containing respectively $N$ input and $N$ output selectors. This allows to roughly divide by two the scheduler hardware complexity. Another extreme, labeled distribution level DL3, is to distribute the selectors over $2N$ separate devices, each device implementing one selector. This is referred to as fully distributed solution, which permits a hardware complexity reduction by a factor of $N$. As an intermediate step, in [13] a further possible solution is proposed: the $N$ input selectors are physically separated in $N$ devices, whereas all $N$ output selectors are implemented in a single device. This solution has a major drawback: it reduces the hardware complexity by a factor of two only. However, having the output selectors in a single device permits coordination among output selectors with no delay. This permits to implement schedulers with more iterations in a single cell time, thus preserving good performance for increasing RTTs; however, this limits scheduler scalability. As such, we focus on schedulers suited to a fully distributed multi-chip scenario (distribution level DL3).

## III. Multicast scheduling algorithms

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

We focus on two previously proposed multicast scheduling algorithms, selected due to their ease of implementation and good performance: WBA (Weight-Based Algorithm) [3] and mRRM [2]. Both rely on a single FIFO queue at each input for multicast. We briefly remind the WBA and mRRM scheduler behavior referring to a scenario where RTT=0 for simplicity. Both schedulers are based on a two-phase request-grant algorithm.

WBA assigns weights to input cells based on their age and fanout at the beginning of every cell time. Once weights are assigned, each OS chooses the heaviest input among those subscribing to it. More precisely, at the beginning of every cell time, each IS computes the weight of the new multicast cell/residue at its HOL based on the age of the cell (the older, the heavier), and the fanout of the cell (the larger, the lighter). Then, each IS sends this weight to all outputs that the cell/residue at its HOL wishes to reach. Each OS grants to the input with the highest weight, independently of other outputs, breaking ties with a random choice. Note that a positive weight should be given to age to avoid input starvation. However, to maximize throughput, fan-out are weighted negatively. Several weight assignments algorithm can be adopted (see [3] for details). In this paper we use the weight definition chosen in the simulator available on the Web site at http://klamath.stanford.edu/tools/SIM/: The cell weight is equal to the number of inputs minus the cell fanout plus the cell age.

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

### A. Improved Multicast Round Robin

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

Further extensions can be envisioned and are studied later when presenting simulation results. Indeed, the single FIFO scheme when RTT=0 and its direct extension to RTT+1 FIFO queues when RTT $> 0$ does not solve the issue of HoL blocking for multicast flows. As such, in some experiments, we introduce a limited number of $k = 2$ FIFO queues at each input and $k = 2 \times (\text{RTT}+1)$ for RTT$> 0$. In this case, whereas

output selectors operate as in the basic scheme, input selectors choose one among the $k$ queues according to queue weights for WBA and to a round-robin scheme for mRRM.

The introduction of more queues at inputs, either to deal with RTTs or to reduce the HoL blocking, introduces the issue of multicast flow assignment to queues. Although several possible assignment strategies were studied in the past [5], [6], in this paper we simply adopt a packet-by-packet load balancing scheme among available queues. We are aware that this may introduce out-of-sequence delivery, but we prefer to avoid studying assignment scheme in this initial work and concentrate on the issue related to dealing with RTTs.

Let us now describe the IMRR (Improved Multicast Round Robin) scheduler, initially disregarding issues related to RTT for simplicity when a single FIFO queue is available. The same concept of preferential input is kept as in the mRRM algorithm: at each time slot, all output selectors keep a single pointer to a preferential input. However, the pointer update rule is stateless; regardless of the granted cell, at each time slot, the pointer is increased (modulo $N$) according to a round robin scheme. This is fundamental to guarantee that the scheduler can run in the fully distributed case, since no coordination among output scheduler is required. Input selectors issue a request containing a weight equal to the fanout of the selected cell. Output selectors choose the request from the preferential input, if any, otherwise they grant the request corresponding to the smallest fanout.

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

## IV. Performance results

We show performance results based on simulation runs exploiting a proprietary simulation environment developed in C language. Statistical significance of the results are assessed by running experiments with an accuracy of 1% under a confidence interval of 95%. We compare IMRR with the previously presented extensions of mRRM and WBA, with a random (RND) scheduler, and with the multi-copy approach, where multicast cells are replicated at inputs and treated as unicast cells. In the multi-copy approach, the unicast SRR scheduler [14] is run, given its good performance and adaptability to the fully distributed multi-chip scenario.

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

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

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

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

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

Despite its good performance in the uniform scenario, recall that the multi-copy approach is reported only as a reference case; indeed, it cannot be used in practice, since, for example, it is unable to sustain a broadcast flow overloading a single input. Indeed, when studying the performance of the switch under gathered traffic in Fig.5, the throughput limitation of the multi-copy approach becomes evident. In this scenario,

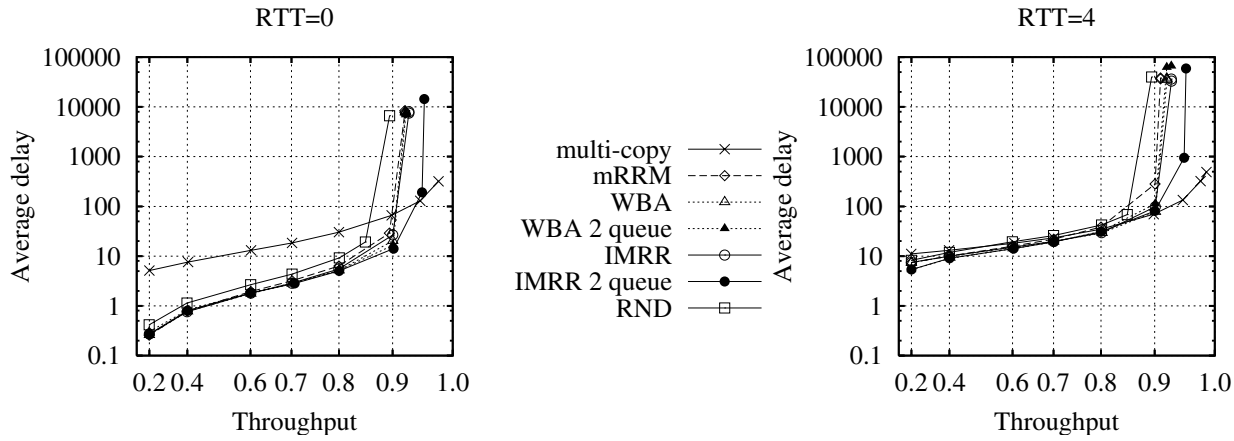


Fig. 4. Performance comparison under uniform Bernoulli cell arrivals.

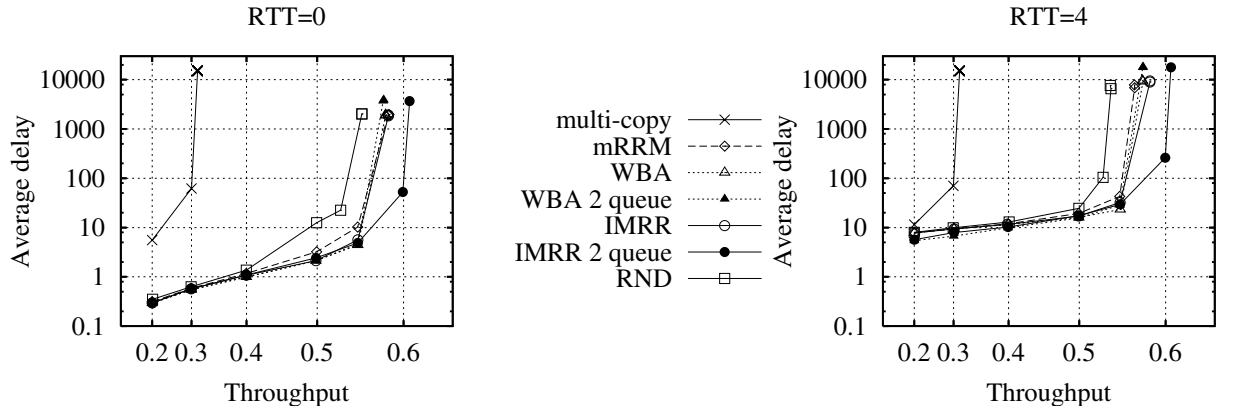


Fig. 5. Performance comparison under gathered traffic with cell arrivals.

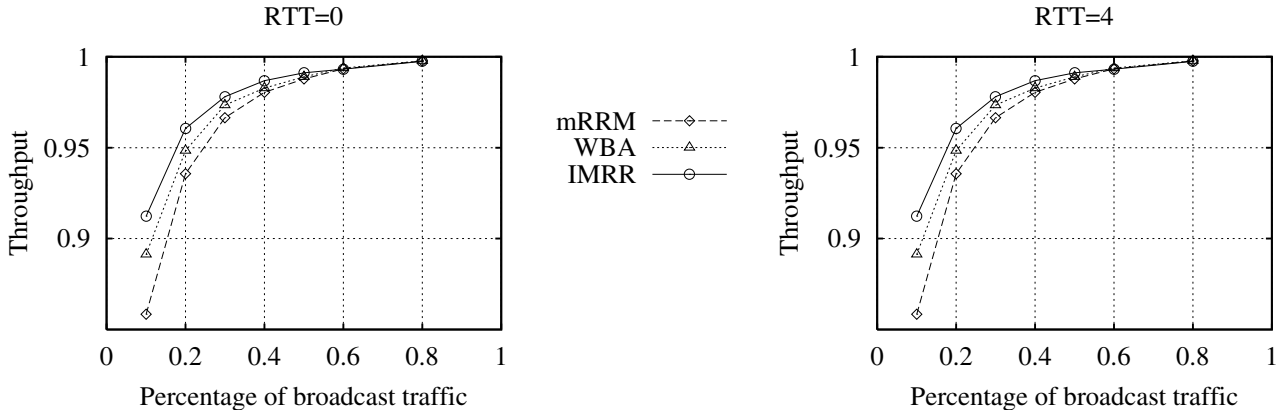


Fig. 6. Performance comparison for uniform traffic and a variable percentage of unicast and broadcast traffic only.

the proposed IMRR approach provides improved throughput especially when using $k = 2$ FIFO queues per RTT. The WBA scheduler, despite its higher complexity, is unable to exploit $k = 2$ FIFO queues to obtain performance benefits. The RND scheduler still presents the worse performance among the multicast schedulers (excluding the multi-copy approach).

IMRR shows good throughput performance also when considering the traffic scenario with unicast and broadcast

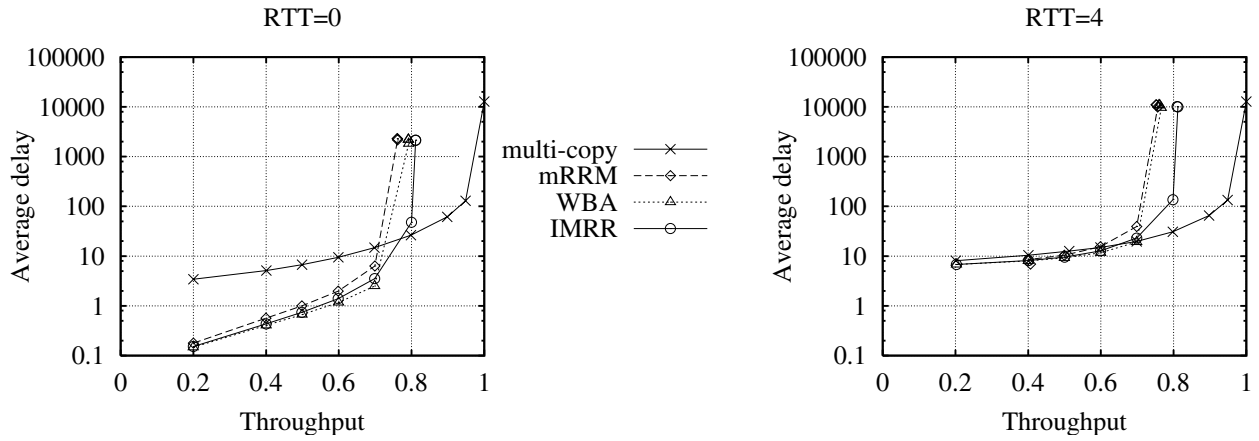


Fig. 7. Performance comparison for uniform traffic: 50% unicast and 50% multicast traffic.

traffic only, reported in Fig.6. When broadcast traffic becomes dominant, all algorithms show similar, good, performance. However, when unicast traffic becomes more significant, differences among algorithms become evident even when using a single FIFO queue per RTT, and IMRR outperforms the other algorithms.

In Fig.7 we report delays as a function of the switch throughput when considering the scenario in which 50% of the traffic is uniform multicast and 50% is uniform unicast. Also in this case IMRR provides throughput benefits with respect to WBA and mRRM both when RTT=0 and RTT $> 0$.

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

## V. Conclusions

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

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

## References

[1] M.Ajmone Marsan, A.Bianco, P.Giaccone, E.Leonardi, F.Neri, "Multicast traffic in input-queued switches: Optimal scheduling and maximum throughput", IEEE/ACM Transactions on Networking, Vol.3, No.11, pp.465-477, June 2003

[2] N.McKeown, B.Prabhakar, "Scheduling Multicast Cells in an Input-Queued Switch", IEEE INFOCOM '96, San Francisco, CA,USA, March 1996

[3] B.Prabhakar, N.McKeown, R.Ahuja "Multicast Scheduling for Input-Queued Switches", IEEE JSAC (Journal on Selected Areas in Communications), Vol.15, No.5, pp.855-866, June 1997

[4] S.Gupta, A.Aziz, "Multicast scheduling for switches with multiple queues", Hot Interconnects02, Stanford, CA, August 2002

[5] A.Bianco, P.Giaccone, E.Leonardi, F.Neri, C.Piglione "On the Number of Input Queues to Efficiently Support Multicast Traffic in Input Queued Switches", HPSR'03, Torino, Italy, June 2003

[6] A.Bianco, P.Giaccone, C.Piglione, S.Sessa "Practical Algorithms for Multicast Support in Input Queued Switches", HPSR'06, Poznan, Poland, June 2006

[7] M.Ajmone Marsan, A.Bianco, E.Leonardi, L.Milia, "RPA: A Flexible Scheduling Algorithm for Input Buffered Switches", IEEE Transactions on Communications, Vol.47, No.12, pp.1921-1933, Dec. 1999

[8] A.Smiljanic, R.Fan, G.Ramamurthy, "RRGS-round robin greedy scheduling for electronic/optical terabit switches", IEEE GLOBECOM'99, Rio De Janeiro, Brazil, Dec. 1999

[9] N.McKeown, "The iSLIP scheduling algorithm for input-queued switches", IEEE/ACM Transactions on Networking, v.7, n.2, pp.188-201, Apr.1999

[10] D.Serpanos, P.Antoniadis, "FIRM: A class of distributed scheduling algorithms for high-speed ATM switches with multiple input queues", IEEE INFOCOM'00, Tel Aviv, Israel, Mar 2000

[11] Y.Li, S.Panwar, H.Chao, "On the performance of a dual round-robin switch", IEEE INFOCOM'01, Anchorage, AK, USA, Apr 2001

[12] A.Smiljanic, "Scheduling of multicast traffic in high-capacity packet switches", IEEE Communications Magazine, Vol.40, No.11, pp.72-77, Nov 2002

[13] C.Minkenberg, F.Abel, E.Schiattarella, "Distributed crossbar schedulers", HPSR'06, Poznan, Poland, June 2006

[14] A.Scicchitano, A.Bianco, P.Giaccone, E.Leonardi, E.Schiattarella, "Distributed scheduling in input queued switches", IEEE ICC 2007, Glasgow, Scotland, June 2007