Practical Algorithms for Multicast Support in Input Queued Switches

Andrea Bianco, Paolo Giaccone, Chiara Piglione, Sonia Sessa
Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy, Email: {firstname.lastname}@polito.it

***Abstract*— This paper deals with multicast flow support in $N \times N$ Input Queued switch architectures. A practical approach to support multicast traffic is presented, assuming that $O(N)$ queues are available at each input port. The focus is on dynamic queueing policies, where, at each input port, multicast flows are assigned to one among the available queues when flows become active: flows are assigned to queues according to switch queue status and, possibly, to flow information. We discuss queueing assignments, scheduling algorithms and flow activity definition models. We explain why dynamic queueing disciplines may outperform static policies, and we show that, even in the most favorable conditions for static policies, they provide comparable performance.**

I. Introduction

We consider cell-based synchronous IQ (Input Queued) architectures, where buffers reside only at input ports, and no internal speedup to the switching fabric is required. Arriving packets are fragmented into fixed-size cells, which are transferred from input to output ports through the switching fabric, following a scheduling discipline that must avoid contention: no more than one cell can be extracted from an input port in one time slot, and no more than one cell can be delivered to an output port in one time slot. Packets are then reassembled at output ports.

One possible way to support multicast traffic is to replicate multicast cells at inputs and treat them like unicast cells. This approach requires a speed increase at input ports, and can lead to poor bandwidth utilization of the switching fabric [1]. Since common switching fabrics (like the crossbar) have *intrinsic multicast capabilities*, i.e., they can transfer a multicast cell in one time slot from an input queue to possibly several output ports [2], we consider only switching fabrics with intrinsic multicast capabilities in this paper.

In the case of unicast traffic only, the well known Head-of-the-Line (HoL) blocking effect induced by FIFO queues [3] can be avoided by using, at each input, separate queues for each output (thus N queues per input, and N^2 queues overall); this queueing architecture is called *Virtual Output Queueing* (VOQ). HoL blocking for multicast traffic can be avoided using $2^N - 1$ queues for each input; this queueing architecture is called *MultiCast Virtual Output Queueing* (MC-VOQ) [1]. However, the MC-VOQ architecture is not practical in medium/large switches because of its poor scalability.

From a practical point of view, since the MC-VOQ architecture is infeasible, it is very important to study the performance of IQ switches when a limited number of queues is available at each input port. This implies that proper assignment strategies of multicast traffic to available queues must be defined. We assume that data in-order delivery is a requirement that is directly supported in the switch, avoiding any reordering at output ports; as a consequence, we cannot rely on cell-by-cell load balancing schemes (if not as a reference case), but we need to define queueing disciplines, i.e., assignments of multicast flows to queues. Previous work includes [4], [5], where performance of IQ switch with k queues per input port, with $k = O(N)$, are discussed. Both queueing and scheduling disciplines are defined and compared by simulation. However, both papers rely on a static assignment of multicast flows to queues. In other words, a pre-computed table contains the assignments of all possible multicast flows to available queues, according to criteria such as: fanout similarity, load balancing among queues or, simply, random or round-robin algorithms.

This static approach has some drawbacks: if not all the possible multicast flows are active at a given time, the queue assignment may turn out to be inefficient, since the assignment criteria envisioned a different multicast flow distribution. On the other hand, if the assignment is done assuming that a multicast traffic matrix is known, this would require a re-assignment whenever a new traffic matrix is detected; thus, traffic matrix estimation must be run, and re-assignments of flows to queues with the in-order delivery constraint becomes very difficult.

For all the above reasons, we focus on dynamic queueing disciplines, i.e., flows are assigned to queues whenever they become active in the switch. Dynamic disciplines have the advantage of being automatically adaptable to traffic changes without requiring traffic matrix estimation. Moreover, when the static flow to queue assignment is badly matched to the current traffic matrix, e.g. not all flows are active at a given time, or when the dynamism of multicast flows is very high, dynamic disciplines clearly have the potential of outperforming static disciplines. As a simple counter-example of the potential benefits of dynamic disciplines, consider the bad behavior of a static discipline queue assignment when only the flows assigned to a single queue per port are active: the system would have the same performance of an IQ switch with a single FIFO queue per port.

In this paper we mainly focus on the problem of the definition of queueing disciplines, relying on previously proposed scheduling algorithms for multicast traffic. Besides comparing performance of different queueing disciplines in the novel context of dynamic flow to queue assignment, we also look at finite buffer management schemes, trying to understand

0-7803-9569-7/06/$20.00 ©2006 IEEE

whether it is wiser to partition a given amount of memory either in several queues, with reduced memory for each queue, or in less queues with a larger amount of buffer for each queue.

II. System Model

Unless otherwise specified, we refer to switches with M input and N output ports, where all input and output lines run at the same data rate. The switching fabric is assumed to operate in a slotted fashion, and to have intrinsic multicast (and broadcast) capabilities, i.e., the cost of transferring in a time slot a cell from one input to one or more outputs does not depend on the number of destinations.

The average amount of offered traffic at each input (output) is called the input (output) load, and is measured in cells per time slot. Input (output) loads are normalized to line rates: a load equal to 1 means a fully utilized input (output) line. The traffic at the input of a switch is said to be *admissible* if no input load is larger than 1, and no output load is larger than 1.

Any multicast cell is characterized by its *fanout set*, i.e., by the set of output ports (destinations) to which the cell is directed. The cell fanout [6] is defined as the number of different destinations of a multicast cell, i.e., the cardinality of the fanout set. All the cells arriving to the same input and with the same fanout set identify a *multicast flow*. Note that unicast traffic is not given special attention, i.e., it is considered as a particular case of multicast traffic with fanout 1.

When a cell transferred in a given time slot through the switching fabric is sent to all its destinations, a total service policy (also named no-fanout splitting) is adopted, whereas a partial service discipline (fanout splitting) implies that only a subset of destinations is served in a given time slot.

The input queue system is organized according to a k-MC-VOQ architecture: each input port has k FIFO queues, with $1 < k < 2^N - 1$. In a k-MC-VOQ switch, the main design issues are related to (i) the scheduling algorithm that chooses the cells to transfer across the switching fabric and (ii) the definition of the queueing discipline that associates each multicast flow with a queue. In the following subsections, we describe separately the two issues.

A. Multicast Flow Activity Definition

When considering static disciplines, assignments for all multicast flows are pre-computed; thus, flows can be considered as always active, since their assignment is fixed and always available. Dynamic disciplines, being based on assignments determined when a flow becomes active, need a multicast flow activity definition.

We consider several possibile definitions of flow activity: flows are always active (AA flows), flows are active for a burst duration (BA flows), flows are active for a cell duration (CA flows) and flows are active as long as there is at least one cell belonging to the flow stored in the switch (SA flows). Note that a flow activity definition determines when a multicast flow to queue assignment decision is taken, thus impacting the system dynamic behavior.

For AA flows, the flow assignment is done (and never modified) when the first cell of the flow arrives at switch input ports; this definition is the most similar to the one taken when considering a static discipline. For SA flows, a new assignment decision is taken when a cell belonging to a multicast flow reaches the switch and no other cells belonging to the same flow are stored in switch memory. For BA flows, a new assignment decision is taken whenever a new burst of cells belonging to a flow arrives. Finally, the most dynamic flow activity definition is for CA flows, where a new, different assignment decision is taken independently for each cell. Note that these last options are used mainly as a reference case, since they do not guarantee cell in-order delivery.

B. Scheduling Discipline

We use two previously proposed multicast scheduling algorithms, named RS (Random Scheduler) and GMSS (Greedy Min Split Scheduler) [5], the former being chosen for its simplicity as a reference case, the latter due to its good performance. To satisfy the in-order cell delivery constraint, when a multicast cell is partially served, it remains at the head of its queue, and will contend for the residual set of destinations in the next time slot.

- **Random Scheduler (RS)**: at each time slot, each input port randomly selects a single not empty queue. The cell at the HoL of this queue sends a request for all the output ports belonging to its fanout set. Each output port selects randomly one request among the ones received (if any) and sends a grant to its corresponding input. Each input sends the cell at the head of the selected queue to all the destinations from which it received a grant.
- **Greedy Min-Split Scheduler (GMSS)**: a weight, representing the product of the queue length by the actual fanout of the cell at the head of the queue is associated with each queue. The queues are examined by decreasing order of queue weights. In each time slot, all inputs and outputs are set as unselected. Then, the scheduler performs sequentially two separate phases. During the first phase, a no-fanout splitting discipline is adopted to schedule at most 2 cells, the value being chosen to balance performance with algorithmic complexity. The scheduler selects the cell with the largest weight for a total service, and marks the corresponding input and outputs as selected. Among all the remaining cells present at unselected inputs and destined only to unselected outputs, the cell with the largest fanout is selected for a total service, marking the corresponding input and outputs as selected. During the second phase, a fanout splitting discipline is adopted. The scheduler examines, by decreasing order of weights, all queues of unselected input ports. The cell at the head of the examined queue is scheduled for a, possibly partial, service from the corresponding input port to all the unselected output ports belonging to its fanout set. The input and output ports chosen in this step become selected. The algorithm iterates the above process, by orderly examining all

queues of unselected input ports, until either all output ports are selected, or no more non-empty queues exist at unselected inputs.

C. Queueing Discipline

The queueing discipline task is to define, independently at each input port, an association between multicast flows and available queues. The queueing discipline is dynamic, i.e., it defines a dynamic association of multicast flows with queues: each new cell is enqueued according to the actual state of the system and not to a pre-defined, fixed, assignment. Note that guaranteeing in-order delivery with dynamic disciplines requires care in reallocating fanout sets to the available queues.

Intuitively, the queueing discipline should be designed such that the following *queueing criteria* are satisfied, as pointed out in [4]:

- cells with the same or "similar" fanout sets should be stored in the same queue, to reduce HoL blocking.
- the fanout sets of cells at heads of different queues should avoid common destinations.
- all queues should be equally used, i.e., some sort of load balancing among queues is useful, to both present a large number of possibilities to the scheduler and to efficiently use memory space.

The first two objectives can be more easily obtained with static disciplines, whereas dynamic disciplines are more naturally tailored to obtain the third objective. The dynamic queueing disciplines analyzed are:

- **Dynamic Random Queue (DRQ)**: each multicast flow is associated randomly with one of the k queues when it becomes active.
- **Dynamic Least Loaded Queue (DLLQ)**: multicast flows are partitioned into the k queues with the aim of dynamically balancing queue loads. We assume that multicast flow loads are known (either by explicit signalling or by estimation); a multicast flow that becomes active is associated with the queue that is currently less loaded.
- **Dynamic Shortest Queue (DSQ)**: multicast flows are partitioned into the k queues with the aim of dynamically balancing queue loads. However, no notion of multicast flow rate is used; rather, when a multicast flow becomes active, it is assigned to the queue that currently stores the least cells.

DRQ is a simple but, in general, not efficient queueing discipline, useful for performance comparison with other disciplines; both DLLQ and DSQ satisfy somehow the load balancing criteria, DSQ being easier to implement and not requiring any knowledge of multicast flow rates.

III. Simulation Study

In our simulations, we fix the number of output ports N either to 16 or 8, and consider different values of input ports $M \leq N$ to generate different degrees of load unbalancement between inputs and outputs. The performance metric is the maximum achievable switch throughput, i.e., we load the switch with admissible traffic with output load that approaches 1. Each FIFO queue has finite length, denoted by L, equal to 100 cells if not otherwise stated; cells are lost when they reach a full queue, according to a drop tail policy. The total amount of available memory at each input port is $B = L \times k$, k being the number of separate FIFO queues available at each input port.

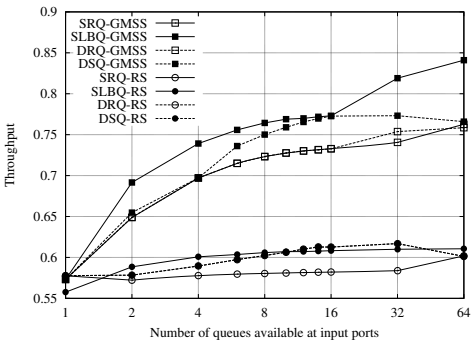

Fig. 1. Switch 5x16 with Bernoulli traffic ($h_m = 3.66$); static vs dynamic disciplines with SA flow activity definition.

Cells are generated according to either an i.i.d. Bernoulli process, i.e., at each time slot, an input port receives a cell with probability ρ, $0 \leq \rho \leq 1$, equal to the input load, or according to i.i.d. Batch arrival process, with average batch size equal to 10 cells. The fanout set of a new cell is generated randomly, according to a specific distribution. We denote by N_c the cardinality of the set of all generated fanout sets, and by P_f the probability of generating a fanout set with fanout f, $1 \leq f \leq N$. The considered distribution for the fanout sets is named *binomial fanout*, which was proven to be very difficult to schedule [5] and highlights disciplines differences: the fanout set is chosen according to a non-uniform binomial distribution, with mean fanout h_m. Hence, $N_c = 2^N - 1$ and $P_f = N/h_m \binom{N}{f} (h_m/N)^f (1 - h_m/N)^{N-f}$.

It was already shown in [5] that it is very important to focus on gathered traffic, i.e., traffic gathered over few active input ports ($M \leq N$) and equally distributed over all output ports. Indeed, when $M = N$ and inputs are equally loaded, the maximum sustainable traffic leads to a load at each input which is at most $1/h_m$. In this situation the efficiency of the scheduling and of the queueing discipline is not critical, and performance differences diminish. If instead the traffic is gathered among few inputs, the normalized input load ρ_{in} for sustainable traffic can approach 1, so that the efficiency in serving cells queued at the inputs becomes important on performance, and differences between disciplines become more evident. In all the considered simulation scenarios, we set $\rho_{out} = 1$, being $\rho_{in} = \frac{N}{M h_m}$.

We start by comparing static vs dynamic disciplines in the most favorable case for static disciplines, i.e., all flows are active; the 5×16 switch is loaded with Bernoulli traffic. Fig. 1 shows the average throughput for static (solid lines) and

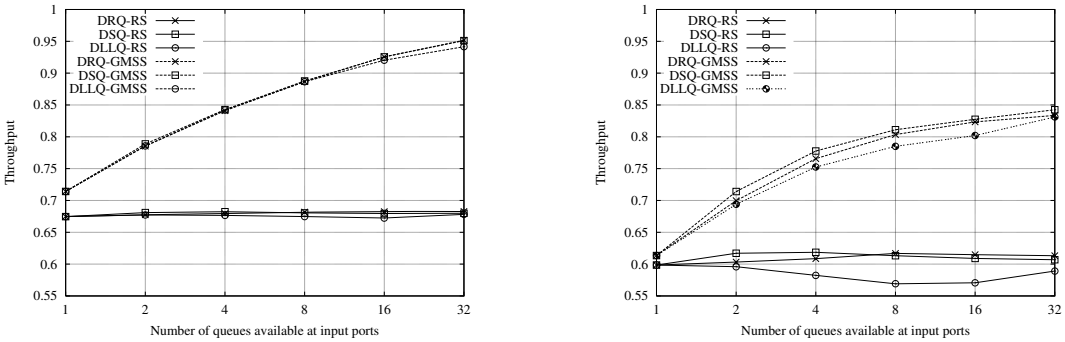

Fig. 2. Switch 3x8 with Bernoulli traffic and SA flow activity definition ($h_m = 3.66$ on the left, $h_m = 2.67$ on the right).

dynamic (dashed lines) disciplines as a function of the number of queues available at input ports k; note that, as the number of queues increases, also the total available memory at input ports increases. In the case of dynamic queueing disciplines, we assume the SA flow definition: flows are considered active, thus not re-enqueued, until at least a cell belonging to the flow is stored in the switch. Different combinations of scheduler (GMSS and RS) and queueing discipline (SRQ and SLBQ for static disciplines, DRQ and DSQ for dynamic disciplines) are considered. The SRQ and SLBQ disciplines are formally defined in [5]; shortly, SRQ is a static random assignment of multicast flows to available queues, whereas the SLBQ discipline is based on heuristically assigning multicast flows with known loads to balance the load among available queues. As already observed in [5], throughput improves when the number of queues k grows, but this improvement is significant only for small $k \leq 16$. Indeed, when the number of queues is large, the performance improvement for increasing k is negligible. Furthermore, the scheduling algorithm has a stronger impact on performance with respect to the queueing discipline. Finally, dynamic disciplines show a throughput comparable with those of static disciplines, an encouraging result, since the choice of matching the static queue assignment to the known traffic matrix is optimal for static disciplines.

Similar observations can be drawn when looking at Fig. 2, where we examine only dynamic disciplines for $M = 3, N = 8$, for SA flows. Again, the scheduler is clearly more important than the queueing discipline, and worse performance are obtained when the input and output loads approach 1, as in the case of $h_m = 2.67$. Results not reported here show that when using Batch arrival, the same general trends are observed; obviously, performance degrade for small values of k. Note also that if we use the same Bernoulli traffic scenario used in Fig. 2, with $h_m = 3.66$, but in a $M = N = 8$ switch, the output throughput saturates to 1 for 32 queues if using GMSS and to 0.92 for RS, again justifying the interest in examining gathered traffic scenarios. For what concerns the comparison among different queueing disciplines, DSQ provides slightly better performance with respect to DRQ, which often outperforms DLLQ. Although DLLQ should intuitively perform better that DRQ, since it exploits a direct measure (or notion) of multicast flows input load, it should be considered that the indirect measure used by DSQ (queue length) takes into account also the difficulty of the scheduler to transfer cells from input queues to output ports; as a consequence, the queue length is actually a more realistic measure of queue load with respect to the multicast flow input load itself. Since DLLQ does not provide any benefit with respect to the simpler DSQ and DRQ policies, we will not further consider it in the remainder of the paper.

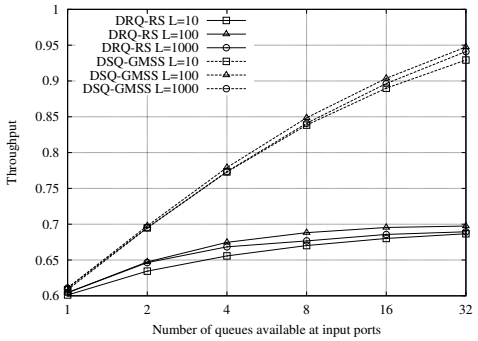

Fig. 3. Switch 3x8 with Batch traffic and SA flow activity definition ($h_m = 3.66$).

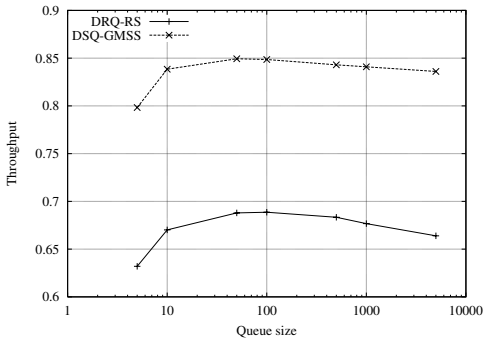

Fig. 4. Switch 3x8 with Batch traffic and SA flow activity definition ($h_m = 3.66$); k=8 queues are available at each input port.

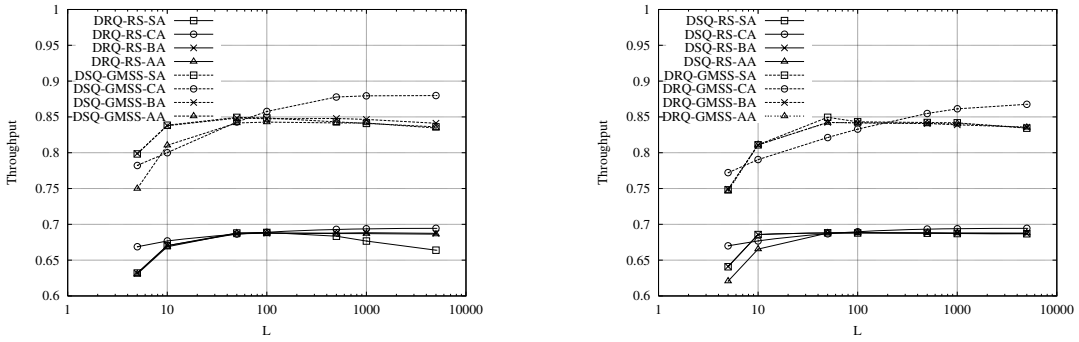

Fig. 5. Switch 3x8 with Batch traffic ($h_m = 3.66$)

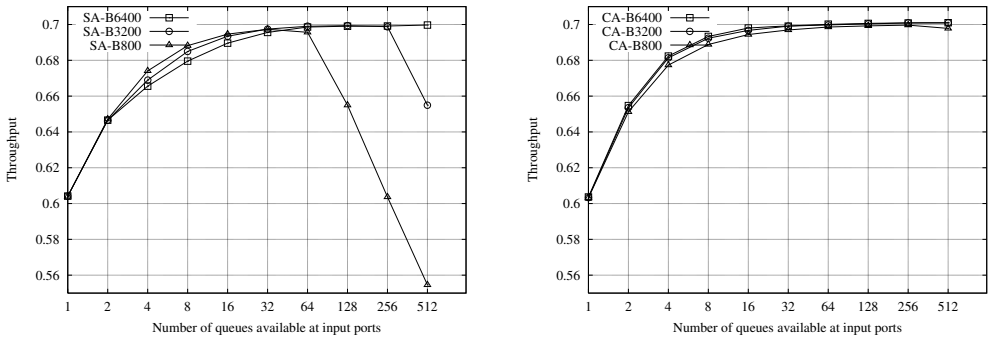

Fig. 6. Throughput when fixing the total amount of available memory B at each input port for DRQ-RS disciplines. Switch 3x8 with Batch traffic ($h_m = 3.66$).

Now we focus on the switch dimensioning problem. Fig. 3 shows the throughput in a 3×8 switch for RS with DRQ (solid lines) and GMSS with DSQ (dashed lines) under Batch arrivals, versus the number of queues, for different values of queue size L, ranging from 10 through 100 up to 1000; SA flows are considered. Whereas the number of queues has a significant impact on performance, the queue size has a less significant impact. However, it is interesting to observe that performance not always improve when increasing the queue length when fixing the number of queues. This is justified by the fact that increasing the queue length on the one hand decreases the loss probability; however, on the other hand, the system is less dynamic, since flows are active for a longer period of time, thus less likely re-enqueued, which induces a performance degradation. This behavior is more evident in Fig. 4, where we vary the queue size and fix the number of queues $k = 8$. For small queue size the loss probability dominates, whereas for large queue size the reduced dynamism in the system does not pay off.

To better understand the influence on the results of the flow activity definition, we plot in Fig. 5 the throughput for the four different flow activity definitions for all the combinations of queueing and scheduling disciplines. No cell re-ordering is considered at output ports for CA and BA flow activity definition. First, observe that only when using the SA flow definition a decrease in throughput when increasing the queue length is observed for the DRQ-RS policy, as already observed in Fig. 4. Second, the CA flow definition, i.e., each cell is enqueued independently, implies always a throughput increase for increasing queue length. Third, for increasing queue length, performance for SA and AA flows should become similar, since it is always less likely that no cells belonging to a flow are stored in the switch memory. This holds in all cases with the exception of DRQ queueing with RS scheduler, where the throughput obtained for the SA flow definition is smaller than the throughput for AA flow. The reason is the following: the random choice of a queue implies that there is always a probability to choose an already filled queue even if an empty queue exists; thus, on average, with RQ, a smaller number of queues can be active with respect to the number of available queues. Any scheduler works better when it can make a choice among a larger number of possibilities, i.e., when more queues are available. When using the SA flow definition, sooner or later a "bad" flow to queue assignment choice can be made. In this case, system performance decrease due to the difficulty in the scheduling decision, and queues build up. This implies that it is increasingly less likely to re-assign flows to queues for the SA flow definition, since more cells belonging to flows are stored in switch memory. As a consequence, it is difficult to move away from a "bad" assignment and overall performance decrease. This is more evident when adopting a RS scheduler, whereas the more efficient GMSS scheduler permits to empty the queues and to avoid this undesirable phenomenon. Finally, best performance are often obtained for

the CA flow definition, which allow the highest dynamism in the system to be obtained.

Finally, we study switch performance with a fixed overall memory size at inputs; in other words, we wish to understand whether it is more convenient to organize the memory with a larger number of shorter queues or a reduced number of longer queues. Indeed, more queues provide more choices to the scheduling algorithm but imply a higher drop probability due to the reduced queue length. In Fig. 6 we report throughput performance for SA flows (left) and CA flows (right) under the combination of DRQ-RS disciplines with Batch traffic as a function of the number of queues, for different values of total memory size (800 cells with triangles, 3200 cells with circles and 6400 cells with squares). First, as the number of queues increases, performance improve, until the size of each queue becomes comparable with the average batch size; in this regime, throughput drops significantly for SA flows. A relatively reduced number of queues is sufficient to obtain a throughput comparable with the asymptotically maximum value. For SA flows, with a medium number of queues, better performance are observed for shorter queue size; this phenomenon is again due to the SA flow activity definition, as previously described, since less dynamism is available in the system. Indeed, for the CA flow definition this behavior disappears.

IV. Conclusions

This paper discusses the effect of using a reduced number of queues in an IQ switch under multicast traffic. Different scheduling algorithms and queueing disciplines are discussed, and their performance are studied by simulation.

Simulation results show that i) gathered multicast traffic highlights performance differences among the proposed schemes, and ii) the effect of queueing disciplines on performance is less significant than the effect of scheduling algorithms.

Load balancing across queues, such as the one obtained by the DSQ queueing discipline, provide performance improvement. Indeed, DSQ provides slightly better performance with respect to DRQ, and often outperforms DLLQ. This shows that using a simpler and indirect measure of multicast flow rates, such as the instantaneous queue length, provides performance benefits due to the improved balancing among queues; indeed, this choice permits to take into account, not only the flow input load, but also the difficulty of the scheduler in transferring cells from input queues to output ports.

The SA multicast flows activity definition may lead to performance reduction for large queue size due to the reduced ability of re-enqueuing flows, which decreases system dynamism.

Finally, when partitioning a given total available memory, it is wiser to distribute it to more queues with reduced size, provided that each queue has a size greater than the average batch size in the incoming traffic flows.

References

[1] Ajmone Marsan M., Bianco A., Giaccone P., Leonardi E., Neri F., "Multicast traffic in input-queued switches: Optimal scheduling and maximum throughput", IEEE/ACM Transactions on Networking, Vol.3, No.11, pp.465-477, ISSN: 1063-6692, June 2003

[2] Hui J.Y., *"Switching and traffic theory for integrated broadband networks"*, Kluwer Academic Publishers, January 1990

[3] Hluchyj M.G., Karol M.J., Morgan S., "Input versus output queueing on a space division switch", *IEEE Transactions on Communications*, vol.35, n.12, pp.1347-1356, December 1987

[4] Gupta S., Aziz A., "Multicast scheduling for switches with multiple queues", *IEEE Hot Interconnects'02*, Stanford, CA, August 2002

[5] Bianco A., Giaccone P., Leonardi E., Neri F., Piglione C., "On the Number of Input Queues to Efficiently Support Multicast Traffic in Input Queued Switches", *HPSR 2003 (High Performance Switching and Routing)*, Torino, Italy, June 2003

[6] Hui J., Renner T., "Queueing strategies for multicast packet switching", *IEEE GLOBECOM'90*, San Diego, CA, December 1990