On the Number of Input Queues to Efficiently Support Multicast Traffic in Input Queued Switches

Andrea Bianco, Paolo Giaccone, Emilio Leonardi, Fabio Neri, Chiara Piglione
Dipartimento di Elettronica - Politecnico di Torino - Italy
e-mail: {andrea.bianco, paolo.giaccone, emilio.leonardi, fabio.neri, chiara.piglione}@polito.it

Abstract— **This paper studies the design of IQ cell-based switch architectures with a reduced number of queues at input ports to support multicast traffic delivery. The design involves the definition of a queueing policy (to select how and where to enqueue packets) and of a scheduling policy (to select when and from which queue to transfer packets, satisfying the transmission constraints given by the switching fabric). The possible tradeoffs between performance figures, queueing policies and scheduling algorithms, are discussed in the paper. Clever design can lead to non-negligible performance gains; thus, several queuing policies and scheduling algorithms are examined by simulation. The main contributions of the paper are the following. First, it is shown that a small number of queues is sufficient to obtain good switch performance under typical traffic patterns. Second, multicast traffic patterns difficult to schedule are obtained by concentrating the load over few input ports, while keeping the output port load high. Third, the scheduling algorithm is shown to have a deeper impact on the switch performance than the queuing policy. Fourth, queueing policies that balance the load among all the available queues are shown to perform better than queuing policies based on other criteria. Finally, a novel greedy scheduling policy that improves switch performance is presented.**

I. Introduction

The efficient support of multicast traffic in high-speed networks is considered crucial by many researchers to provide high-quality services to a number of applications. In the Internet, most of the routers present in the core are based on switching architectures with input queues (IQ), for which the efficient support of multicast traffic can be very difficult to implement. Indeed, it has already been shown that, to achieve the maximum throughput, an $N \times N$ IQ switch requires a large amount of queues, in the order of 2^N for each input, and a complex scheduling algorithm, which is not practical even for very small switches.

We consider pure IQ architectures, where buffers reside only at input ports, and no internal speedup to the switching fabric is required. These architectures are considered to be the most promising for the implementation of extremely fast packet switches and routers [1], thanks to the fact that the aggregate bandwidth required in their switching fabric does not grow with the sum of the data rates of input links. Instead, OQ (Output Queued) switch architectures with N input/output ports require switching fabrics and output memories whose bandwidth must be up to N times the sum of the data rates of input links. Moreover, under multicast traffic, buffer space may be used more efficiently by IQ switches than by OQ switches, in the sense that any multicast cell currently in the switch can be stored using only one buffer position.

We consider a *cell-based* paradigm within switches: arriving packets are fragmented into fixed-size cells, which are stored into buffers at input ports. Cells 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.

In the case of unicast traffic only, when unicast cells are stored in just one FIFO queue per input port, the cell at the head of the queue can block the access to the switching fabric of subsequent cells, leading to the well-understood Head-of-the-Line (HoL) blocking effect [2], which limits the maximum throughput achievable by IQ switches. HoL blocking 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).

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, as pointed out in [3]. Furthermore, some 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, with no extra cost [4], [5] with respect to unicast transfer. In this paper we consider only switch fabrics with intrinsic multicast capabilities, and exploit them to achieve the highest bandwidth utilization.

Multicast scheduling algorithms can be based on fanout splitting or no-fanout splitting disciplines. When a *fanout splitting* discipline is adopted, a multicast cell can be transferred to its destinations in multiple time slots. During a single time slot, the cell is transferred only to a subset of the destinations; in a given time slot, a *partial service* [10], [12] is adopted when a cell reaches a subset of its remaining destinations, whereas a *total service* is adopted when a cell reaches all its (possibly remaining) destinations. When a *no-fanout splitting* discipline is adopted, a multicast cell is transferred to all its destinations in a single time slot.

In the case of fanout splitting, every partial service causes an increase of the input load, since the same cell must be transmitted several times, thus leading to throughput degrada-

0-7803-7710-9/03/$17.00 © 2003 IEEE.

tion [11]. On the contrary, no-fanout splitting disciplines avoid the increase of input load due to partial services [3], but at the same time some outputs may idle because of the contentions among different inputs; usually this latter fact leads to more significant throughput degradation.

Many scheduling algorithms for multicast traffic have been proposed, but a large number of them maintain only a queue at each input port. Examples of these algorithms are the Concentrate algorithm, TATRA and WBA [6]. Other proposed scheduling algorithms combine efficiently fanout-splitting with no-fanout splitting disciplines [7]. In all these cases, the throughput degradation given by the HoL blocking cannot be avoided.

HoL blocking for multicast traffic can be completely avoided using about 2^N queues for each input; this queueing architecture is called *MultiCast Virtual Output Queueing* (MC-VOQ) [3], [8]. However, the MC-VOQ architecture is not practical in medium/large switches because of its poor scalability. Note that the contribution of [3], [8] is on the definition of the optimal (i.e., that guarantees throughput maximization) queueing and scheduling policies; it is shown that 100% throughput may not be achieved even with the optimal queuing and scheduling policies under a generic admissible multicast traffic.

From a practical point of view, it is very interesting to study the tradeoff between complexity (when the number of queues is reduced with respect to 2^N per input port) and performance (when high throughput is achieved and the sequence of cells is preserved). One of the first works on this topic was presented in [9], which studied an IQ switch with only k queues per input port, with $k = O(N)$. In this paper we extend the results of [9] to more general queueing systems and scheduling algorithms.

II. System Assumptions

In this section we introduce some basic definitions on multicast traffic scheduling, and we illustrate the switch architecture considered in this paper.

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. All the queues are assumed to be FIFO.

The average amount of offered traffic at each input (output) is called the input (output) load, and is measured in cells per time slot. We normalize input (output) loads to line rates: a load equal to 1 means a fully utilized input (output) line (i.e., 1 cell per time slot). 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. An input traffic is said to be *sustainable* if it can be transferred through the switch.

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 [10], [11], named *fanout size* in this paper, is defined as the number of different destinations of a multicast cell, i.e., the cardinality of the fanout set. We say that a cell has *fanout destination* j when output port j ($1 \leq j \leq N$) belongs to the fanout set of the cell. All the cells arriving to the same input and with the same fanout set identify the same *multicast flow*. The fanout set of a cell can be represented by a binary string of length N, $c = [c_N, \ldots, c_1]$ with $c \in \{0, 1\}^N$ and c_j is 1 iff the cell has fanout destination j. The *fanout value* is the value obtained by reading the string as a binary number. It is possible to define the *distance* between two fanout sets; in our study we have considered only the Hamming distance (i.e. the number of different bits in the same positions in the two strings representing the two fanout sets) between the corresponding binary strings, where null distance corresponds to identical fanout sets.

The input queue system is organized with k FIFO queues for each input, with $1 < k < 2^N - 1$; this architecture is named k-MC-VOQ architecture. Note that $2^N - 1$ is the number of all the possible fanout sets and corresponds to the number of queues used in a MC-VOQ architecture [3].

In a k-MC-VOQ switch, the main design issues are related to (i) the definition of the queueing policy that associates each multicast flow with a queue and (ii) the scheduling algorithm that chooses the cells to transfer across the switching fabric. Note that the choice of the queue structure obviously affects the scheduling discipline, since the cells that can be examined in each time slot are always a (small) subset of all cells stored at input ports. On the other hand, the scheduling algorithm may be tailored to the chosen queue architecture. However, we will describe in the following subsections the two issues separately.

A. Scheduling Algorithm

We propose three multicast scheduling algorithms. Because of the higher throughput achievable, these scheduling algorithms adopt a fanout splitting discipline. We impose that the switch delivers the cells while preserving the FIFO order for each input-output pair; to satisfy this 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 the outputs port 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 Scheduler (GS)**: a weight, whose meaning is discuss below, is associated with each queue. At each time slot all input and output ports are initially *unselected*. 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.
This scheduler is similar to the one proposed in [3]. One natural choice of weight is the queue length, since it is the same weight used in defining the optimal policy. However, if the maximum queue length is finite, this weight is significant only when losses are not experienced; on the contrary, all the queues experiencing losses have constant queue length, independently from their congestion level. Thus, we studied greedy algorithms also with different definition of weights. In some of our simulations (not reported here for lack of space), we compared the performance of the original GS using the queue length as weight with a variant of GS using the product of the queue length by the actual fanout size of the cell at the head of the queue. We observed generally better performance of the latter version of GS, and for this reason in this paper we consider only this possibility.

- **Greedy Min-Split Scheduler (GMSS)**: as in the case of Greedy algorithm, all the queues are examined by decreasing order of queue weights (that have the same definition of GS). In each time slot, all the 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 $i = 2$ cells. 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 in unselected inputs and destined *only* to unselected outputs, the cell with the largest fanout size for a total service is selected, marking the corresponding input and outputs as selected. During the second phase, a fanout splitting discipline is adopted. The GS algorithm described above is run to choose the cells among all the unselected inputs and outputs. The parameter i has been set equal to two to balance performance with algorithmic complexity.

With RS, some outputs can be idle because of the contentions among different inputs during the single iteration of the algorithm. With GS, outputs are never idle when they can receive cells from unselected inputs; furthermore, GS uses the queue length as congestion information and exploits also the information about the state of fanout sets of the cells at HoL. In addition, GMSS combines the advantages of a no-fanout splitting discipline during the first phase of the algorithm with the advantages of a fanout splitting discipline during the second phase.

For these reasons we expect that GMSS performs the best among the three algorithms from the throughput point of view, and GS performs better than RS. Our expectations will be met by the simulation results.

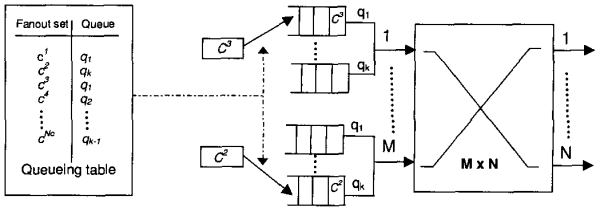

Fig. 1. Example of static queueing policy. Two cells arrive at inputs 1 and M, the first one with fanout set c^3 and the second with fanout set c^2. From the queueing table, c^3 is destined to queue q_1 whereas c^2 to queue q_k.

B. Queuing Policy

A queuing policy must be defined to associate each multicast flow with a queue. The queueing policy can be dynamic or static. In the former case, the policy defines a dynamic association of the multicast flow with the queues, by which each new cell is enqueued according to the actual state of the system. In the latter case, the policy defines a static association between the flows and the queues; the association is stored in a unique *queueing table*, shared among all input ports, which is computed "a priori" at system startup. In this paper we consider only static queueing policies; they permit also to simply avoid the out-of-sequence delivery problem: guaranteeing in-order delivery with dynamic policies requires care in reallocating fanout sets to the available queues. Figure 1 shows an example of the queueing process.

Note that only when $2^N - 1$ queues are present for each input, the association between multicast flows and queues is bijective. Otherwise, with a reduced number of queues, more than one flow is associated with the same queue and this can cause HoL blocking. However, recall that, even with $2^N - 1$ queues, if a cell that received partial service is not re-enqueued to a different queue, a new form of HoL blocking arises, due to the limited number of destinations reachable by the HoL cell with respect to the fanout set of the cells sitting behind in the queue. If instead partially served cells are re-enqueued to avoid HoL blocking, in-order delivery cannot be guaranteed.

Intuitively, the queueing policy should be designed such that the following *queueing criteria* are satisfied, as pointed out in [9]:

- cells with the same or "similar" fanout sets should be stored to the same queue, to reduce the HoL blocking.
- the fanout sets of cells at heads of different queues should be "diverse", in the sense that they should avoid common destinations.

Moreover, load balancing among queues may be beneficial to improve performance.

The choice of the queuing policies adopted in this paper is driven by those intuitive observations. We propose to measure the similarity and diversity degree between two fanout sets by considering the Hamming distance between them.

The queueing policies we propose are:

- **Random Queueing (RQ)**: each multicast flow is associated randomly with one of the k queues.

- **Minimum-Distance Queueing (MDQ)**: this scheme is designed to easily allocate the fanout sets to the queues. A fanout set, called *representative of the queue*, is associated with each queue. Each flow is allocated to the queue with the representative for which its Hamming distance from the flow is minimized. If more than one representative have minimum distance from the given flow, the policy selects one of them randomly.
- **Load-Balanced Queueing (LBQ)**: multicast flows are partitioned into the k queues such that the output loads are equalized. The solution of this partitioning problem is solved by the following simple heuristic. A list is created with all the multicast flows sorted by increasing order of a specified metrics (either its fanout size **(LBQ-FS)** or its fanout value **(LBQ-FV)**). If the overall output load is η_T, then the expected output load is η_T/k for each queue. By orderly scanning the sorted list, the queueing policy creates partitions with the minimum set of the queues whose aggregate load is non smaller than η_T/k.

RQ is an intuitive queueing policy, very useful for performance comparison with the other policies; it is not efficient, since on average does not satisfy all the queueing criteria discussed earlier. MDQ policy satisfies most of the queueing criteria, since multicast flows with similar fanout sets are stored in the same queue. Since the MDQ can generate some unfairness between queues with different loads, we propose LBQ policies to highlight the effect of load balancing schemes.

Note that the choice of the representatives of the queues in the MDQ scheme may be formalized as an optimization problem. In some scenarios, the choice of the representative has a significant impact on performance results. We devised the following iterative heuristic to guide the representative choice in our simulations. For each iteration, k representatives are selected at random, among all the possible fanout sets corresponding to existing (i.e., fanout set that may be generated according to the traffic pattern) multicast flows. The Hamming distance between the representative of each queue and all the enqueued fanout sets is computed. This procedure is iterated a number of times. At the end, we choose the representatives corresponding to the average minimum distance on all queues. In this way, we are able to find a set of "distant" representatives, partitioning the space of all possible fanout sets according to the queueing criteria. This is not, in general, an optimal solution; better performance can be obtained under particular traffic patterns by choosing the representatives with an "ad-hoc" solution. However, this approach has the advantage that can be adopted regardless of the specific traffic pattern.

III. Simulation Study

In our simulations, we fix the number of output ports N and consider different values of input ports $M \leq N$, to generate different degrees of load unbalancement between inputs and outputs. The FIFO queues have finite length; we suppose that the length of each queue is equal to 100 cells; a cell is lost when it reaches a full queue.

TABLE I
OUTPUT THROUGHPUT ACHIEVABLE BY GS IN A $M \times 16$ SWITCH, UNDER UNIFORM FANOUT TRAFFIC.

ρ	M	k	Output load	Output throughput
0.123	16	32	0.984	0.984
0.123	16	6	0.984	0.984
0.123	16	5	0.984	0.982
0.123	16	3	0.984	0.97
0.123	16	1	0.984	0.918
0.985	2	6	0.985	0.646
0.985	2	5	0.985	0.645
0.985	2	3	0.985	0.639
0.985	2	1	0.985	0.614

A. Traffic scenarios

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 ρ, $0 \leq \rho \leq 1$, equal to the input load. 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 size f. The considered distributions for the fanout sets are the following:

- *uniform fanout*. The fanout set is chosen uniformly at random among all the possible fanout sets. Hence, $N_c = 2^N - 1$ and $P_f = \binom{N}{f}/N_c$.
- *uniform maximum fanout*. The fanout set is chosen uniformly at random among all the possible fanout sets whose fanout size is less or equal than h_M. Hence, $N_c = \sum_{k=1}^{h_M} \binom{N}{k}$ and $P_f = \binom{N}{f}/N_c$.
- *binomial fanout*. The fanout set is chosen according to a non-uniform binomial distribution, with mean fanout size h_m. Hence, $N_c = 2^N - 1$ and

$$P_f = \binom{N}{f} (h_m/N)^f (1 - h_m/N)^{N-f}.$$

In terms of traffic distribution among input and output ports, we consider both a scenario in which all input (and output) ports are equally loaded and a scenario where the traffic is gathered over few input ports and equally distributed over all output ports.

B. Importance of gathered traffic

We study first the performance of the GS scheduler under different traffic scenarios for a $M \times N$ switch, with fixed $N = 16$. These results are almost independent of the adopted queueing policy.

Table I shows the output throughput under a traffic scenario with uniform fanout. We explored the sensibility of the performance to the input load ρ, to the number of active input ports M, and to the number of queues k for each input and the output load (which is equal to $\rho M h_m/N$, with h_m denoting the average fanout size).

When $M = 16$, all input ports are active and equally loaded. In this case, Table I shows that 6 queues are sufficient

TABLE II

OUTPUT THROUGHPUT ACHIEVABLE BY THE GS SCHEDULER IN A $M \times 16$ SWITCH, UNDER UNIFORM MAXIMUM FANOUT TRAFFIC WITH $h_M = 4$.

ρ	M	k	Output load	Output throughput
0.286	16	5	0.98	0.95
0.873	5	5	0.998	0.63

to achieve the maximum throughput. When k decreases, the output throughput decreases to 0.918 just for one queue. This high value of throughput is not surprising, since it has been already shown theoretically [10] that multicast traffic is less critical to schedule than uniform traffic when the switch use a single queue at each input port. This is due to the fact that a multicast cell offers to the scheduler, when adopting a fanout splitting discipline, more transmission possibilities then a unicast cell. Moreover, when $M = N$ and inputs are equally loaded, the maximum sustainable traffic leads to a load at each input which is at most $1/h_m$ (note that input load in Table I is 0.123). In this situation the efficiency of the scheduling and of the input queueing discipline is not critical, and performance differences diminish.

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, and differences between our different policies become more evident. For example, Table I shows that, when $M = 2$, the final throughput is much lower than the one achievable when all inputs are active.

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 these situations, an efficient handling of these flows internally to the router/switch may permit to sustain rates, hence to provide a quality of service, that would not be possible with straightforward approaches, such as cell replication at input ports.

Table II shows some results under a traffic scenario with uniform maximum fanout and $h_M = 4$ (which corresponds to an average fanout size equal to 3.66). Output load is almost equal to 1 for both cases. Having fixed $k = 5$, we note that losses are high only for gathered traffic (M low).

From these results and many others not reported here for lack of space, we can conclude that it is difficult to achieve the maximum throughput when we concentrate the load on a reduced number of input ports and the output load is high. Thus, in the following section we restrict our studies only to gathered traffic scenarios to investigate the impact of the scheduling and queueing policies on performance.

C. Effect of queueing and scheduling policies

We study the throughput behavior with respect to the number of queues k, the queuing policy and the scheduling algorithm. The switch is loaded by gathered traffic, with

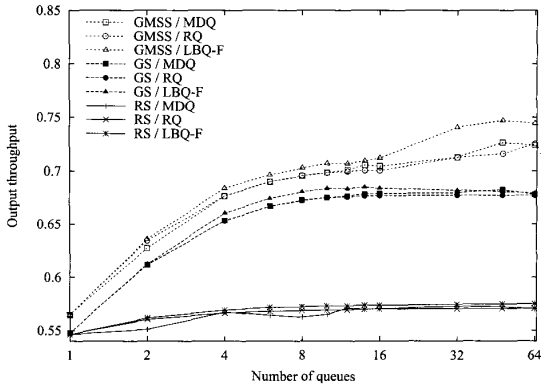

Fig. 2. Switch 5x16 with uniform maximum fanout distribution of fanout set ($h_M = 4$); labels follow the convention: scheduling policy/queueing policy.

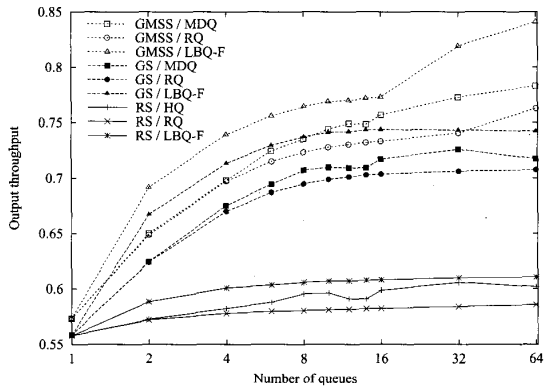

Fig. 3. Switch 5x16 with binomial distribution of fanout set ($h_m = 3.66$); labels follow the convention: scheduling policy/queueing policy.

$M = 5$ and with output load close to saturation, equal to 0.998 (corresponding to an input load equal to 0.873). The two considered scenarios are with uniform maximum fanout (with $h_M = 4$) and with binomial fanout (with $h_m = 3.66$ set equal to the average fanout size of the first scenario). In the first scenario the number of possible fanout sets is 2516, whereas in the second scenario it is $2^N - 1 = 65535$.

Figure 2 shows the throughput achieved under the first traffic scenario for different combinations of scheduler (GMSS, GS and RS) and queueing policy (MDQ, RQ, LBQ-FS). Note that LBQ-FV does not appear in the figures, since it shows the same performance as MDQ. In all the cases, we observe a throughput improvement when the number of queues k grows. But this improvement is significant only when k is small (not larger than 16). When the number of queues is large, the performance improvement for increasing k is negligible. For this reason, we propose the rule of thumb that the number of queues should be between N and $2N$ for each input port to design a robust queueing system.

Furthermore, we observe the stronger impact of the schedul-

ing algorithm on the performance with respect to the queueing policy. Indeed, by changing the queueing policy, the output throughput does not changes significantly; on the contrary, by changing the scheduling algorithm, the throughput passes from about 0.57 for RS (dashed lines) to 0.73 for GMSS (solid lines). In particular, GMSS with LBQ-FS outperforms all the other schemes.

Figure 3 shows the throughput achieved under the second traffic scenario, again for the all combinations of scheduler and queueing policy. The same considerations as above hold, but now the throughput is more sensible to the queueing policy, due to the larger number of possible fanout sets. The GMSS scheduler still outperforms the other schedulers. The best queueing policy is always LBQ-FS, and the worst is RQ; the difference in throughput is at most about 10%. This motivates our final observation. Load balancing across the queues may have relevant effects on the performance.

IV. Conclusions

This paper discusses the effect of a reduced number of queues in an input-queued switch under multicast traffic. Different scheduling algorithms and queueing policies are discussed, and their performances are studied by simulation in a number of their possible combinations. Our simulation results show that only gathered multicast traffic highlights significant performance differences among the proposed schemes. Under gathered traffic, we studied the effect of the number of queues and found out that $2N$ queues are usually sufficient to achieve the highest throughput. The effect of the queueing policy on the performance is less significant than the effect of the scheduling algorithm; load balancing scheme across queues provide performance improvement over other queueing schemes. A novel greedy scheduling policy that provides better performance than previously proposed scheduling algorithms has been presented.

References

[1] Chao H.J., "Next generation routers", *Proceedings of the IEEE*, vol.90, n.9, pp.1518-1558, September 2002

[2] 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

[3] Ajmone Marsan M., Bianco A., Giaccone P., Leonardi E., Neri F., "On the throughput of Input-Queued cell-based switches with multicast traffic", *IEEE INFOCOM'01*, Anchorage, Alaska, April 2001

[4] Guo M., Chang R., "Multicast ATM switches: survey and performance evaluation", *Computer Communication Review*, vol.28, n.2, pp.98-131, April 1998

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

[6] Prabhakar B., McKeown N., Ahuja R., "Multicast scheduling for input-queued switches", *IEEE Journal on Selected Areas in Communications*, vol.15, n.5, pp.855-866, June 1997

[7] Veltri L., "Maximum throughput in multicast input queued packet switches", *IEEE ICC'01*, Helsinki, Finland, June 2001

[8] Ajmone Marsan M., Bianco A., Giaccone P., Leonardi E., Neri F., "Optimal multicast scheduling in Input-Queued switches", *IEEE ICC'01*, Helsinki, Finland, June 2001

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

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

[11] Kim C.K., Lee T.T., "Performance of call splitting algorithms for multicast traffic", *IEEE INFOCOM'90*, San Francisco, CA, June 1990

[12] McKeown N., Prabhakar B., "Scheduling multicast cells in an input-queued switch", *IEEE INFOCOM'96*, San Francisco, CA, March 1996