Fair Integrated Scheduling of Unicast and Multicast Traffic in an Input-Queued Switch

Enrico Schiattarella*
Dipartimento di Elettronica, Politecnico di Torino
Corso Duca degli Abruzzi 24, 10129 Torino, Italy
enrico.schiattarella@polito.it

Cyriel Minkenberg
IBM Research, Zurich Research Laboratory
Säumerstrasse 4, 8803 Rüschlikon, Switzerland
sil@zurich.ibm.com

***Abstract*— We present a scheme to concurrently schedule unicast and multicast traffic in an input-queued switch. It aims at providing high performance under any mix of the two traffic types as well as at avoiding starvation of any connection. The key idea is to schedule the two traffic types independently and in parallel, and then arbitrate among them for access to the switching fabric. The unicast and multicast matchings are combined into a single integrated matching. Edges that are excluded from the integrated matching are guaranteed to receive service at a later time, thus preventing starvation. We use simulation to evaluate the performance of a system employing the proposed scheme and show that, despite its simplicity, the scheme achieves the intended goals. We also design an enhanced remainder-service policy to achieve better integration and further improve performance.**

I. Introduction

The vast majority of Internet traffic today consists of unicast (point-to-point) connections. However, efficient support for multicast (point-to-multipoint) traffic is essential for communication applications such as audio- and video-conferencing, multimedia content distribution (radio, TV) and remote collaboration, as well as computing applications such as the implementation of collective operations or snoop-based cache coherency in parallel computers. Ideally, a network switch should be able to achieve high performance under any mix of the two traffic types.

Multicast packets can be treated as unicast simply by sending a separate copy of the packet to each of the intended destinations; conversely, unicast packets can be considered as multicast packets that have only one destination and are treated without any differentiation. These trivial solutions allow the switch to handle both types of traffic concurrently, but are far from optimal and in general lead to poor performance.

Another issue to address when both unicast and multicast are present is fairness. A traffic type must not be allowed to monopolize switch resources; however, it is also important to guarantee that *all* connections of a given traffic type receive service. When both conditions are met, we say that the switch scheduler is *fair*.

*This work was performed while the author was at the IBM Zurich Research Laboratory, Rüschlikon, Switzerland.

This research is supported in part by the University of California under subcontract number B527064.

In this work we propose a novel method for integrated scheduling of unicast and multicast traffic. It leads to high utilization of switch resources under any traffic mix, guarantees fairness, and exhibits a number of other desirable properties.

In the remainder of this section, we recall some relevant results and discuss related work. In Section II we present our scheme and highlight its key features. The performance results obtained by simulation are shown in Section III. In Section IV we propose an improvement to the base scheme and show the benefits it provides. Next we briefly discuss implementation issues. Finally we summarize the results and propose directions for further work.

A. Background

In *input-queued* (IQ) switches, packets are buffered only at the inputs, and a centralized scheduler resolves the contentions for the access to the switching fabric. This architecture is very attractive because it allows the switching fabric and the memories to run at the same data rate as the input links, thus providing high scalability of the data path.

IQ switches often operate in a synchronous fashion: time is divided into *slots* of equal duration, and during a time slot a fixed-size data unit called *cell* can be transmitted through the fabric. Incoming packets are segmented into cells at the inputs and reassembled at the outputs. If the fabric is a crossbar, then only one cell can depart from each input and only one cell can arrive at each output during a time slot. However, a cell departing from one input can be received at multiple outputs. In other words, the crossbar naturally supports multicast traffic because it can replicate a cell to multiple outputs at no additional cost.

To achieve high throughput, unicast cells arriving at an input are placed in different queues depending on their destination. These per-destination queues are called *virtual output queues* (VOQs) [1] and prevent the head-of-the-line (HOL) blocking phenomenon, which would severely limit the maximum throughput [2].

The problem of scheduling unicast traffic consists of deciding in every time slot which VOQ at each input will be served. This is equivalent to finding a bipartite graph *matching* between the set of switch inputs and outputs. An *edge* connecting node i to node j indicates that the crosspoint (i, j) is to be closed. Unicast scheduling has received much

attention in the past, and very efficient algorithms that achieve 100% throughput are known [3]–[5].

Scheduling multicast cells taking advantage of the crossbar replication capabilities, on the contrary, is a more difficult and less understood problem. The *fanout* of a multicast cell is the set of outputs to which the cell is destined.[1] A multicast matching consists of edges that connect a single input to one or more outputs. The problem has been studied from a theoretical point of view in [6] and [7], and its computational hardness is established in [8]. In [9] the optimal scheduling discipline is defined, but neither the discipline itself nor the assumed queuing architecture are practically implementable. Nevertheless in [10] the authors provide important insight into the nature of the problem and propose algorithms with reasonable complexity and relatively good performance.

B. Previous work

Although the problem of supporting unicast and multicast concurrently is clearly important, not much attention has been devoted to it in the past. The problem has been thoroughly studied from a theoretical point of view in [8] and its hardness has been assessed. These authors also propose an integration scheme that consists of scheduling multicast first and using the remaining resources for unicast. This scheme, which we call "sequential," predictably leads to high performance because it uses the switch resources very efficiently. The multicast scheduler has all the resources at its disposal and can produce its best matching. The unicast scheduler, on the contrary, is constrained by the remaining resources but, owing to the VOQs, it can fully exploit them and increase the size of the total matching. The main disadvantage of this scheme is that it easily leads to starvation of unicast traffic. A single input loading the switch with broadcast traffic would suffice to prevent unicast from getting any service at all.

Moreover, the problem was also considered in [11], but the proposed solution is mainly suitable for shared-memory switches.

II. Fair Integrated Scheduling

Our integration scheme is conceived for a synchronous, IQ, crossbar-based, $N \times N$ switch (Figure 1). We assume that each input maintains N VOQs for unicast and a single FIFO queue for multicast.

A. Reference architecture

At every time slot, contentions among the cells of a single traffic type are resolved separately by specialized schedulers. The unicast scheduler receives requests from the inputs for nonempty VOQs and produces a one-to-one matching between the inputs and the outputs. The multicast scheduler examines the fanout of the cells that are at the HOL of the multicast queues and produces a one-to-many matching. Fanout splitting is allowed: during a time slot a multicast cell can receive partial service, i.e., it is being transmitted only to a subset of its destinations.

[1]We use the same term to refer to the cardinality of the set as well.

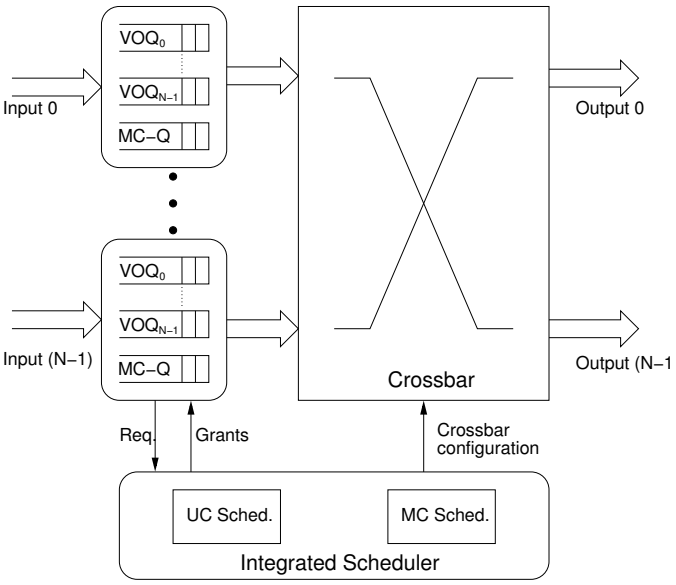

Fig. 1. Reference architecture

As the two schedulers run in parallel and independently, the matchings they produce in general are overlapping, meaning that they have conflicting edges. To obtain a consistent configuration for the crossbar, the two matchings must be combined into a single one. An *integration block* decides which unicast and multicast edges will be part of the integrated matching. The set of edges that are excluded from the integrated matching is called the *remainder*.

The *request filter* is a block capable of reserving a subset of the switch inputs and outputs by dropping the corresponding unicast and multicast requests. Reservations at any time slot may be made on the basis of information provided by a number of sources, including current requests and the integration block.

Employing two different schedulers that run in parallel provides important advantages. The designer is free to choose the algorithms that best fit his or her needs. The system can easily be partitioned over multiple chips. The minimum time-slot duration is determined by the scheduling time of the slowest scheduler, whereas, if the schedulers ran in sequence, it would be limited by the sum of the two.[2]

A block diagram of this scheme, called "FILM" (FILter & Merge), is shown in Fig. 2.

B. Achieving fairness

In the FILM scheme, each connection experiences two points of contention: first it competes with the other connections belonging to the same traffic type, then with those of the other traffic type. To achieve fairness we must make sure that every connection regularly has a chance to win both contentions.

[2]We assume that the delay contributed by the additional blocks is much lower than the scheduling times. As we will see in Section V, which discusses implementation complexity, this assumption is likely to hold.

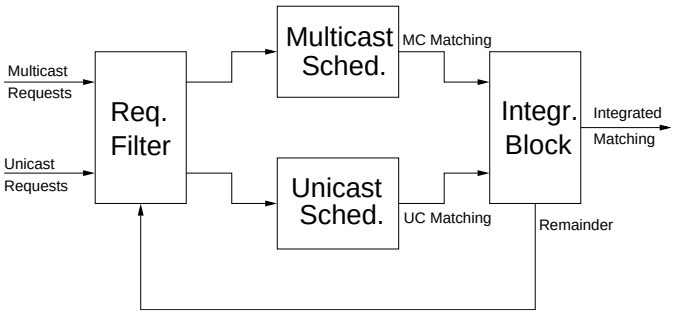

Fig. 2. The FILM integration scheme

A scheduling algorithm is starvation-free if it guarantees that no queue is allowed to remain unserved indefinitely. As this is a fundamental property, many algorithms exhibit it. Unicast algorithms such as iSLIP and DRRM prevent starvation by using pointers that keep track of which VOQs have been served most recently. Multicast algorithms, on the other hand, often take into account the age of a cell or the order in which cells at different inputs have advanced to the HOL of their queues (e.g. WBA and TATRA [10], respectively). We require both schedulers to employ starvation-free algorithms to be sure that all connections eventually get past the first contention point.

Connections that have been selected by their schedulers will still remain unserved if the integration block excludes them from the integrated matching. The scheduler is unaware of the fact that granted service has in fact been withdrawn, so fairness is no longer guaranteed. A solution to this problem is to ensure that all edges that are part of the remainder actually receive service, albeit in a later time slot.

C. Integration policy

The performance of multicast scheduling algorithms varies considerably, as demonstrated in [10]. This is because the single FIFO queuing architecture causes HOL blocking, therefore the algorithms must carefully choose which inputs to serve to mitigate its effects. For example, it is shown that "concentrating the residue" at every time slot (which roughly means providing full service to as many inputs as possible) greatly helps in draining the queues fast. Hence, special care should be taken when manipulating multicast matchings to avoid compromising the effectiveness of the choices made by the scheduler.

Unicast scheduling, on the contrary, is less sensitive to withdrawal of resources because the VOQs provide the scheduler with a wide choice of connections to serve. Moreover it is important to note that if unicast and multicast contend for an input, only one edge is lost if multicast wins, whereas multiple edges might be removed if it loses.

Following these considerations, we opt for an integration policy that gives strict priority to multicast over unicast. Hence, the algorithm implemented in the integration block can be formulated as follows:

1) Start with an empty matching.
2) Add all multicast edges.
3) Add all nonconflicting unicast edges.

As a consequence, the remainder always contains only unicast edges.

D. Remainder-service policy

As noted above, if a remainder is produced in a time slot, it is important to ensure that all the edges it contains are eventually served. This can be done according to different policies, the simplest one being to serve all of them in the next time slot. As these edges are part of a matching, they do not conflict with each other. In addition, the resources they claim are known and can be reserved to avoid further contention.

At every time slot, new unicast and multicast requests are issued. The request filter drops all those that involve inputs and outputs that are needed to serve the remainder produced in the preceding time slot and submits the others to the corresponding scheduler. Accordingly, the integration block issues grants for the edges in the remainder as well as for those in the current matching. A new remainder is produced and fed back to the request filter for the next time slot.

An important property of the scheme is that, as a consequence of filtering unicast requests, the remainders produced in two consecutive time slots are disjoint, i.e., have no inputs or outputs in common. This is crucial for fairness because it ensures that all switch resources eventually become available for scheduling. Reserving resources for the remainder does not persistently preclude access to any input or output.

We expect this combination of integration and remainder-service policy to achieve good link utilization. The resources allocated to the remainder are fully utilized, and any remaining resources can be assigned to either unicast or multicast. The integration block preserves the matching produced by the multicast scheduler, but tries to enlarge it by adding unicast edges.

III. Simulation Results

We have studied the performance of a system employing the FILM scheme by simulation. In particular, we observed the total throughput as well as the individual throughputs of unicast and multicast traffic as the fraction of multicast traffic (MCF) grows from 0 (unicast only) to 1 (multicast only). Ideally, the throughput achieved by each traffic type should be equal to the corresponding share of the output load, and the total should be 100%.

The simulated system is an 8×8 switch with infinite buffers at the inputs. The unicast scheduler uses iSLIP with three iterations, and the multicast scheduler uses WBA. Simulations run for 1 million cell times, and results are collected after a quarter of the total simulation time has elapsed.

Cells are generated according to an i.i.d. Bernoulli process, i.e., every input port receives a cell with probability ρ, equal to the input load. Each cell has a probability P of being a multicast cell. The fanout of multicast cells is uniformly distributed between 2 and 8. Traffic is uniform, i.e., all outputs have the same probability of being the destination of a unicast cell or of belonging to the fanout of a multicast cell.

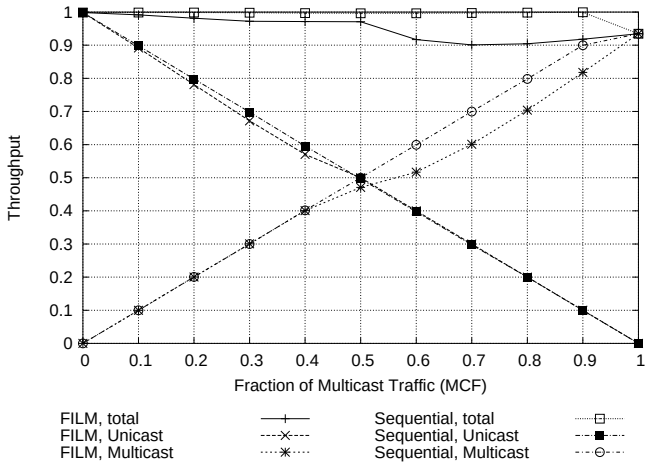

Fig. 3. Performance of the FILM integration scheme

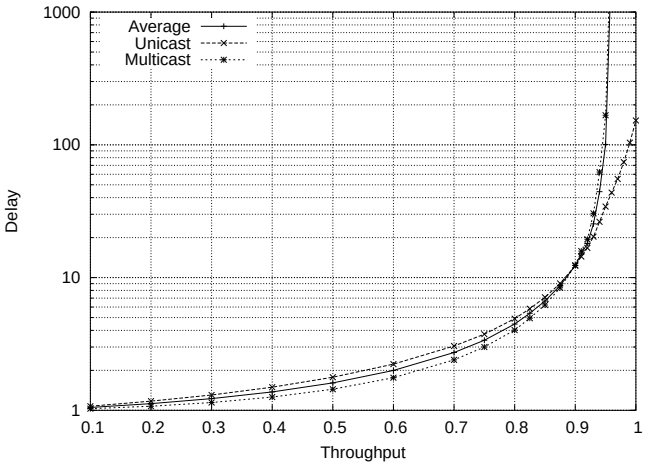

Fig. 4. Delay vs. throughput curves of the FILM integration scheme

Note that, under these conditions, when the switch is loaded with multicast traffic only, the multicast scheduler limits the maximum switch throughput to approximately 0.93, whereas it is 1.0 when only unicast traffic is present.

The total load on the switch is $\rho(P\overline{F} + (1-P))$, where $\overline{F}$ is the average fanout. In our case, $\overline{F} = 5$, whereas P and ρ are varied to obtain the desired multicast load on the switch while keeping the total load equal to 1.

Figure 3 shows the throughput achieved by FILM with the integration and remainder-service policies described in the preceding section. The performance of the sequential scheme, which is close to ideal, is also shown for reference.

The total throughput achieved by our scheme is always higher than 0.9. Unicast throughput exhibits very little degradation (on the order of a few percent) when it is the predominant traffic type, and it achieves ideal performance when multicast traffic predominates. However, multicast throughput progressively decreases with respect to output load as MCF grows from 0.4 to 1.0. The worst case is MCF = 0.7, when multicast throughput is 0.6 instead of 0.7. This also corresponds to the point at which the overall throughput is at its minimum (0.9).

Figure 4 shows the delay experienced by unicast and multicast cells as a function of the throughput when MCF = 0.5, i.e., when each traffic type is responsible for half of the output load. The unicast curve is bounded for any value of the total throughput, whereas the multicast curve saturates when it approaches 1.0.

IV. Enhanced Remainder-Service Policy

Although the scheme presented above provides overall good performance and is quite simple, it has a drawback: it penalizes multicast traffic most, especially when it is predominant.

Multicast performance is limited because, at every time slot, some switch resources are used to discharge the remainder. Although it is essential to eventually serve all edges that are not selected in the merge, it is not necessary to do so immediately. Thanks to the disjoint remainder property, it is possible to *accumulate* the remainders produced in consecutive time slots and serve the individual edges when the conditions are most favorable. The remainder-service policy identifies which edges should be served at every time slot and filters the corresponding multicast requests. Unicast requests, in contrast, are always filtered using all the accumulated edges to obtain disjoint remainders.

A good policy should be able to serve the edges in the remainder rapidly and at the same time cause as little disruption as possible to the flow of multicast cells. We propose an *enhanced* policy that serves a remainder edge if it uses

1) an input not requested by multicast OR
2) an output not requested by multicast OR
3) an input that discharged a multicast cell in the preceding time slot.

The first two rules obviously aim at improving integration: if it is possible to use a resource that would otherwise remain idle, it is desirable to do so. In this case the cost of serving a remainder edge is to make one output (first rule) or one input (second rule) unavailable to multicast.

The third rule instead stems from the general observations on multicast scheduling found in [10]. The scheduler tends to favor cells that contend with few others. Cells that have just advanced to the HOL still have their full, usually large, fanout and cause many conflicts. They are unlikely to receive much consideration, so postponing their scheduling should not significantly affect the quality of the matching. This rule is particularly important because it enables fairness: the multicast scheduler guarantees that the HOL cell at any input will be served in finite time; consequently, the inputs becomes available to serve remainder edges. Many algorithms (such as TATRA, WBA and mRRM [12]) ensure that at least one multicast cell is fully discharged at every time slot.

Figure 5 shows the performance of FILM when the enhanced policy is used, under the same conditions as assumed in Section III. The benefits on multicast traffic are evident: Throughput increases when ($0.4 < \text{MCF} < 0.9$) and closely tracks the output load up to MCF = 0.7. Unicast, on the other

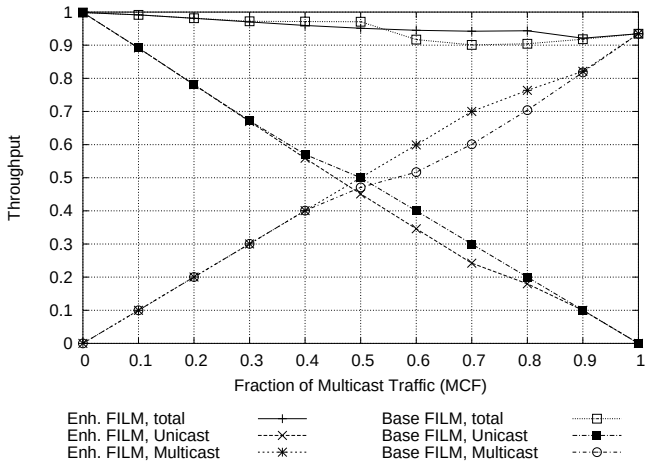

Fig. 5. Performance of FILM with enhanced remainder-service policy

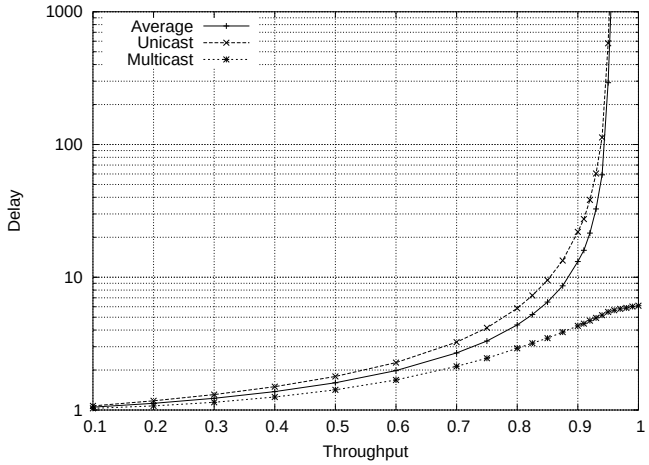

Fig. 6. Delay vs. throughput curves of the enhanced FILM integration scheme

hand, shows a moderate decrease in the same range. In the worst case (MCF = 0.7), the difference with respect to the output load is slightly less than 0.06. Overall throughput is noticeably increased when multicast predominates, whereas it shows little degradation when both traffic types are equally active.

Figure 6 shows the delay vs. throughput curve for this situation (MCF = 0.5). Multicast experiences very low delay, seeming to be almost insensitive to the presence of unicast. Unicast delay instead saturates when the total throughput is approximately 0.95.

V. Implementation Complexity

In this section we discuss some implementation aspects of the FILM scheme to assess its complexity.

A. Integration policy

As the integration policy always prioritizes multicast over unicast, its implementation is quite straightforward. From the output of the multicast scheduler, it is immediately known which inputs and which outputs are used by the multicast matching. This information ($2N$ bits) in turn determines whether an edge in the unicast matching is to be interpreted as part of the integrated matching or of the remainder. In the former case, grants are released immediately, in the latter the information is buffered for subsequent time slots. The remainder can be stored using N registers, each $(\log_2 N + 1)$ bits wide.

B. Base remainder-service policy

The request filter needs to know which inputs and outputs are used by the remainder edges so that it can drop the corresponding requests. This information is available at the integration block and can be carried to the request filter with a channel that is $2N$ bits wide. Filtering a request for an input-output pair simply translates to ANDing it with the negated values of the corresponding signals.

C. Enhanced remainder-service policy

When the enhanced policy is used, the request filter needs more information and performs more complex operations. It needs to know exactly which edges are in the remainder, not only which inputs and outputs are taken. This means that $N(\log_2 N + 1)$ bits must be transferred from the integration block. The information about which inputs discharged a multicast cell in the preceding time slot consists of N bits and can be maintained by the queue managers. Finally, the information about which inputs and which outputs are being requested by multicast ($2N$ bits) is readily available as it can be derived from the requests themselves.

Unicast requests are filtered using all edges in the remainder as before, whereas multicast requests are now filtered depending on which remainder edges are served. This information is produced at the request filter block by ORing the signals corresponding to the three conditions that grant service to an edge. The integration block also needs to know which edges are served, as it has to issue the appropriate grants. As the remainder edges are part of a matching, only N bits need to be transferred from the request filter to the integration block.

As a final remark, note that all the operations described above can be performed in parallel and implemented using combinational logic only.

VI. Conclusions

We have shown how *fair integrated* scheduling of unicast and multicast traffic can be achieved by first scheduling the two traffic types separately and then arbitrating among the results for access to the switching fabric. The integration block combines the matchings produced by the two schedulers, producing an integrated matching and a remainder. Edges in the remainder must receive service in subsequent time slots to prevent starvation. The policy used to select which remainder edges to serve has an impact on the overall performance of the scheme and on the service received by the two traffic types. The first policy we have proposed is extremely simple and performs well, but tends to penalize multicast. The second is more sophisticated and is able to serve remainder edges, resulting in only minimal interference with the flow of multicast cells. It

leads to a very high overall performance and an almost ideal treatment of multicast traffic, at the cost of some additional complexity.

In future work, we will investigate policies that have different goals or have additional advantages over those proposed here. In particular, it would be desirable to have more fine-grained control over the partitioning of resources between unicast and multicast.

REFERENCES

[1] Y. Tamir, G. Frazier, "High performance multi-queue buffers for VLSI communication switches", in *Proc. 15th Ann. Symp. Comp. Archi.*, Jun. 1988, pp. 343-354.

[2] M. Karol, M. Hluchyj, S. Morgan, "Input versus output queueing on a space division switch", *IEEE Trans. Commun.*, vol. 35, no. 12, pp. 1347-1356, Dec. 1987.

[3] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches", *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188-201, Apr. 1999.

[4] H. Chao, J. Park, "Centralized contention resolution schemes for a large-capacity optical ATM switch," in *Proc. IEEE ATM Workshop*, Fairfax, VA, May 1998, pp. 11-16.

[5] N. McKeown, A. Mekkittikul, V. Anantharam, J. Walrand, "Achieving 100% throughput in an input-queued switch", *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1260-1267, August 1999.

[6] J. Hui, T. Renner, "Queueing analysis for multicast packet switching", *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp. 723-731, Feb./Mar./Apr. 1994.

[7] Z. Liu, R. Righter, "Scheduling multicast input-queued switches", *J. Scheduling*, vol. 2, pp. 99-114, 1999.

[8] M. Andrews, S. Khanna, K. Kumaran, "Integrated scheduling of unicast and multicast traffic in an input-queued switch", in *Proc. IEEE INFOCOM '99*, New York, NY, Mar. 1999, vol. 3, pp. 1144-1151.

[9] M. Ajmone Marsan, A. Bianco, P. Giaccone, E. Leonardi, F. Neri, "Multicast traffic in input-queued switches: Optimal scheduling and maximum throughput", *IEEE/ACM Trans. Networking*, vol. 11, no. 3, pp. 465-477, Jun. 2003.

[10] B. Prabhakar, N. McKeown, R. Ahuja, "Multicast scheduling for input-queued switches", *IEEE J. Sel. Areas Commun.*, vol. 15, no. 5, pp. 855-866, Jun. 1997.

[11] C. Minkenberg, "Integrating unicast and multicast traffic scheduling in a combined input- and output-queued packet-switching system", in *Proc. ICCCN 2000*, Las Vegas, NV, Oct. 2000, pp. 127-234.

[12] N. McKeown, B. Prabhakar, "Scheduling multicast cells in an input-queued switch", in *Proc. IEEE INFOCOM '96*, San Francisco, CA, Mar. 1996, vol. 1, pp. 271-278.