Fair Scheduling for Input Buffered Switches

Nan Ni
IBM Corporation
Austin, Texas 78758

Laxmi N. Bhuyan
Computer Science and Engineering
University of California
Riverside, CA 92521

Abstract—**Input buffered switch architecture has become attractive for implementing high performance switches for workstation clusters. It is challenging to provide a scheduling technique that is both highly efficient and fair in resource allocation. In this paper, we first introduce an iterative fair scheduling(*i*FS) scheme for input buffered switches that supports fair bandwidth distribution among the flows and achieves asymptotically 100% throughput. We then apply the idea of fair scheduling to switches with multicasting capability and propose an *m*FS scheme which allocates bandwidth to various flows according to their reservations. We show that *m*FS produces throughput comparable to the existing schemes while distributing the bandwidth as per the given reservations. Extensive simulation results are presented to validate the effectiveness of our proposed schemes.**

Keywords— **Switching, input buffered switch, scheduling, multicast switching, fair bandwidth allocation.**

I. Introduction

Clustered computers are not only suitable for parallel computing but they can be used for multimedia data processing, where quality of service(QoS) is important. Large clusters and high-performance applications require lower latency, higher throughput, guaranteed bandwidth and additional features in the network not offered by standard LAN technology. As a result, the growth in the use of clustered systems has put increasing demands on switches, both for high speed and QoS. These switches transfer packets from a set of input links to a set of output links leading to the next node. Buffering is always needed for switches to temporarily store packets that can not be handled by the switch forwarding engine. Buffering can be done either at the input side or at the output side of a switch. Although output buffering can achieve better throughput, it is known to suffer from poor scalability[4], [20], [10]. Accordingly, most high performance switches employ input queueing technique with crossbar backplane[15], [18].

Input queueing, however, is subject to head of line(HOL) problem with maximum throughput of 58.6% using FIFO input queue[10]. Solution has been found which constructs separate queues at each input so that a packet will not be blocked by packets going to other destinations. Major issues in the input buffered switch architecture design include buffer management and scheduling. We have studied various queueing and buffer allocation schemes in our previous research[4], [20]. In this paper, we seek to address the issue of fairness in scheduling(i.e., input to output matching) for quality of service. Figure 1 gives a block diagram of the cell-based switch architecture we consider throughout the paper. A cell is a fixed-size packet which is the unit of message flow control in a cluster. The switch consists of four components: input ports, output ports, crossbar switching fabric and a scheduler. At any given switch cycle(slot), at most one cell from each input can be routed to the output side and each output can only accept at most one cell. As a result, the scheduling is more intricate than that for output queued switch. Most existing work on scheduling for input buffered switch attempts to achieve high throughput by looking for maximum bipartite matching between inputs and outputs. Schemes such as PIM[1], *i*SLIP[13] and Shakeup[6] repeatedly search for matches at each time of scheduling. Some approaches can achieve 100% throughput asymptotically[14].

This research has been supported by NSF Grants CCR 9622740 and CCR 9810205. This work was done while the authors were at Texas A&M University.

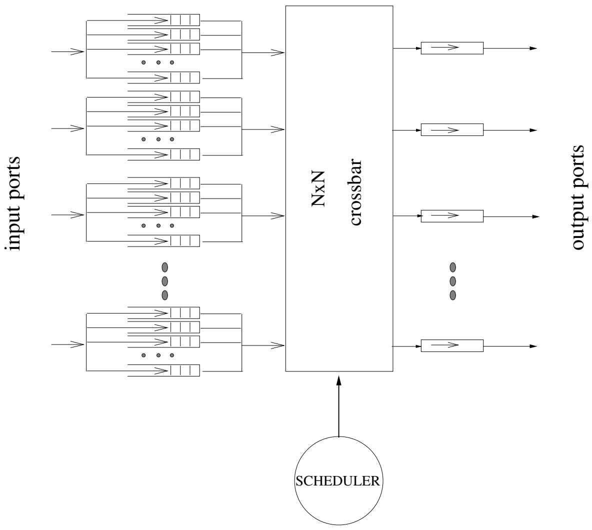

Fig. 1. Block diagram for an input buffered switch architecture

In addition to the demand for high throughput, there is an increasing need for supporting applications with diverse performance requirement where fairness in resource allocation is important. However, these two goals are usually contradictory. Due to the fact that only a single cell can be transmitted from each input to the crossbar in any given slot, cells forwarded based on maximal set of input-output match may not coincide with those satisfying the resource requirement. On the other hand, passing cells conforming to resource requirement may result in under utilization of the switch. The aim of this work is to find a scheduling technique supporting highly efficient switching as well as fair link bandwidth sharing. According to [3], bandwidth allocation of a link is fair if for each flow, the received bandwidth is proportional to its share of reservation. We know that the fluid flow queueing(or Generalized Processor Sharing(GPS)[17]) algorithm which sends packets in a bit-by-bit round robin fashion is absolutely fair but unrealistic.

Numerous fair queueing algorithms have been developed to approximate the GPS algorithm in units of packets(see [25] for a survey). However, most work on fair queueing has been conducted in the context of output queueing due to its conceptual simplicity. Situation becomes complicated in the case of input buffered switch where flows not only fight for output link bandwidth but also have to compete to access the crossbar. Nevertheless, in order to achieve high performance as well as to provide quality of service, it is imperative to maintain fairness in input buffered switch scheduling. Very limited work has been done in this aspect[23], [11], [24].

In the first part of this paper, we propose an iterative fair scheduling(*i*FS) scheme which provides high throughput, low latency as well as fair bandwidth distribution among the contesting flows. As with other iterative approaches, *i*FS tries to increase the number of input-output matches during each iteration, but the major difference is that instead of picking a cell from each input in a probabilistic([1], [6]) or round robin([13], [22]) fashion, a cell is chosen based on the bandwidth requirement of the flows. In this way, we are able to allocate bandwidth to various flows in proportion to their reservation and prevent misbehaved flows from taking advantage at the expense of others.

In the second part, we apply the fair scheduling idea to input buffered switches with multicasting capability. The two disciplines for multicasting are "one-shot", where all the copies of a multicast cell must be sent in the same slot to the output side, and "fanout-splitting" in which copies can be delivered to the outputs over a number of cycles. Both the schemes have been studied extensively[16], [9], [21], however, to the best of our knowledge, the issue of fair resource allocation in multicasting is not addressed beyond the starvation prevention or the priority to heavy loaded queues[21]. In this work, we propose a multicasting fair scheduling(*m*FS) scheme for input buffered switch where for flows with conflicting outputs, the allocated bandwidth is proportional to their corresponding bandwidth reservations.

The rest of the paper is organized as follows. Section II gives an overview of the related work on scheduling for input buffered switches. In Section III we propose the iterative fair scheduling(*i*FS) for unicast traffic. Extensive simulation results are presented in Section IV. The multicast version of the fair scheduling(*m*FS) scheme is given in Section V. Finally, Section VI concludes the paper.

II. Related Work

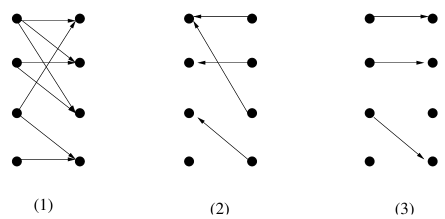

Fig. 2. 3-step procedure in iterative matching

An iterative scheduling algorithm essentially consists of three major steps in each iteration, as illustrated in Figure 2.

1. *Request stage*: An input may have several requests. Each unmatched input sends requests to the outputs for which it has cells for.
2. *Grant stage*: There may be several requests to an output. Each unmatched output chooses one from several received requests and sends a grant signal to one of the inputs.
3. *Accept stage*: Each unmatched input may receive grant signals from several outputs. Upon receiving grant signals, each input sends accept signal to only one of the outputs offering the grants.

Probabilistic Iterative Matching(PIM) suggested by Anderson *et al.*[1] is the first switch scheduling algorithm that employs an iterative approach. At grant stage, each output sends out a grant signal to a randomly selected requesting input. Again at accept stage, each input accepts one output from several grant signals at random. It takes an average of $O(logN)$ iterations for the PIM to converge. The consequence of random selection in grant and accept stage of PIM is that an input could possibly end up in not being served for a long time. Thus, this scheme is not starvation free. Moreover, implementation of random selection among the member of a time-varying set is not an easy task.

The *i*SLIP scheme proposed by McKeown [13] uses rotating priority(round-robin) arbitration to schedule active inputs and outputs in turn. A grant pointer is kept for each output to track the input with the highest priority. Similarly, there is an accept pointer for each input which tells the output with the highest priority. Whenever a match is found, the corresponding grant/accept pointers are incremented(modulo the number of ports). Compared to PIM, *i*SLIP is simpler in implementation[8] and achieves higher throughput[13].

Matches are irrevocable in both PIM and *i*SLIP, which means that later iterations can only add upon previously made matches but can not change them even if better matches can be found. To escape this "local maximum", Goudreau *et al.* offer a Shakeup technique in [6] where each unmatched input is allowed to force a match for itself randomly even though an existing match has to be knocked off. In other words, Shakeup attempts to find the global maximum, but the feasibility of its implementation in real system is unknown because it takes more iterations to converge.

The issue of fair resource allocation is not considered in the above mentioned approaches. Iterative schemes that do address the fairness issue include statistical matching [1] and Weighted Probabilistic Iterative Matching(WPIM)[24]. Statistical matching is similar to PIM, except that the matching process is now initiated by the outputs, each generating a grant signal to a randomly selected input based on the reserved proportion of bandwidth. It can happen that an input is picked when its queue is empty and the switch is poorly utilized. WPIM is also built upon PIM where based on the reservation, every input flow is assigned a quota that can be used in a frame of a constant number of slots. During each frame, flows that have not reached their quotas secure equal share of bandwidth by random selection as done in PIM. To accomplish this, an additional masking stage is added to the 3-step procedure to exclude those inputs that have consumed their quotas in the current frame.

Scheduling fairness is also addressed in other work under slightly different assumptions. Stephens and Zhang[23] considered a switch where inputs and outputs are decoupled so that each output independently reads from any input with a packet for it. Li and Ansari gave end-to-end delay bound provided that incoming flows conform to the (r, T) traffic model[11].

For multicast scheduling, the one-shot scheme is simple but not work conserving. If any of the copies of a cell loses in the contention, none of them can be sent. With fan-out splitting approach, only those copies that are unsuccessful will continue to compete for output ports in the next slot. Random selection techniques have been studied for both one-shot and fan-out splitting schemes[16], [9]. At each slot, contending input cells(for one-shot scheme) or copies of the cells(for fan-out splitting scheme) are chosen randomly to be transmitted to the outputs. The TATRA scheme proposed in [21] is of type fan-out splitting. The idea is adopted from the popular block-packing game Tetris. Every multicast cell can be imagined as a block in Tetris, which is an amalgamation of smaller blocks corresponding to its copies to the outputs. A single FIFO queue is maintained at each input, and whenever a cell is released from an input, its component blocks are scattered to the columns corresponding to the outputs. At each time slot, the blocks at the bottom row are passed over to the output side and the other blocks drop one row down. A head-of-line cell is guaranteed to be forwarded in no more than N slot times(N is the number of input ports). TATRA is work conserving and has proved to be better than random selection schemes[21]. WBA, a weight-based variation of TATRA, gives weight to each cell based on its age and the number of fan outs. As far as we know, there is no hardware fair scheduling algorithm for multicasting.

III. Iterative Fair Scheduling Scheme

In this section, we first introduce a definition of fairness in input buffered switch scheduling and then propose an iterative fair scheduling(*i*FS) scheme that can be used to achieve fair bandwidth allocation.

Let $f(i, j)$ denote the jth flow from input i. It goes to output $d_{i,j}$ and reserves a bandwidth of $B_{i,j}$. The number of flows from input i is n_i. Let $t_{i,j}(t_1, t_2]$ be the amount of traffic(in bits) reaching the output from flow $f(i, j)$ in time interval $(t_1, t_2]$. We say that two flows $f(i_1, j_1)$ and $f(i_2, j_2)$ are in contention if $i_1 = i_2$ or $d_{i_1,j_1} = d_{i_2,j_2}$. Note that we do not define precisely what a *flow* is because we would like our *i*FS scheme to be applied in a broad sense: Depending on the desired granularity of QoS and scalability concern, a flow can be as fine as a TCP connection or as coarse as a class of aggregated connections with similar bandwidth requirement.

On the analogy of the definition of the GPS server for a single shared resource in [17], we consider a scheduling scheme for input buffered switches to be fair as follows.

Definition 1: For any two back logged flows $f(i_1, j_1)$ and $f(i_2, j_2)$ that are in contention, a scheduling scheme is fair in $(t_1, t_2]$ if

$$\frac{t_{i_1,j_1}(t_1, t_2]}{t_{i_2,j_2}(t_1, t_2]} = \frac{B_{i_1,j_1}}{B_{i_2,j_2}}$$

The discrepancy of this definition from the one of GPS[17] is that here the flows are contesting to access the crossbar as well as to access the output links. In other words, we are trying to allocate correlated resources because a cell must first make its way out of the input buffer before going to the intended output.

Nevertheless, we find that the existing fair queueing algorithms, which have been successful in allocating a single shared resource, can be applied here to facilitate our effort to support fair bandwidth distribution. In a fair queueing algorithm, a virtual system time $V(t)$(corresponding to the number of rounds made till time t in GPS) is maintained. Every incoming packet is assigned a virtual starting time and a virtual finishing time depending on its bandwidth requirement. Virtual starting time and virtual finishing time denote the virtual time when a packet should begin and finish sending if GPS were used. The transmitting order is then regulated according to non-decreasing starting time[7], non-decreasing finishing time[3] or a combination of both[2]. Naturally, flows with larger bandwidth reservations will take greater proportion of bandwidth since they tend to have smaller virtual starting time(or finishing time).

Our basic idea of *i*FS is to enhance the iterative approaches described in Section II by giving out grant signals from the individual outputs based on virtual time and then try to resolve input contention in the accept stage.

For each output link we maintain a fair queueing engine, which assigns a virtual time to every incoming cell based on bandwidth reservation of the flow. Note that the arriving cells are queued in the input buffer first-in-first-out on a per flow basis. This is required for implementing a fair queueing algorithm where each output must keep track of the active flows to compute the virtual time. It is different from other iterative approaches where input queues are arranged on per output basis. Also note that by applying fair queueing algorithm, each output has the knowledge of the active flows destined to it, so we can initialize the scheduling from the outputs and save the request stage in the common 3-step procedure(Section II).

In each iteration, the first step in our scheme is for every unmatched output to independently send a grant signal to one of the unmatched inputs which has the cell with the minimal virtual time corresponding to that output. Such a cell is marked as a "candidate" with respect to its input. It may so happen that an input receives multiple grants from different outputs and have multiple candidates, which means that several flows from this input have minimal value of virtual time with their relevant outputs. Then, each unmatched input selects among its candidates a cell(called "winner") with the oldest age at the switch and sends accept signal to its desired output. That is, in accept stage, an input resolves the contention on a first-come-first-serve(FCFS) basis. The justification is that since these candidate cells are from the same upstream node, the fact that the cell arrives first among the contending cells implies that it has the smallest virtual time among them at the previous node and naturally should be the first one to depart from the current node.

In summary, the *i*FS scheme can be formalized as the following:

- Initially, all inputs and outputs are considered as unmatched and none of the inputs have any candidates.
- Then in each iteration:

1. *Grant stage*: Each unmatched output selects a flow with the smallest virtual time for its head-of-line cell and marks the cell as a candidate for the corresponding input. Grant

signal is then given to the input.

2. *Accept stage*: Each unmatched input examines its candidate set, selects a winner according to age and sends an accept signal to its output. The input and output are then considered as matched. Reset the candidate set to empty.

- At the end of each switch cycle, the winning cells are transferred from the input side to the output side.

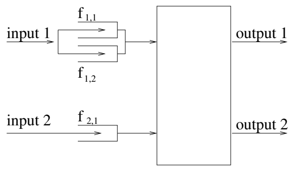

Fig. 3. Fair queueing ordering may not be strictly followed in *i*FS

Notice that although a fair queueing algorithm is applied in *i*FS, the access order to an output ruled by the fair queueing algorithm is not strictly followed due to performance concern. Consider the example in Figure 3 where flows $f(1,1)$, $f(1,2)$ and $f(2,1)$ are going to output 1, 2 and 1, respectively. Suppose that at a certain time t, the head-of-line cell from $f(2,1)$ has greater virtual time than that from $f(1,1)$. In the first iteration, output 1 gives grant to flow $f(1,1)$ and output 2 gives grant to $f(1,2)$ according to fair queueing algorithm. Also assume that head-of-line cell of $f(1,2)$ has older age than that of $f(1,1)$. Hence, input 1 sends an accept signal to output 2 and both input 1 and output 2 are marked as matched. In the next iteration, output 1 finds flow $f(2,1)$ with the smallest virtual time because $f(1,1)$ from input 1 has been excluded. As a result, input 2 and output 1 are matched and we see that the head-of-line cell from flow $f(2,1)$ is transmitted before that of flow $f(1,1)$ even though the latter has a smaller virtual time. This "out-of-order" transfer actually improves the throughput, because otherwise output 1 would be left idle in current cycle. Fortunately, this violation has only instant effect and will not yield unfairness. Continue with the example. In the next time of scheduling, the cell from $f(1,1)$ still has the smallest virtual time for output 1. Flow $f(2,1)$ gets "penalized" since the virtual time for its head-of-line cell at this point is even greater.

Considering other iterative scheduling schemes, we see that grants in *i*SLIP are given in round-robin manner without any respect to bandwidth reservation. WPIM guarantees fair bandwidth sharing, but in a much coarser granularity. Before running out of quota in WPIM, every flow has equal access to the bandwidth. Consequently, the flows with larger reservations get more bandwidth only after other flows run out of their quotas, which usually happens towards the end of a frame. If we look at the bandwidth distribution within a frame, the bandwidth share is disproportional. Statistical matching also has only two stages. However, since grants are given by outputs to inputs randomly, it is possible that an input port is selected to receive grant signal when its queue is empty. It has been shown in [24] that statistical matching has inferior performance than WPIM. Unlike Shakeup where matches in prior iterations can be modified, matched input-output pairs are ruled out in later iterations in *i*FS. Although Shakeup is attractive theoretically, its feasibility for high performance switch is unclear because one "knock-off" of an existing match could possibly trigger a chain of "knock-offs" which would take more iterations to converge to global maximum matches.

The issue we do not consider in detail here is the implementation feasibility. Efficient implementation of N fair queueing engines is crucial to the *i*FS for an $N \times N$ switch. It is to our advantage that the switch under discussion is cell based. The calculation of virtual time can be simplified due to fixed packet length and the fair queueing engine is boiled down to an engine for weighted round robin instead. In comparison, the switch proposed in [23] needs $3N$ fair queueing engines to deal with variable length packets for an $N \times N$ switch. A second advantage is that, unlike PIM, WPIM or Shakeup, no random number generator is needed for *i*FS, which again greatly reduces the implementation complexity.

IV. Performance Evaluation of *i*FS

Following the criteria in [19] for assessing a resource allocation scheme, we evaluate the proposed *i*FS in two aspects: efficiency and fairness. The principal metrics for efficiency are throughput and delay. A cell based simulator is developed and the simulations are conducted with the assumption of infinite buffer size. The offered load refers to the probability that a cell arrives at an input in a given slot and the underlying fair queueing algorithm for *i*FS is self-clocked[5].

We begin by evaluating the throughput and the average delay of *i*FS under benign i.i.d. Bernoulli traffic, where at any given slot, a cell arrives with the probability determined by the offered workload. The simulation is performed on a 16×16 switch with 16 flows per input each destined for a different output, for a total of 256 flows. In the first scenario, each input port sends cells with destinations uniformly distributed among all the output ports. Figure 4 shows the delay versus the offered load for *i*FS, *i*SLIP and WPIM with the number of iterations equal to four. The *i*SLIP is known to offer the lowest delay for an input buffered switch. The delay using output buffered switch is also shown for comparison. Under this circumstance, we observe that the average cell delay for *i*FS is almost identical to *i*SLIP and WPIM. They are all very close to output buffered switch, which is the lower bound for the delay. Hence, *i*FS is capable of achieving asymptotically 100% throughput for uniform traffic.

Next we consider a non-uniform Bernoulli traffic model as used in [24]. The assumption is that four of the switch ports are connected to servers and the remaining twelve to clients. Each client sends 10% of its generated traffic to each of the four servers, and the remainder is uniformly distributed among the other clients. Similarly, each server directs 95% of its traffic to the clients and the remaining 5% to the other servers. Figure 5 indicates that *i*FS is very close to *i*SLIP and WPIM in terms of average cell latency and can reach a throughput of 78.5%.

Now we turn to examine the effectiveness of *i*FS to support the fair bandwidth sharing when a link is overloaded. This time, we simulate a 4×4 switch so that we can plot the results with clarity. Assume that every input has four flows each going to a different output. Without loss of generality, let the jth flow from link i go to output j and denote it as $f(i,j)$ following the notation in Section III. Also assume that the flows to output 1, $f(1,1)$, $f(2,1)$, $f(3,1)$ and $f(4,1)$, have reserved 10%, 20%,

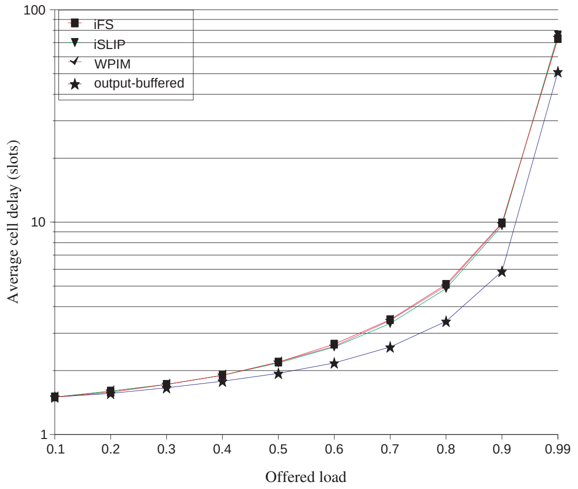

Fig. 4. Performance of *i*FS, *i*SLIP and WPIM under uniform traffic

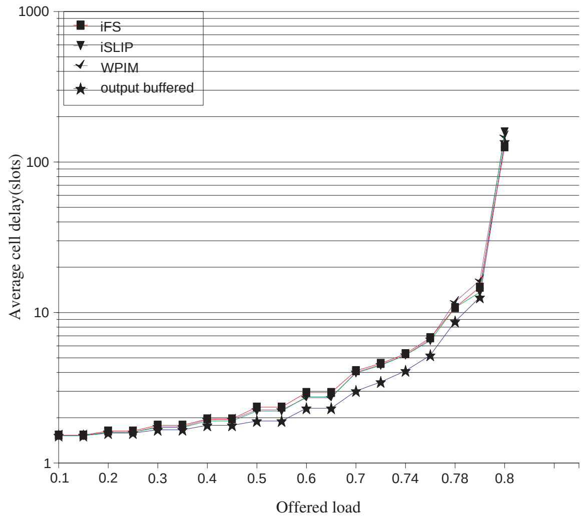

Fig. 5. Performance of *i*FS, *i*SLIP and WPIM under non-uniform traffic

30% and 40% of the bandwidth, respectively. But they always maintain the same actual arrival rate. Others are background flows with a load of 5% each. We vary the input rate of the flows to output 1 and plot the received bandwidth share in Figure 6 and 7 using *i*SLIP and *i*FS scheduling. For workload under 25%, the throughput for every flow is able to keep up with the input workload for both the schemes. However, for workload beyond 25%, all the four flows are still treated equally in *i*SLIP, and therefore they obtain the same share of bandwidth(each 25%) despite of the variance in bandwidth reservation. The *i*FS, on the other hand, is observed to differentiate the flows according to the promised share when the load is greater than 25%. Ill-behaved flows are prevented from influencing the well-behaved ones. For load beyond 40%, each flow receives its allocated bandwidth. Let us look closely at the sharing when input load is between the range of 25% and 40%. At 30%, for instance, flow $f(4,1)$ does not consume its share of 40% of bandwidth. The unused part is distributed to flow $f(1,1)$ and flow $f(2,1)$ so that they acquire a larger fraction of bandwidth than their reservations. Flow $f(1,1)$ receives 13.3%(versus a reservation of 10%) and flow $f(2,1)$ receives 26.6%(versus a reservation of 20%). Such behavior also conforms to the fairness requirement in [3], which states that the unused portion should be assigned equally to other active flows.

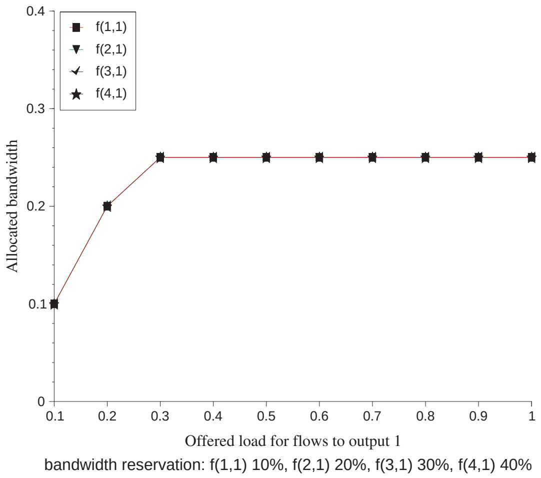

Fig. 6. Received bandwidth using *i*SLIP

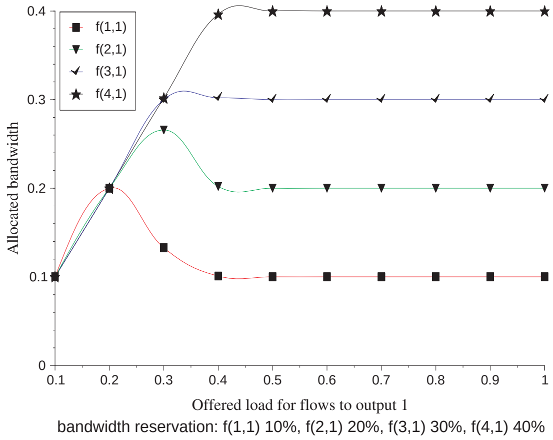

Fig. 7. Received bandwidth using *i*FS

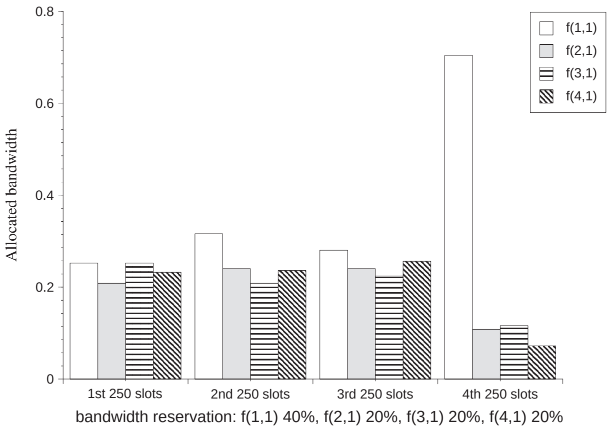

Fig. 8. WPIM: bandwidth distribution within a frame of 1000 slots

WPIM scheme also complies with the bandwidth requirement of each flow by restricting the number of transmitted cells during a frame within a limit determined by its reservation. Accord-

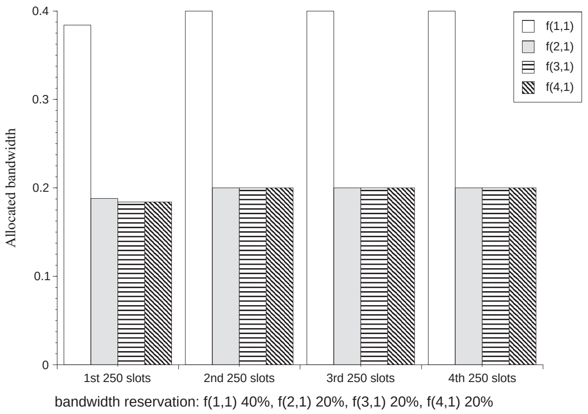

Fig. 9. *i*FS: bandwidth distribution within a frame of 1000 slots

ingly, bandwidth requirement can be met and the well-behaved flows are protected. However, a careful study reveals the drawback of WPIM whose mechanism rules that link bandwidth is evenly distributed among existing flows until some use up their quotas. Flows running out of their quotas are then excluded from accessing the link in the current frame, and the bandwidth is again allocated equally among the remaining flows. Let us inspect the flow with the largest bandwidth reservation. Under WPIM, this flow shares the link bandwidth equally with all the other flows at the beginning of each frame. Its share increases gradually as the quotas for other flows are exhausted. As an example, consider the four flows going to output 1 but with bandwidth reservations of 40%, 20%, 20% and 20% this time. Frame length is taken as 1000 slots as in [24]. We observe that every flow gets its fair share by the end of a frame under both WPIM and *i*FS. But if we break a frame into four 250-slot subframes, we notice that the bandwidth distribution for WPIM is not fair, as depicted in Figure 8. During the first 750 slots, bandwidth is almost equally distributed(about 25% each) regardless of the reservation. Towards the end of the frame, flows with less reservation($f(2,1)$, $f(3,1)$ and $f(4,1)$) have used up their quota. Flow with the highest reservation($f(1,1)$) then consumes almost all the capacity in the last 250 slots. Therefore, WPIM provides bandwidth guarantee at coarse granularity of 1000 slots. The *i*FS, on the other hand, can provide fair sharing both at coarse and fine grain levels. The received bandwidth in *i*FS complies with the reservations even within subframes as illustrated in Figure 9. Some may argue that if the length of a frame in WPIM is small enough, it is able to support fair bandwidth sharing in finer granularity. But we find that the WPIM scheme is "memoryless" regardless of the frame length since the scheduler does not have the information about the flow history except in the current frame. Imagine a scenario where the flow with the greatest reservation actually sends very little in the beginning, it cannot get "compensation" later when it raises the traffic. If it has nothing to send in prior frames, it loses its quota in those frames for good. Even though memoryless is necessary to enfore fairness, it is not desirable for frames that are too short. Some fair queueing algorithms have been devised to compensate such a flow to a certain extent[12].

As for all the iterative approaches, the number of iterations required for convergence is a constraining factor because all the iterations must be done within a single switch cycle. In the case of 16×16 switch, 16 iterations are needed in the worst case. Figure 10 shows the effect of the number of iterations on the average cell latency under uniform traffic. We can see that with two iterations, a throughput of over 90% can be achieved and four iterations are adequate to obtain a throughput of nearly 100%. This is consistent with the findings in [1], [24] and [13] that $log(N)$ iterations are enough for the algorithm to converge.

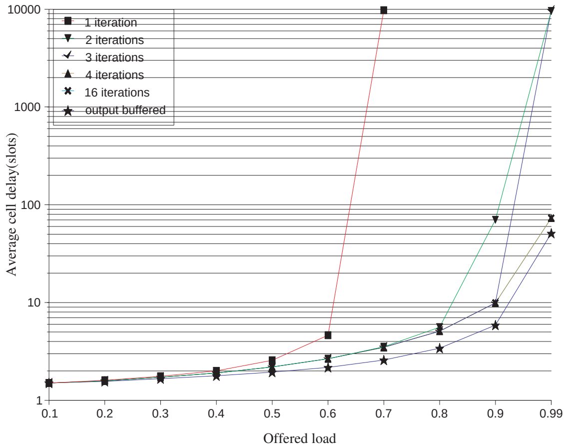

Fig. 10. Latency versus the number of iterations for *i*FS

In summary, the evaluation using uniform and non-uniform synthetic traffic models indicates that the *i*FS scheme is very promising in supporting fair bandwidth distribution as well as maintaining high overall throughput.

V. Fair Scheduling for Multicasting

The issue of fairness in multicasting scheduling for input buffered switch has not been addressed in the literature. In this section, we extend our idea of fair scheduling to input buffered switches with multicasting capability. The multicast fair scheduling(*m*FS) scheme is proposed in Section V-A and is compared with TATRA[21] in Section V-B.

A. The mFS scheme

Same as in Section III, incoming multicast cells are queued in FIFOs based on the flow, but the scheduling for multicasting packets is more complicated than its unicast counterpart because every flow is transmitting cells to multiple outputs and the fan out set(the aggregation of outputs for a multicasting cell) for one flow is different from that of another. Using similar notations for unicast traffic, we have n_i flows from input i and let $f(i,j)$ be the jth flow which reserves a bandwidth of $B_{i,j}$ and has a fan-out set of $A_{i,j}$ of $o_{i,j}$ outputs. Also, let $t_{i,j,k}(t_1, t_2]$ denote the traffic observed at output k during time $(t_1, t_2]$ from flow $f(i,j)$, where $k \in A_{i,j}$. For each flow, cells in input buffer are passed to crossbar in an FCFS manner, and an input cell is not considered as transmitted until all of its fan-outs reach their outputs. The meaning of "bandwidth" is somewhat ambiguous here. It could be interpreted as the bandwidth consumed by an incoming flow or as the sum of bandwidth consumed by all the fan outs of the flow. In fact, the latter is $o_{i,j}$ times the former for flow $f(i,j)$. We consider the bandwidth as for the incoming flows in the subsequent discussion and introduce the definition of fairness for multicast scheduling on analogy of the one in Section III for unicast.

Definition 2: For back logged flows f_{i_1,j_1} and f_{i_2,j_2} which either originate from the same input or the intersection of their fan-out set is not empty, that is, $i_1 = i_2$ or $A_{i_1,j_1} \cap A_{i_2,j_2} \neq \phi$, the scheduling is fair during $(t_1, t_2]$ if

$$\frac{t_{i_1,j_1,k_1}(t_1,t_2]}{t_{i_2,j_2,k_2}(t_1,t_2]} = \frac{B_{i_1,j_1}}{B_{i_2,j_2}} \text{ for } k_1 \in A_{i_1,j_1} \text{ and } k_2 \in A_{i_2,j_2}$$

We are particularly interested in fair bandwidth assignment under heavy traffic and have found the following assertion when all the flows are back logged during $(t_1, t_2]$. Its proof is straight forward.

Assertion 1: A scheduling is fair if and only if
for any flows $f(i, j_1)$ and $f(i, j_2)$ in input i,

$$\frac{(\Sigma_{k\in A_{i,j_1}} t_{i,j_1,k}(t_1,t_2])/o_{i,j_1}}{(\Sigma_{k\in A_{i,j_2}} t_{i,j_2,k}(t_1,t_2])/o_{i,j_2}} = \frac{B_{i,j_1}}{B_{i,j_2}} \quad (1)$$

and for any inputs i_1 and i_2,

$$\frac{\Sigma_{j=1}^{n_{i_1}}(\Sigma_{k\in A_{i_1,j}} t_{i_1,j,k}/o_{i_1,j})}{\Sigma_{j=1}^{n_{i_1}} B_{i_1,j}} = \frac{\Sigma_{j=1}^{n_{i_2}}(\Sigma_{k\in A_{i_2,j}} t_{i_2,j,k}/o_{i_2,j})}{\Sigma_{j=1}^{n_{i_2}} B_{i_2,j}} \quad (2)$$

when all the flows are back logged at the switch.

Equation (1) indicates that within each input, the bandwidth is distributed in a fair manner among its component flows. Additionally, if we consider the sum of all the flows from an input as an aggregated flow, then among the inputs, the bandwidth is again distributed proportionally to their reservations, which is stated in Equation (2).

For multicasting, it is very difficult to initiate the scheduling process from the output as done in *i*FS because that would require each flow to maintain a virtual time with respect to every output in its fan out set. The assertion suggests a two level scheduling scheme. The first level (done independently and in parallel by all the inputs) controls the entry to crossbar, which is considered as the shared resource, among the flows from within each input. A fair queueing algorithm can be employed to regulate the access order. The second level maintains a fair access to the output links. Although this scheme can be applied to both one-shot and fan-out splitting approaches, we address the latter only because one-shot is inferior in performance.

In our multicast fair scheduling(*m*FS) scheme, counters are maintained to keep the total number of cells transmitted over the crossbar from each input(e.g., counter i for input i). Each incoming cell is assigned a virtual time upon arrival. At the beginning of each scheduling, all the outputs are considered unmatched. And then

1. The fair queueing engine at each input selects a flow with the smallest virtual time and its head-of-line cell becomes a candidate.
2. Find the input i with the smallest value of $\frac{counter_i}{\Sigma_{j=1}^{n_i} B_{i,j}}$ and mark its candidate as winner.
3. Mark the outputs corresponding to all the fan-outs (or the remaining fan-outs if some of the copies has been transmitted previously) of the winner as matched. Increment $counter_i$ by 1.
4. Go to input $i+1$, $i+2$ ... till $i-1$(modulo N) (stop if all the outputs are matched) in order and check whether the candidates have any fan outs to any unmatched outputs. If so, mark the outputs as matched and remove the corresponding fan-outs. If all the fan-outs of a candidate are removed, increment the counter by 1 for its corresponding input.

The first three steps in the *m*FS algorithm correspond to the two-level approach which gives highest priority to the input with the least number of transmitted cells compared to the reservation. In the fourth step, the scheduler continues to look for matches from other inputs in a round robin way and thus this scheme is work conserving.

B. Performance evaluation for the mFS Scheme

Our proposed *m*FS Scheme is evaluated against TATRA[21], the most promising hardware multicasting scheme existing in the literature. We start off by comparing the delay and then show the effectiveness of *m*FS in supporting fair bandwidth allocation among the flows. The input arrivals are assume to be Bernoulli distributed.

We first examine a 2×8 switch and consider an example that has two flows from each input with randomly assigned fan out set of average size of four. Figure 11 shows that *m*FS exhibits higher throughput and lower delay than TATRA, especially beyond the input rate of 30%. Similar performance is shown in Figure 12 for an 8×8 switch with three sets of examples, all having average fan out of four. The *m*FS is observed to have consistently better performance than TATRA at relatively higher offered load. Intuitively, since different flows from an input in *m*FS tend to take turns to transmit their cells, the scheduler presumptively have more choices from these diverse cells brought to the head of the line. In TATRA, however, each input maintains an single FIFO queue. Once a cell is scheduled, the one brought in to the front in the next cycle is more likely from the same flow. As a result, the conflicting outputs are the same, and those cells that cannot be scheduled previously will still be unlikely to go. At the low end of input rate, *m*FS has lower delay in some cases and has larger delay in the others. We believe that the correlation of fan-out sets among the flows is an important factor to determine the relative performance.

To examine the bandwidth distributions, we inspect four flows with fan out sets: of {2,6,8}, {3,4,5,7}, {2,3,4,5} and {1,2,3}. We also assume that these flows have bandwidth reservations of 10%, 20%, 30% and 40% and label them as flow A, B, C and D in order. Flows B, C and D share a common fan out to output 3, and fan-out sets for flows A and B are disjoint. Figure 13 presents the bandwidth received by the flows versus offered load for *m*FS. We see that flows B, C and D obtain bandwidth of 22.5%, 33% and 45%, respectively. Flow A also secures the same bandwidth of 22.5% as flow B because both of them can send cells in the same slot without any conflict. As shown in Figure 14, TATRA distributes the bandwidth among flow B, C and D equally and each acquires a 33%. Flow A also get 33% since it can transmit whenever flow B does. If all the flows share

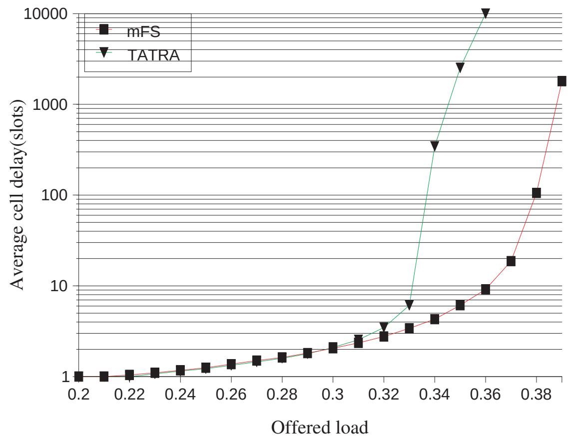

Fig. 11. Multicast: average cell delay in a 2 × 8 switch

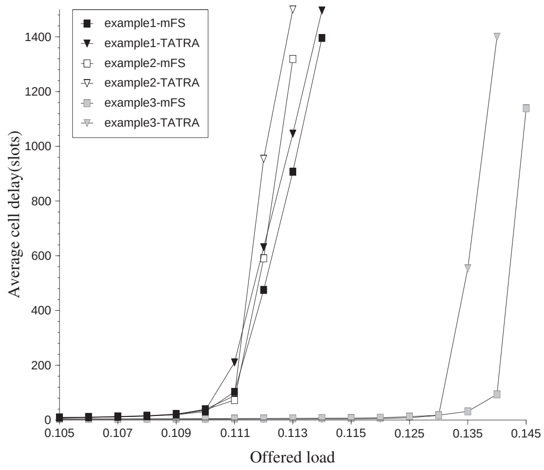

Fig. 12. Multicast: average cell delay in an 8 × 8 switch

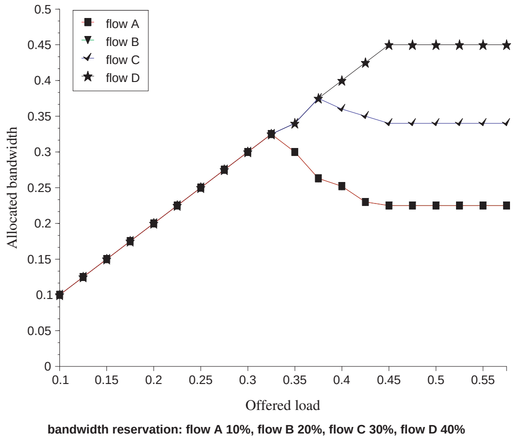

Fig. 13. Multicast:Bandwidth distribution for *m*FS

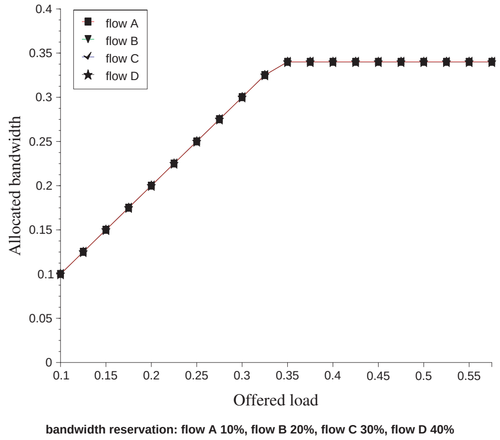

Fig. 14. Multicast:Bandwidth distribution for TATRA

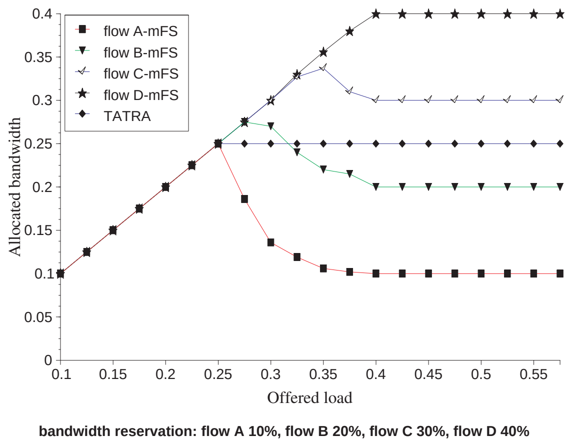

Fig. 15. Multicast:Bandwidth distribution for *m*FS and TATRA

a common output in their fan out sets, the achieved bandwidth will be proportional to the reservations in *m*FS. For example, replace the fan out sets for the above four flows with {1,2,6,8}, {1,3,4,5,7}, {1,4,5} and {1,2,3}. The achieved throughput is shown in Figure 15 under *m*FS and TATRA. As expected, the relative received bandwidth among the flows is 1:2:3:4 for the four flows with *m*FS, and that all flows get equal share of the bandwidth when TATRA is used(one curve to present all the four flows).

VI. Concluding Remarks

In this paper, we explored the scheduling schemes for input buffered switches to support fair bandwidth allocation. We first proposed an iterative fair scheduling(*i*FS) algorithm capable of scheduling cells so that each flow receives bandwidth proportional to its reservation under heavy traffic. The *i*FS includes a fair queueing engine at each output and input cells are scheduled based on a fair queueing algorithm. In this way, well-behaved flows are protected from adversely affected by ill-behaved ones and thus receive proper share of resources. We showed that the fairness support does not compromise the average cell latency when compared with other iterative scheduling schemes. We extended the idea of fair scheduling to multicasting and developed an *m*FS algorithm which supports fair bandwidth allocation for flows with fan out conflict. Simulations demonstrated that *i*FS and *m*FS are very effective in regulating traffic, particularly under heavy load. An assumption we made throughout the paper is infinite input buffer size. We believe it is worthwhile to study the effect of buffer size on the perceived bandwidth .

References

[1] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker. High speed switch scheduling for local area networks. *ACM Transctions on Computer*

Systerms, 11(4):319–352, November 1993.

[2] J. Bennett and H. Zhang. WF^2Q: worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM'96*, pages 120–128, San Francisco, CA, March 1996.

[3] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking Research and Experience*, 1(1):3–26, September 1990.

[4] J. Ding and L. N. Bhuyan. Evaluation of multi-queue buffered multistage interconnection networks under uniform and non-uniform traffic patterns. *International Journal of Systems Science*, 28(11):1115–1128, 1997.

[5] S. J. Golestani. A self-clocked fair queuing scheme for broadband applications. In *Proceedings of IEEE INFOCOM'94*, pages 636–646, Toronto, Canada, June 1994.

[6] M. W. Goudreau, S. G. Kolliopoulos, and S. B. Rao. Scheduling algorithms for input-queued switches: Randomized techniques and experimental evaluation. In *Proceedings of IEEE INFOCOM'00*, volume 3, pages 1634–1643, Tel Aviv, Israel, March 2000.

[7] P. Goyan, H. Vin, and H. Chen. Start-time fair queueing: A scheduling algorithm for integrated services. In *ACM SIGCOMM*, pages 157–168, Palo Alto, CA, August 1996.

[8] P. Gupta and N. McKeown. Design and implmentation of a fast crossbar scheduler. *IEEE Micro*, pages 20–28, Janurary/Febrary 1999.

[9] J. F. Hayes, R. Breault, and M. K. Mehmet-Ali. Performance analysis of a multicast switch. *IEEE Transactions on Communications*, 39:581–587, April 1991.

[10] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Transactions on Communications*, COM-35(12):1347–1356, December 1987.

[11] S. Li and N. Ansari. Input-queued switching with QoS guarantees. In *Proceedings of IEEE INFOCOM'99*, pages 1152–1159, New York City, NY, March 1999.

[12] S. Lu, V. Bharghavan, and R. Srikant. Fair scheduling in wireless packet networks. *IEEE/ACM Transactions on Networking*, 7(4):473–489, August 1999.

[13] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 7(2):188–201, April 1999.

[14] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *Proceedings of IEEE INFOCOM'96*, pages 296–302, San Francisco, CA, March 1996.

[15] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz. Tiny tera: a packet switch core. *IEEE Micro*, pages 26–33, January/February 1997.

[16] M. K. Mehmet-Ali and S. Yang. Performance analysis of a random packet selection policy for multicast switching. *IEEE Transactions on Communications*, 44(3):388–398, March 1996.

[17] A. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in intergrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[18] C. Partridge, P. P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, T. Ma, and J. Mcallen. A 50-Gb/s IP router. *IEEE/ACM Transactions on Networking*, 6(3):237–248, June 1998.

[19] L. L. Peterson and B. S. Davie. *Computer Networks — A System Approach*. Morgan Kaufmann, San Francisco, CA, 2000.

[20] M. Pirvu, N. Ni, and L. N. Bhuyan. Exploring the switch design space in a CC-NUMA multiprocessor environment. In *Proceedings of International Parallel and Distributed Processing Symposium*, pages 703 –710, Cancun, Mexico, May 2000.

[21] B. Prabhakar, N. McKeown, and R. Ahuja. Multicast scheduling for input-queued switches. *IEEE Journal on Selected Areas in Communications*, 15(15):855–866, June 1997.

[22] D. Serpanos and P. Antoniadis. FIRM: A class of distributed scheduling algorithm for high-speed ATM switches with multiple input queues. In *Proceedings of IEEE INFOCOM'00*, volume 2, pages 548–555, Tel Aviv, Israel, March 2000.

[23] D. C. Stephens and H. Zhang. Implementing distributed packet fair queueing in scalable switch architecture. In *Proceedings IEEE INFOCOM'98*, pages 282–290, San Francisco, CA, March 1998.

[24] D. Stiliadis and A. Varma. Providing bandwidth guarantees in an input-buffered crosbar switch. In *Proceedings of IEEE INFOCOM'95*, pages 960–968, Boston, MA, April 1995.

[25] H. Zhang. Service displines for guaranteed performance service in packet-switching networks. In *Proceedings of the IEEE*, volume 83, pages 1374–1396, October 1995.