FIRM: A Class of Distributed Scheduling Algorithms for High-speed ATM Switches with Multiple Input Queues

D.N. Serpanos and P.I. Antoniadis
Department of Computer Science
University of Crete
Knossos Avenue
P.O. Box 1470
GR-71110 Heraklion, Crete
Greece
E-mail: {serpanos,antoniad}@csd.uch.gr

Abstract— **Advanced input queuing is an attractive, promising architecture for high-speed ATM switches, because it combines the low cost of input queuing with the high performance of output queuing. The need for scalable schedulers for advanced input queuing switch architectures has led to the development of efficient distributed scheduling algorithms.**

We introduce a new distributed scheduling algorithm, FIRM, which provides improved performance characteristics over alternative distributed algorithms. FIRM achieves saturation throughput 1 with lower delay than the most efficient alternative (up to 50% at high load). Furthermore, it provides improved fairness (it approximates FCFS) and tighter service guarantee than others. FIRM provides a basis for a class of distributed scheduling algorithms, many of which provide even more improved performance characteristics.

I. Introduction

The structure of the memory subsystem is one of the main issues in the architecture and design of high-speed ATM switches. The memory subsystem is composed of the cell memory, which is organized in queues (physical or logical) of cells, and the scheduler, which is responsible to select the cells to be transmitted, when more than one incoming cells require transmission over the same physical outgoing link.

There are three typical memory organizations used in ATM switches: (i) *input queuing*; (ii) *output queuing*; (iii) *advanced input queuing (multiple input queues per input)*. In input queuing switches, each switch input link maintains a queue (FIFO), which stores incoming cells, and the scheduler decides which input queue will be served, when many cells require transmission over the same outgoing link. Unfortunately, input queuing switches suffer from a blocking phenomenon, called Head of Line blocking (HoL), which limits the throughput of the outgoing links to less than 60% under uniform traffic [4].

In switches with output queuing, each output maintains a queue (FIFO) with the incoming cells that request transmission from that output. The scheduler enqueues the incoming cells to each queue sequentially, since cells from different incoming links can arrive to the same output queue simultaneously. Output queuing switches can achieve throughput of 100%, in contrast to input queuing ones, but they need an internal fabric that transfers data N times faster than the speed of each link (for an $N \times N$ switch). So, output queuing switches provide an efficient but expensive solution, especially considering the scalability requirements on switches, i.e. as the link speed and/or the number of attached links increases.

Advanced input queuing is a third alternative architecture, which achieves high throughput, comparable to output queuing, at the cost of input queuing. The architecture uses multiple input queues (FIFO's) per input, in order to avoid the HoL phenomenon; in a typical configuration, in an $N \times N$ switch, each input maintains N queues, one for each output (see Figure 1). The architecture is attractive for high-speed switches not only because it does not require switch internal speed-up, but also because it can achieve high performance with the use of appropriate schedulers and because it can be readily used in existing switches and for the backplanes of communication systems.

Many scheduling algorithms have appeared in the literature, which have been either developed for switches with advanced input queuing [5] [9] [7] [10] [6], or apply to them [3] [2]. All these algorithms provide important and interesting results for different applications, but they differ significantly in their ease of implementation and scalability. Scalability is important to high-speed sched-

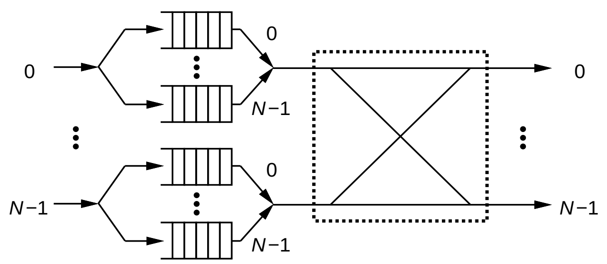

Fig. 1. Advanced Input Queuing Architecture

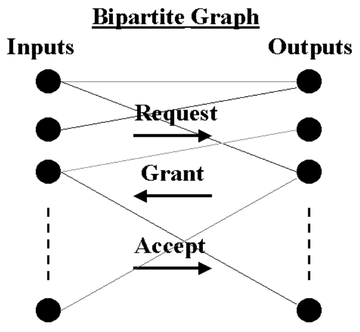

Fig. 2. Distributed Scheduling Protocol

ulers for ATM switches, because of the ATM requirements for scalability to high-speed links.

One important class of scalable schedulers which has been developed in the recent years is based on distributed scheduling algorithms based on graph matching techniques. In this approach, the scheduling problem in an $N \times N$ switch is modeled using a bipartite graph (see Figure 2), where each part contains N nodes, and one part corresponds to the input ports, while the second part to the output ports. The requests from input queues to the corresponding output ports are represented as edges, creating a bipartite graph. Using this model, distributed schedulers calculate a matching, i.e. they identify a subset of non-conflicting requests that can be served simultaneously. The calculation of the matching is performed in an iterative fashion, where each iteration augments the matching calculated thus far. There exist 3 main distributed scheduling algorithms, PIM [1], RRM [8] and iSlip [9]. The need to implement fast and efficient schedulers makes PIM an un-attractive solution, because its operation requires random choices, which are costly and "slow" in terms of performance. RRM and iSlip provide more attractive solutions, since they replace the random choices with round-robin schemes, and can be thus easily implemented efficiently. However, among these algorithms, only iSlip achieves throughput 1 under load 1; RRM suffers from blocking which limits its saturation throughput to less than 65%.

In this paper, we introduce a new distributed scheduling algorithm, FIRM, which provides improved characteristics over all existing algorithms: *(a)* it achieves saturation throughput 1 with lower delay (up to 50% over iSlip, at high load), and *(b)* provides improved fairness (closer to FCFS) and tighter service guarantee than others. Additionally, FIRM maintains more state information than alternatives and provides a basis for a class of algorithms, many of which provide even more improved performance characteristics.

The paper is organized as follows. Section II describes the existing distributed scheduling algorithms using an appropriate notation. Section III introduces FIRM, describes its differences from available algorithms, and illustrates its improved performance characteristics. Section IV finally describes a class of algorithms that can be derived from FIRM and achieve even more improved performance.

II. Distributed Scheduling Algorithms

All existing distributed scheduling algorithms are iterative and in every step (iteration) they use a handshake protocol between inputs and outputs to choose requests to serve. Every step performs the following operations, as illustrated in Figure 2:
1. inputs broadcast their requests to the outputs;
2. each output selects one request independently and issues a Grant to it;
3. each input chooses one Grant to accept, since it may receive several Grant signals.
Functional differences among the various algorithms exist only in the way that outputs choose which input to issue the Grant to, and in the way inputs choose which Grant to accept. These functional differences result in significant performance differences.

A. Notation

We introduce an appropriate notation, in order to describe the various distributed scheduling algorithms. We consider a $N \times N$ switch with N inputs, $In[0] \dots In[N-1]$, and N outputs, $Out[0] \dots Out[N-1]$. Every input $In[i]$ maintains the following state information:
1. table $R_i[0] \dots R_i[N-1]$, where $R_i[k] = 1$, if $In[i]$ has a request for $Out[k]$ (0, otherwise);
2. table $Gd_i[0] \dots Gd_i[N-1]$, where $Gd_i[k] = 1$, if $In[i]$ receives a Grant from $Out[k]$, (0, otherwise);
3. variable A_i, where $A_i = k$, if $In[i]$ accepts the Grant from $Out[k]$, (-1, if no output is accepted).
Analogously, each output $Out[k]$ maintains the following state information:
1. table $Rd_k[0] \dots Rd_k[N-1]$, where $Rd_k[i] = 1$, if $Out[k]$ receives a request from $In[i]$ (0, otherwise);

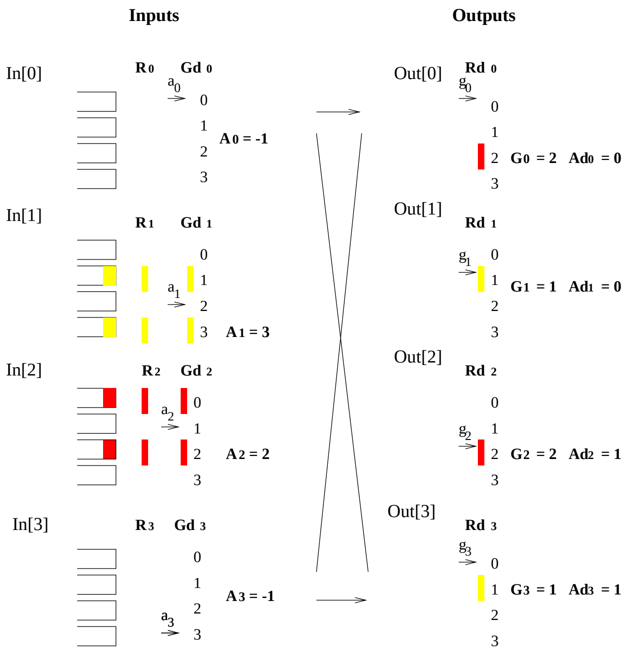

Fig. 3. Example of scheduling pointers - Cycle 0

2. variable G_k, where $G_k = i$, if $Out[k]$ sends a Grant to $In[i]$ (-1, if no input is granted);
3. variable Ad_k, where $Ad_k = 1$, if the Grant from $Out[k]$ is accepted (0, otherwise)

Each choice to issue a Grant at an output is made with a process that examines candidate inputs in some fashion, e.g. randomly, or using round-robin. In order to describe this process, we use two variables per output: g_k is a variable that shows which input will be examined first (to issue a Grant) during a step, while g'_k shows the input to be examined first in the consecutive step. Analogously, the choice of which Grant to accept at an input is made with a similar process, which is described through use of the variables a_i and a'_i.

An example of a 4×4 switch and the corresponding state information is shown in Figure 3. $In[1]$ has queued cells for $Out[1]$ and $Out[3]$; so, $R_1[1] = 1$ and $R_1[3] = 1$. Furthermore, $Out[1]$ and $Out[3]$ issue a Grant to $In[1]$, i.e. $G_1 = 1$ and $G_3 = 1$, resulting to $Gd_1[1] = Gd_1[3] = 1$. Since $a_1 = 2$, $Out[3]$ receives the Accept of $In[1]$ ($A_1 = 3$).

B. Algorithms

The various distributed scheduling algorithms differ in the way they implement the choices at the inputs and the outputs, i.e. in the calculation of the new values of variables g_k and a_i mentioned above. PIM [1] is the first proposed distributed algorithm which introduced the handshake protocol described previously. In PIM, all choices for Grant generation at the outputs and for Grant acceptance at the inputs are made randomly. So, using the introduced notation, in every step PIM executes the following operations for all $Out[k]$ and $In[i]$:

$$\begin{aligned} Rd_k[i] &= R_i[k] \\ g_k &= random(\text{among Requests}) \\ G_k &= g_k \\ Gd_i[k] &= 1, \text{if } G_k = i \\ a_i &= random(\text{among Grant's}) \\ A_k &= a_i \end{aligned}$$

The algorithm needs $\log N$ iterations to achieve good performance, i.e. to find a good matching, and is also complex to implement, due to randomness, and can be unfair [8].

RRM [8] overcomes the complexity and unfairness of PIM by issuing Grant and Accept signals with a round-robin policy (locally at every input and output). In every step, each output issues a Grant to the request that appears next in a fixed round-robin order. Analogously, each input chooses which Grant to accept. So, in every step, RRM executes the following operations for all $Out[k]$ and $In[i]$:

$$\begin{aligned} Rd_k[i] &= R_i[k] \\ g_k &= g'_k \\ a_i &= a'_i \\ G_k &= i, \text{if } (Rd_k[i] = 1 \text{ and } g_k = i) \text{ OR} \\ &\quad (Rd_k[i] = 1 \text{ and } Rd_k[j] = 0, \\ &\quad \text{where } g_k \le j < i \pmod N) \\ Gd_i[k] &= 1, \text{if } G_k = i \\ A_i &= k, \text{if } (Gd_i[k] = 1 \text{ and } a_i = k) \text{ OR} \\ &\quad (Gd_i[k] = 1 \text{ and } Gd_i[j] = 0, \\ &\quad \text{where } a_i \le j < k \pmod N) \\ g'_k &= G_k + 1 \pmod N \\ a'_i &= A_i + 1 \pmod N \end{aligned}$$

RRM performs worse than PIM, because it suffers from a "synchronization phenomenon" among round-robin pointers, where some requests experience very high delays [8]. As the algorithm does not provide any service guarantee, these delays are significantly high under some traffic loads.

iSlip [9] overcomes these limitations of RRM and resolves the pointer synchronization problem by making a change in the management of round-robin pointers: at every output, the round-robin pointer is not changed (moved), unless the corresponding Grant is accepted. In every step, iSlip executes the following operations for all

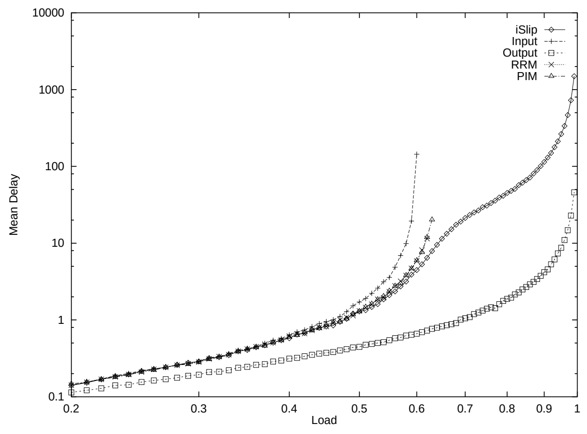

Fig. 4. Comparison of Existing Scheduling pointers

$Out[k]$ and $In[i]$:

$$
\begin{aligned}
Rd_k[i] &= R_i[k] \\
g_k &= g'_k \\
a_i &= a'_i \\
G_k &= i, \text{if } (Rd_k[i] = 1 \text{ and } g_k = i) \text{ OR} \\
&\quad (Rd_k[i] = 1 \text{ and } Rd_k[j] = 0, \\
&\quad \text{where } g_k \le j < i \ (\mathrm{mod} N)) \\
Gd_i[k] &= 1, \text{if } G_k = i \\
A_i &= k, \text{if } (Gd_i[k] = 1 \text{ and } a_i = k) \text{ OR} \\
&\quad (Gd_i[k] = 1 \text{ and } Gd_i[j] = 0, \\
&\quad \text{where } a_i \le j < k \ (\mathrm{mod} N)) \\
if(Ad_k = 1) &\quad g'_k = G_k + 1 \ (\mathrm{mod} N) \\
else &\quad g'_k = g_k \\
a'_i &= A_i + 1 \ (\mathrm{mod} N)
\end{aligned}
$$

This seemingly small change leads to significantly better performance: the scheduler achieves 100% throughput with one iteration and load 1, while PIM and RRM could not reach such performance (the performance of input queuing, output queuing, RRM and iSlip is illustrated in Figure 4 for a 16×16 switch).

iSlip also provides a service (fairness) guarantee, i.e. a bounded time in which a posted request is served. The service guarantee is at least $N^2 + (N-1)^2$ cycles, as the following theorem proves.

Theorem II.1: In the worst case scenario iSlip serves a posted request in at least $N^2 + (N-1)^2$ cycles.

Proof: We prove the result with the description of a specific scenario.

Assume that there exists only one request for $Out[k]$ (from $In[i]$), i.e. $Rd_k[i] = 1$ and $Rd_k[j] = 0$ for $0 \le j < N$ and $i \ne j$. Furthermore, assume that $g_k = ((i + 1) \bmod N)$, and $a_i = ((k+1) \bmod N)$.

iSlip will result in $G_k = i$ in this case. If $In[i]$ has a request $R_i[j]$, where $j \ne i$, which has received a Grant, i.e. $Gd_i[j] = 1$, then the accepted Grant at $In[i]$ will be the Grant received by $Out[j]$, i.e. $a_i = j$. This illustrates that the issued Grant from $Out[k]$ will not be accepted. Actually, this phenomenon, where $In[i]$ accepts the Grant from an output different from $Out[k]$ can occur $(N-1)$ steps consecutively; i.e. the maximum delay to accept the Grant from $Out[k]$ is N cycles.

Although this scenario is the worst case scenario when there are no new requests arriving at $Out[k]$, the overall delay for the Grant to be accepted at $In[i]$ can be increased, if new requests arrive at $Out[k]$. For example, if a new request arrives for $Out[k]$ from an $In[m]$ (before the Grant to $In[i]$ is accepted), where $m \ne i$, this will result in $Out[k]$ issuing a Grant to $In[m]$ in the next step. Actually, the worst case is when $Out[k]$ issues $(N-1)$ Grant signals to $In[i]$, which are not accepted, and the request from $In[m]$ arrives at that time (i.e., it "steals" the Grant). Since this can occur for all inputs $In[m]$, $m \ne i$, all Grant signals from $Out[k]$ can be declined (not accepted) for $(N-1) \times (N-1) = (N-1)^2$ cycles. This brings the switch to a state, where all inputs have requests for $Out[k]$ and $g_k = (i - (N-1)) \bmod N$, and none of them has been Granted and accepted.

From this point on, although no new request can be posted for $Out[k]$, it will be still the case that each issued Grant may need N cycles until it is accepted (the corresponding input may accept Grant signals from all other outputs first). Thus, an additional delay of N^2 cycles may be experienced before the Grant from $Out[k]$ to $In[i]$ is accepted.

Thus, overall, the total delay between posting of the request at $In[i]$ for $Out[k]$ and its service is: $(N-1)^2 + N^2$. QED. ■

III. FIRM ALGORITHM

Among the distributed scheduling algorithms, PIM uses randomness in every decision, while RRM and iSlip use a Round-Robin Policy. The only difference between RRM and iSlip is the way they calculate the value of the g'_k round-robin pointer (i.e., the round-robin pointer update after issuing the Grant signals); the update of the a_i pointer (after the Accept phase) is the same in both algorithms.

iSlip's concept for the update of the g_k round-robin pointer is the following: at an output a request is selected for a Grant using round-robin, and if the Grant is not accepted, the round-robin pointer remains unchanged (next to the last position where a Grant was accepted).

We introduce a new, alternative concept to implement the round-robin policy at the outputs: at an output a request is selected for a Grant using round-robin, and if the

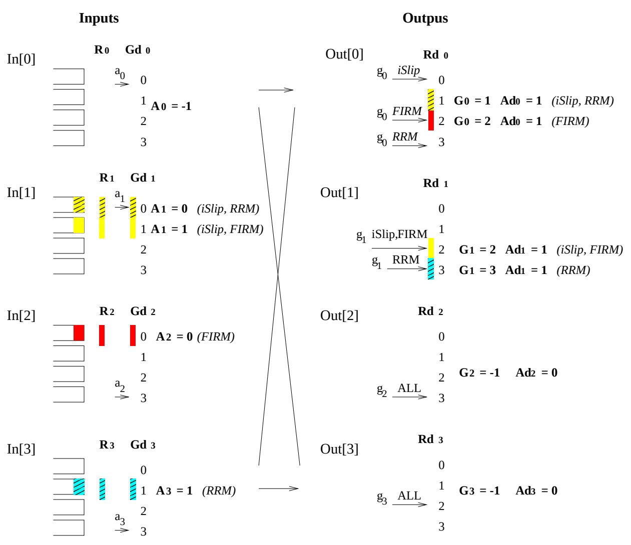

Fig. 5. Example of Scheduling pointers - Cycle 1

Grant is not accepted, the round-robin pointer is placed to this request for which the Grant has not been accepted. Effectively, this approach is more fair, because it makes certain that this request will be served next and thus, approximates FCFS better than the one in iSlip. We use this concept in a new algorithm, called FIRM (Fcfs In Round robin Matching), which differs from iSlip (and RRM) only in the way it updates the pointers g_k.

The difference in the update of the g_k pointers is illustrated in the example shown in Figure 5. The figure shows the switch of Figure 3, and its state, after the first cycle (Cycle 0), with two new requests: one at $In[1]$ for $Out[0]$, and one at $In[3]$ for $Out[1]$. The different algorithms have resulted, in general, in different values of pointer g_k, due to their round-robin policies (e.g., $Out[0]$, $Out[1]$). The different values of g_k for iSlip and FIRM may result in different Grant signals being issued in subsequent cycles: in Cycle 1, $Out[0]$ will send a Grant to $In[1]$ when using iSlip, while FIRM will result in issuing a Grant to $In[2]$. The better approximation of FIRM to FCFS, relatively to iSlip, is illustrated in the example as well: $Rd_0[2]$ has arrived earlier than $Rd_0[1]$, and FIRM serves it first, while iSlip serves $Rd_0[1]$ first.

A. *The FIRM algorithm*

In detail, FIRM operates as follows (following the handshake protocol described previously):
Request: All inputs post their requests to the outputs;
Grant: Every output sends a Grant to the request that appears next in a fixed, round-robin schedule. If the Grant is accepted in Step 3, the round-robin pointer, g_k, is incremented $(\bmod N)$ to one location beyond the Grant'ed input. *If the Grant is NOT accepted, pointer g_k is placed to the Grant'ed input.*
Accept: Every input accepts the Grant that appears next in a fixed, round-robin schedule. The pointer a_i is incremented $(\bmod N)$ to one location beyond the Accept'ed output.
Using the introduced notation, FIRM operates as follows:

$$\begin{aligned}
Rd_k[i] &= R_i[k] \\
g_k &= g'_k \\
a_i &= a'_i \\
G_k &= i, \text{if } (Rd_k[i] = 1 \text{ and } g_k = i) \text{ OR} \\
&\quad (Rd_k[i] = 1 \text{ and } Rd_k[j] = 0, \\
&\quad \text{where } g_k \le j < i \; (\bmod N)) \\
Gd_i[k] &= 1, \text{if } G_k = i \\
A_i &= k, \text{if } (Gd_i[k] = 1 \text{ and } a_i = k) \text{ OR} \\
&\quad (Gd_i[k] = 1 \text{ and } Gd_i[j] = 0, \\
&\quad \text{where } a_i \le j < k \; (\bmod N)) \\
if(Ad_k = 1) &\quad g'_k = G_k + 1 \; (\bmod N) \\
else &\quad g'_k = G_k \\
a'_i &= A_i + 1 \; (\bmod N)
\end{aligned}$$

B. *FIRM characteristics*

B.1 Fairness

FIRM is more fair than iSlip, in terms of FCFS service, i.e. it approximates FCFS closer than iSlip with the use of the round-robin pointers. This issue is illustrated with the example in Figures 3 and 5.

As Figure 3 shows, $Out[0]$ in the example receives a request for service from $In[2]$. Since $g_0 = 0$, the scheduler, reaches $Rd_0[2]$ and sends a Grant to $In[2]$, independently of the specific round-robin algorithm (i.e., RRM, iSlip and FIRM issue the same Grant). Since $In[2]$ does *not* accept this Grant ($A_2 = 2$ in the example), the scheduler needs to calculate the new value of g_0 according to its operation. iSlip does not move g_0, i.e. it remains $g_0 = 0$, while FIRM changes $g_0 = 2$; this configuration is illustrated in Figure 5.

This resulting situation allows unfairness to appear in iSlip (unfairness in terms of FCFS policy): if a new request arrives at $Out[0]$ from $In[1]$ (or $In[0]$), then the next Grant issued by iSlip will be for this newly arrived request. This is unfair, because this request has arrived later than the request of $In[2]$. FIRM overcomes this unfairness problem, because it moves g_0 to the position of the non-accepted Grant, and thus forces the scheduler to issue the next Grant to the un-successful request; this will occur, until the Grant is accepted. Thus, FIRM approximates FCFS better than iSlip.

It is important to note that, both iSlip and FIRM do not allow the pointer g_0 to overpass the position of a non-accepted Grant, and thus they offer some service guarantee (a time bound in which a posted request is guaranteed to receive service). In contrast, RRM allows g_0 to overpass the position of a non-accepted Grant and thus does not provide any service guarantee.

B.2 Service Guarantee

FIRM provides a tight service guarantee (tighter than iSlip), as the following theorem proves.

Theorem III.1: FIRM guarantees that a posted request will be served in N^2 cycles.

Proof: The worst case service scenario for FIRM is the situation where a request, e.g. from $In[i]$ to $Out[k]$, has to wait for all the other inputs to be served by the corresponding $Out[k]$, before $In[i]$ is served, i.e. when $Rd_k[j] = 1$, for $0 \leq j < N$ and $g_k = (i+1) \bmod N$. The delay between the posting of the request and its service is composed of two components:

$$Time\_to\_serve = Time\_to\_grant + Time\_to\_accept$$

The delay to issue the corresponding Grant is $(N-1) \times N$ cycles, because in the worst case scenario, all other $(N-1)$ inputs have to be served first, and each one requires at most N cycles to accept the corresponding Grant. So, $Time\_to\_grant = (N-1) \times N$, resulting to:

$$\begin{aligned} Time\_to\_serve &= Time\_to\_grant + Time\_to\_accept \\ &= (N-1) \times N + N = N^2 \end{aligned}$$

■

B.3 System Information

One important characteristic of FIRM is that it provides additional information to the system (scheduler), than RRM or iSlip. Specifically, FIRM allows inputs to know how many outputs are ready to serve their requests, i.e. how many outputs have issued a Grant to them and are waiting to be accepted. This information does not exist in RRM or iSlip, because there is no guarantee that after a Grant is issued, the output will persist on Grant'ing service to the same request until it's served.

This information enables us to develop a class of FIRM-based algorithms, which can actually provide even more improved performance, as we describe in Section IV.

B.4 Performance Evaluation

Figure 6 compares the performance of iSlip with FIRM for one iteration (note: the scales of all the plots are logarithmic in both axes). The presented results are obtained through simulation of a 16×16 switch with i.i.d. Bernoulli arrivals and uniform distribution for cell destination; this is a typical traffic pattern used to compare switch architectures[1]. As the results indicate, FIRM provides improvement over iSlip in average delay which reaches approximately 50% at loads above 0.95 for one iteration. At light loads below 0.5, FIRM provides higher delays, which are comparable to iSlip, while for medium loads, $0.5 - 0.7$, delays with FIRM are less than 15% higher. It is important that this improvement is not affected as the iterations increase. Figure 7 shows the results for one and for 4 iterations of the algorithms. Interestingly, the delays with one iteration of FIRM are much better than the ones achieved with 4 iterations of iSlip at very high loads (over 0.95). This is an important result, because the delay to execute one iteration of FIRM is comparable to the delay of one iSlip iteration, and thus in very high loads one can provide better performance with a scheduler that is $\log N$ times faster.

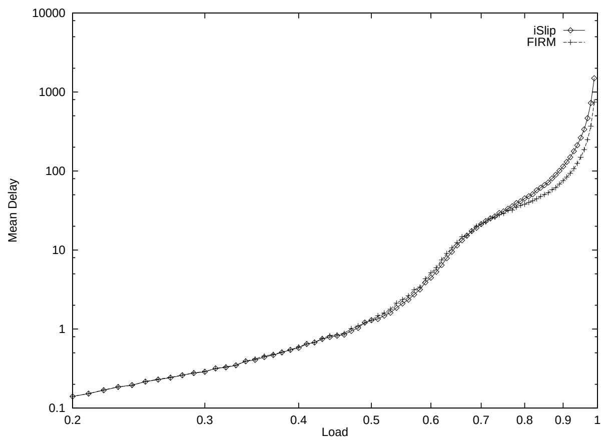

Fig. 6. FIRM vs. iSlip

IV. A Class of Simple and Efficient Distributed Schedulers

FIRM can be used as the basis of a class of distributed scheduling algorithms, because it provides additional information to the system, as mentioned in Subsection III-B.3. In the following, we describe 3 FIRM-based algorithms, which incorporate concepts introduced by existing scheduling algorithms. Specifically, we describe algorithms that incorporate the concepts of *phases* and *time reservation*.

[1]The measurements were made achieving 95% confidence intervals with half width 0.05.

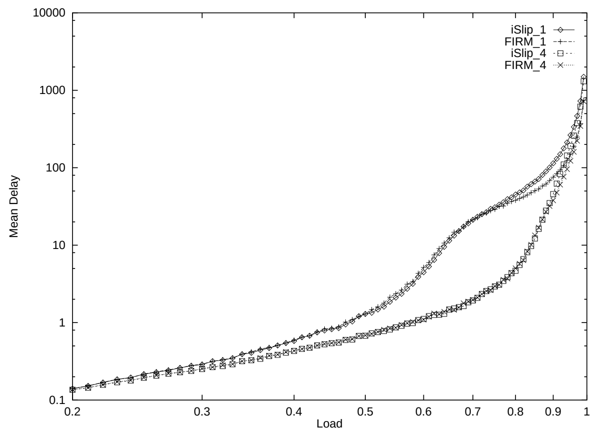

Fig. 7. FIRM and iSlip with Iterations

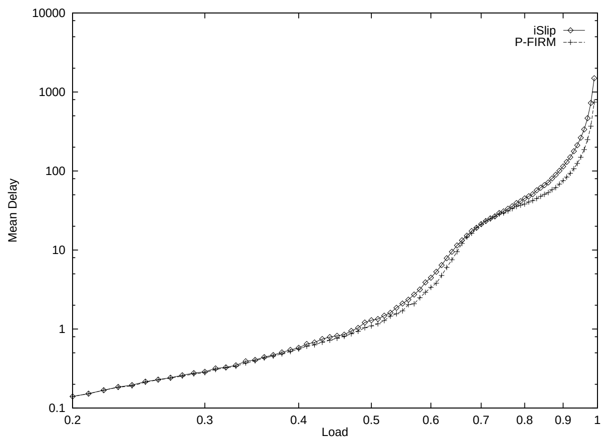

Fig. 8. P-FIRM vs. iSlip

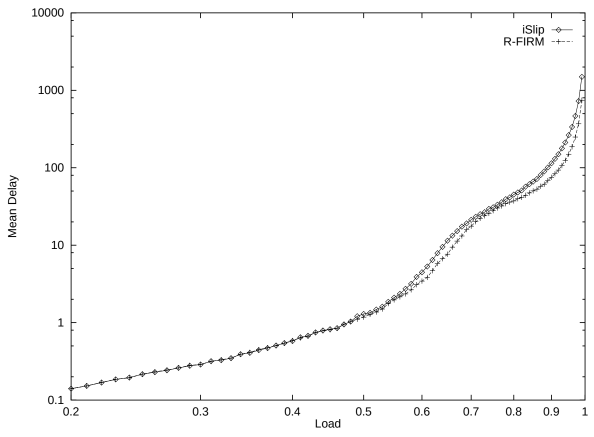

Fig. 9. FIRM with Time Reservation (R-FIRM)

A. P-FIRM: FIRM with Phases

The concept of *phases* has been used in existing scheduling algorithms [3] [2]. The idea of using phases is simple: during a time interval, i.e. a phase, all requests that have been posted before the beginning of the phase are scheduled for service. In the meantime, all new arriving requests are collected. After the end of the phase, the collected requests are served during a new phase, and so on.

FIRM can be extended to include phases. The resulting algorithm, P-FIRM, defines phases on a per input basis, i.e. it does not post newly arriving requests, until all Grant's that have been received by the input port are accepted. Using our notation, P-FIRM does not update the table $R_i[]$ in every cycle, but only when the table $Gd_i[]$ becomes empty ($Gd_i[j] = 0$ for $0 \leq j < N$). P-FIRM can be considered as an effort to accept Grant signals at the inputs with a service discipline that approximates FCFS closer: Grant signals that have been issued are served first (in a phase), and then new requests are considered for a new phase of Grant's. Effectively, FCFS service holds between requests in different phases.

As Figure 8 illustrates P-FIRM provides better delays than iSlip for all loads. Specifically, for high loads, above 73%, P-FIRM achieves the same improvement as FIRM (up to 50%), while for medium loads it provides improvement over iSlip that reaches 28%. This is a very important result, because the complexity of P-FIRM is the same as that of FIRM: the only difference between them is a condition (the status of issued Grant signals), which is checked in the beginning of each cycle.

B. R-FIRM: FIRM with Time Reservation

Considering that FIRM provides at the inputs the information of how many Grant signals have been issued for them, one can introduce time reservation mechanisms in the algorithm, similarly to existing algorithms [7] [10]. For example, one can calculate the sequence in which Grant signals will be accepted at an input, given the used round-robin policy. Given this information, inputs can inform outputs about when their Grant's will be accepted, thus allowing outputs to schedule service more efficiently among all requests, whether old and newly arrived.

R-FIRM is a FIRM variation which we have developed introducing the described time reservation scheme. In order to include time reservation, we have extended the basic handshake protocol in two main ways:

1. the Accept signal includes a time parameter (the minimum time interval until when the Grant will be accepted);
2. outputs reserve cycles to serve inputs for which they receive Accept signals (based on the given time parameter);
3. a second type of Grant is introduced, the "Soft-Grant", which is issued by outputs that have made a cycle

reservation. The goal of Soft-Grant signals is to enable the system to implement transfers additional to the ones that are realized due to time reservations. For this purpose, Soft-Grant signals do not affect the position of g_k. Thus, they do not affect the basic FIRM operations and the service guarantee offered by the algorithm.

R-FIRM, provides better performance than iSlip at all loads above 0.4, as Figure 9 illustrates, while at lower loads it provides comparable delays. In comparison to P-FIRM, R-FIRM provides the same improvement over iSlip for loads above 0.75. At medium loads ($0.5-0.75$), both algorithms provide improved performance over iSlip, but each one is better than the other in a different range; e.g., R-FIRM is better than P-FIRM at load range $0.62 - 0.75$ by a percentage reaching 20%.

C. *PR-FIRM: FIRM with Phases and Time Reservation*

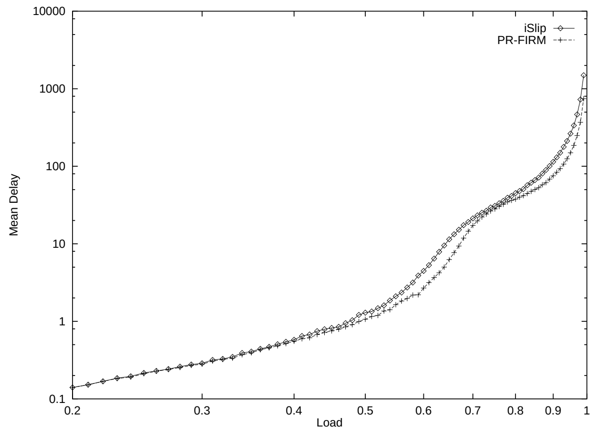

Fig. 10. FIRM with Phases and Reservations (PR-FIRM)

The concepts of phases and time reservation can be combined in an additional algorithm, PR-FIRM, which provides even better performance than the previous ones. Actually, PR-FIRM provides better performance than all previous algorithms at all loads. Figure 10 depicts this improvement, relatively to iSlip, showing a significant improvement, reaching $45 - 50\%$, in high as well as in medium loads.

Given the different performance characteristics of FIRM and its variations, we summarize their effect on average cell delay in Figure 11, where we plot the normalized delay improvement over iSlip for all loads.

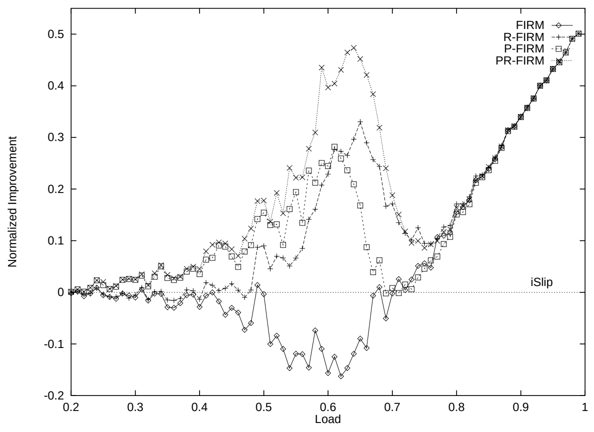

Fig. 11. Normalized Delay Improvement of FIRM Algorithms

V. Conclusions

We introduced a novel distributed scheduling algorithm, FIRM, which provides improved performance characteristics over alternative distributed algorithms. FIRM achieves saturation throughput 1 similarly to iSlip, but achieves lower average cell delay, reaching an improvement of 50% at high load. Furthermore, it provides improved fairness, it approximates FCFS, and a tighter service guarantee.

In addition to its good performance characteristics, FIRM provides a basis for a class of distributed scheduling algorithms, many of which provide even more improved performance characteristics, as we have shown.

References

[1] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High Speed Switch Scheduling for Local Area Networks. *ACM Transactions on Computer Systems*, pages 319–352, November 1993.

[2] I. Gopal, D. Coppersmith, and C.K. Wong. Minimizing Packet Waiting Time in a Multibeam Satellite System. *IEEE Transactions on Communications*, 30:305–316, 1982.

[3] T. Inukai. An Efficient SS/TDMA Time Slot Assignment Algorithm. *IEEE Transactions on Communications*, 27:1449–1455, 1979.

[4] M.J. Karol, M.G. Hluchyj, and S.P. Morgan. Input Versus Output Queueing on a Space-Division Packet Switch. *IEEE Transactions on Communications*, 35:1347–1356, 1987.

[5] R.O. LaMaire and D. N. Serpanos. Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues. *IEEE/ACM Transactions on Networking*, 2(5):471–482, October 1994.

[6] M.G. Ajmone Marsan, A. Bianco, and E. Leonardi. RPA: A Simple, Efficient, and Flexible Policy for Input Buffered atm Switches. *IEEE Communications Letters*, 1(3):83–86, May 1997.

[7] H. Matsunaga and H. Uematsu. A 1.5 Gb/s 8x8 Cross-Connect Switch Using a Time Reservation Algorithm. *IEEE Journal on Selected Areas on Communications*, 9:1308–1317, 1991.

[8] N. McKeown. *Scheduling Cells in an Input-Queued Switch*. PhD thesis, University of California at Berkeley, May 1995.

[9] N. McKeown. WHITE PAPER: A Fast Switch Backplane for a Gigabit Switched Router. *Business Communications Review*, 27(12), December 1997.

[10] H. Obara. Optimum Architecture for Input Queuing ATM Switches. *IEE Electronics Letters*, pages 555–557, March 1991.