

# A Fast Noniterative Scheduler for Input-Queued Switches with Unbuffered Crossbars\*

Kevin F. Chen, Edwin H.-M. Sha, S. Q. Zheng  
Department of Computer Science  
University of Texas at Dallas  
Richardson, TX 75083, USA  
{fxc015200, edsha, sizheng}@utdallas.edu

## Abstract

*Most high-end switches use an input-queued or a combined input- and output-queued architecture. The switch fabrics of these architectures commonly use an iterative scheduling system such as iSLIP. Iterative schedulers are not very scalable and can be slow. We propose and study a new fabric scheduling scheme that is fast and scalable. This scheduler finds maximum matching in a single iteration. It provides full throughput and incurs very low delay. It is fair and of low complexity. It greatly outperforms traditional iterative schedulers. It also renders arbitration egress memory unnecessary. Its only drawback is that it requires the support of several reads to an input memory at the same time. But simulations show this drawback is immaterial as read multiplicity is very low in various traffic conditions.*

## 1. Introduction

There has recently been renewed interest to build new switch fabric architectures as line rates go from 10 Gbps to 1 Tbps and beyond. Existing architectures are not very scalable. As memory technology evolves, switching techniques that would otherwise be considered unworkable may now be implemented. New switch fabrics can and should now be fast and highly scalable. In this paper, we propose and analyze such a novel fabric architecture.

By queuing structure, there are input-queued (IQ) switch, output-queued (OQ) switch, and combined input- and output-queued (CIOQ) switch. An OQ switch buffers cells at the output ports. OQ switches guarantee 100% throughput since the outputs never idle as long as there are packets to send. OQ switches are hard to implement. An  $N \times N$  OQ switch must operate  $N$  times faster than the line rate. Memory technology cannot meet that kind of high-

speed requirement. Therefore, IQ and CIOQ switches have gained wide-spread attention and adoption.

The most common architecture is the CIOQ switch in which buffering occurs both at the input and at the output. Output queues are for traffic scheduling which provides fine-tuned service support. As line rates increase, egress memory adds significant cost and latency in the system's datapath. Egress memory can be removed when fabric and traffic schedulings are all done at the input ports.

Both IQ and CIOQ switches use virtual output queuing by which each input maintains a separate queue for cells destined for each output or of a flow of a certain service requirement. Such a queue is called a *virtual output queue* (VOQ). Virtual output queuing removes head-of-line (HOL) blocking that can severely limit the throughput when only a FIFO queue is used for all the packets at each input.

It is customary to use a crossbar to interconnect the input and output ports due to its simplicity and nonblocking property. Crossbar access by the input cells has to be arbitrated by the fabric scheduler. Traffic scheduling manipulates the cells further to meet rate and delay requirements of various services. Fabric and traffic schedulers can be considered as separate identities. They must work in coordination to maximize datapath utilization.

A crossbar can either have memory or have no memory at its crosspoints. Fabrics using a memoryless crossbar find a bipartite matching between the input ports and output ports in each time slot. The matching has to be one input port to one output port. It is not an easy task and usually takes several iterations of an iterative algorithm. Typical iterative algorithms are like iSLIP. Our work is for a fabric using an unbuffered crossbar. Our scheduling algorithm is not iterative. Matching is done only once each time and its efficiency is much higher.

We show that our scheduling scheme, called SRA, is simple, fast, and scalable. We analyze SRA's characteristics. We also describe our simulation results that demon-

\*Work supported in part by TI University Program, NSF EIA-0103709, Texas ARP 009741-0028-2001, NSF CCF-0309461, and Microsoft, USA.

strate that SRA is far more efficient than the popular matching algorithms. We hope our architecture provides a viable alternative for designing next-generation switch fabrics.

## 2. System Model

Figure 1 shows the switch fabric architecture of an IQ switch. The switch consists of  $N$  ingress ports, an  $N \times N$  memoryless crossbar interconnect, and  $N$  egress ports. Each input port has  $N$  buffers to queue cells as needed. Each queue is a VOQ holding cells destined to different outputs or cells of different flows. Per-flow VOQs facilitate QoS services. If per-flow VOQs were present, there could be more than  $N$  VOQs in an input port.

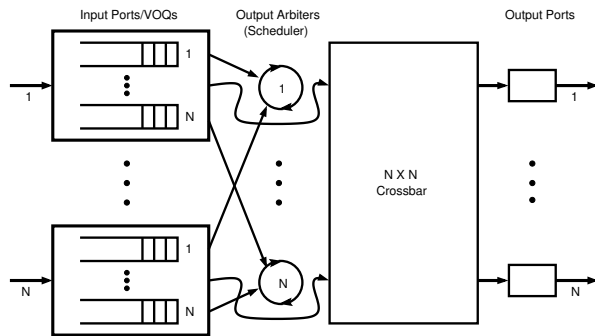


Figure 1. The architecture of the IQ switch.

### 2.1. Switch Fabric

The switch fabric excludes other functionalities that may reside in the port cards such as IP lookup, segmentation of cells in input cards, and demultiplexing and reassembly of cells in output cards. The switch fabric operates in a timing reference of its own. If the frequency of the fabric's timing is  $s$  times faster than the frequency of the link feed, we say that the *speedup* of the fabric is  $s$ . We consider the cells as of equal size. Cells are easier to synchronize and hence simplify scheduling.

Traffic feeds up to one cell into the fabric at each input port every time slot of the line timing. If traffic exits the switch one cell at each output port every line timing slot, we say that the switch achieves full throughput (100%). Consequently, the switch is *work-conserving* since the switch would keep processing as long as there are cells to route. A switch fabric of full throughput also implies it is *nonblocking*.

For each output port, there is an arbiter that keeps track of the input ports having packets destined to it and their order of arrival. Those arbiters can be placed near the input ports to keep track of VOQ status easily. At each time slot,

the arbiter grants a send to the input whose cell arrived the earliest. There is no memory existing for arbitration at an output port. Neither is there *backpressure* applied from the output ports. Backpressure is needed by many CIOQ architectures. Backpressure involves sending signals toward input ports when certain congestion thresholds are crossed in the queues at the output ports. Backpressure exerts flow control. No backpressure usage is a distinct feature of our architecture.

A crossbar is nonblocking and has an unified algorithm to switch cells at its crosspoints as needed. Fabric scheduling is to arbitrate how the cells access the crossbar in an orderly fashion so as to maximize the crossbar utilization in a time slot.

### 2.2. Fabric Scheduling

A number of algorithms have been proposed for scheduling an IQ switch to obtain high throughput. Iterative algorithms find a bipartite matching between the inputs and outputs. The scheduler matches an input with an output and finds the maximal number of those pairs in a time slot. This usually takes a few iterations. Those pairs are found globally and do not conflict one another. The scheduler uses the information on the states of the input queues and the output readiness to make the matching decision.

Typical iterative algorithms include iSLIP [1,2], PIM [3], and DSRR [4]. These algorithms find maximal matchings in  $O(\log N)$  iterations. The iSLIP algorithm works in iterations each consisting of three steps (request, grant, accept). DSRR is an enhancement to iSLIP and works similarly to iSLIP but updates pointers differently. PIM is a randomized iterative matching algorithm. We are to compare SRA to PIM, iSLIP, and DSRR in performance evaluation.

Matching algorithms like the three above appear to have some drawbacks. First, they are not scalable. The number of exchanged messages a port has to process is equal to the product of  $N$  and the number of service levels. Second, they require the fast feedback of the states of both the input and the output ports. As a result, the scheduler has to be placed in a central location. It not only impedes scalability, but also worsens the fault-tolerance of the system.

## 3. The SRA Algorithm

SRA stands for *single round-robin arbitration*. Each output port uses a round-robin arbiter to select the input port to send in a time slot. PIM, iSLIP, and DSRR all have round-robin arbiters at both the input and output ports. SRA has several significant advantages. In this section, we describe the algorithm and then analyze its properties that make it simple, fast, and scalable.

### 3.1. Description

SRA is not iterative. It selects a set of up to  $N$  cells to send to up to  $N$  outputs in a single time slot. Each cell goes to a different output. In theory, these cells can come from one,  $N$ , or any other number of inputs. That is, each input can send up to  $N$  cells in a time slot. This is where SRA differs from existing algorithms which allow each input to send at most only one cell out in each time slot. Apparently this increases efficiency since there is little reason not to let the input send more than one cell in a time slot when other inputs have no cells to send.

The SRA algorithm works as follows:

(1) At the outputs. Each output arbiter maintains a FIFO queue of status information of the inputs that have cells destined to the output. This queue can be no longer than  $N$  at any time. It always chooses the input at the head of the queue to send in the time slot. Then the output arbiter sends a grant to the input and removes the input from the head of the status queue. After the input has sent, if it still has cells queued, that input is queued again into the tail of the status queue, else the status element for that input is gone.

(2) At the inputs. Upon receiving a grant, the input checks if the corresponding VOQ is to become empty if the cell has been sent. If yes, it sends a status signal to the output arbiter indicating the VOQ is to be empty, so the output arbiter will not keep an element for this input in its FIFO queue again. Then the input port sends a cell to the crossbar with the designated output information. The input sends status information about any of its VOQs to the corresponding output only when the VOQ changes from being empty to having a cell arrived and from having cells to becoming empty.

### 3.2. Increased Scalability

In each time slot (a cell time), a fabric scheduler must perform a matching and synchronously switch on and off the cross-points of the crossbar to send up to  $N$  cells out. High line rates ever stringently require a scheduling action to be prompt. Since SRA finds a maximum matching in a single iteration, it is capable of fast scheduling actions. On the other hand, an iterative algorithm doing multiple iterations would be too slow to support high line rates.

SRA needs fewer messages to operate than an iterative matching algorithm. For each matching, an output arbiter sends only one grant message, an input can send up to  $N$  status messages out if the status of all  $N$  VOQs changes. Over the entire fabric, there are at most  $N^2$  messages exchanged between the inputs and the outputs. Unlike SRA, iSLIP needs  $N^2$  requests,  $N^2$  grant notifications, and  $N$  accepts for each iteration. For iSLIP to converge, at least  $\log N$  iterations are needed. Thus iSLIP needs a total of

$(2N^2 + N) \log N$  messages. PIM and DSRR each need about the same number of messages as iSLIP.

Another reason for SRA being better scalable is that SRA does not need constant feedback from the outputs about their readiness. CIOQ switches need this feedback for the outputs to make granting decisions. Therefore, the SRA scheduler need not be in a central location. The output arbiters can be placed near the input ports such that the input status updates can be done more easily. These output arbiters can be spatially distributed and execute in parallel (Figure 1).

Since SRA needs few messages to operate, the scheduler can expand to handle a much bigger  $N$  without adversely affecting speed. This is demonstrated by our simulations as we shall see in Section 4.

### 3.3. Performance Analysis

We first show that SRA matches the maximum number of inputs to the maximum number of outputs in each time slot. We then show that SRA facilitates 100% throughput and that it is fair.

**Theorem 1.** *SRA always finds the maximum matching.*

*Proof.* Traffic arrives in each time slot. The line rate ensures that each input gets no more than one cell. That is, in each time slot, there can be  $m$  cells and  $0 \leq m \leq N$ . If the  $m$  cells go to different outputs, the output arbiters can schedule to send all of them out in the time slot. If the  $m$  cells go to  $n$  ( $< m$ ) destinations,  $n$  ( $< m$ ) cells will be scheduled. In any case, in each time slot, if there are cells at the VOQs that are going to  $m$  ( $0 \leq m \leq N$ ) outputs, SRA ensures that  $m$  cells each from a VOQ and for each of the outputs will be scheduled. Hence, SRA always finds the maximum number of matching between VOQs and the outputs. ■

That an input port can send  $m$  ( $1 \leq m \leq N$ ) cells and the input ports altogether are allowed to send no more than  $N$  cells in a time slot is called the *free rule*. There exist analytical studies of throughput under the free rule [5–8]. These studies assume traffic arrival is i.i.d. Bernoulli with uniformly distributed destinations. Since SRA is a free-rule scheduling policy, so in theory it provides full throughput.

**Theorem 2.** *SRA is fair.*

*Proof.* We consider a loading scenario more general than uniform i.i.d. Bernoulli. Assume that the loading rate at each input is the same  $\lambda$  and is admissible. Admissible traffic does not oversubscribe any input or output port. Let  $\lambda_{ij}$  be the loading rate of traffic going from input  $i$  to output  $j$ . We have

$$\lambda_{ij} = \delta\lambda,$$

where  $\delta$  is the fraction of  $\lambda$  for traffic going from input port  $i$  to output port  $j$ . Let  $\mu_{ij}$  be the portion of the service rate  $\mu$  at output port  $j$  that serves the traffic of  $\lambda_{ij}$ . Then

$$\mu_{ij} = \delta\mu.$$

By the admissible rule, the rate of traffic arriving at port  $j$  from all the input ports combined does not exceed  $\mu$ . Also, the output arbiter works in a round-robin fashion serving each input port that has a cell in turn in each time slot. Since SRA sustains full throughput, traffic of  $\lambda_{ij}$  will receive its fair share of service. Hence we have the above equation. Also, in steady state,  $\lambda = \mu$ . Thus we obtain

$$\lambda_{ij} = \mu_{ij}.$$

Therefore, SRA is fair. ■

Note that SRA is fair per VOQ or fabric-wide. Subsequently, per-port fairness is also guaranteed.

## 4. Simulation Results

We simulated SRA against PIM, iSLIP, and DSRR in various traffic conditions. Performance metrics are cell delay and throughput vs. offered load. Offered load is the number of cells per time slot per input. The results show that SRA outperforms the other three greatly and provides high throughput and low delay.

### 4.1. Uniform Traffic

We first tested the performance of SRA when the incoming traffic is i.i.d. Bernoulli with destinations uniformly distributed. Since SRA works in one iteration, we first ran PIM, iSLIP, and DSRR for only one iteration. We then ran them for four iterations. In all these cases,  $N$  is 16. That is, the switch size is  $16 \times 16$ . We used the same traffic pattern for all four schemes.

In the case of one iteration, when the load is 20% or less, all four schemes perform the same. But when the load increases, they perform hugely different. When the load is less than 60%, PIM, iSLIP, and DSRR show about the same performance, and SRA is 6 times faster than the other three. When the load exceeds 60%, PIM becomes unstable and iSLIP performs much better than DSRR. At this time, SRA outperforms the others by many times over. Compared to PIM and DSRR, iSLIP works much better. In terms of throughput, the situation is similar.

In the case of four iterations, PIM, iSLIP, and DSRR all perform better than with one iteration. But compared to SRA, they are still off by a few times as shown in Figure 2 and Figure 3. The differentiation becomes the clearest when traffic load reaches 96% and higher. The advantage

of DSRR over iSLIP is now obvious. DSRR approaches PIM very closely overall. Both PIM and DSRR perform better than iSLIP. The delay values for PIM, iSLIP, DSRR, and SRA at load 99.5% are 217.122780, 451.399003, 265.475697, 90.784238 cell times respectively. SRA outperforms the others by 3 to 5 times at all load values. On throughput, the algorithms show similar performances.

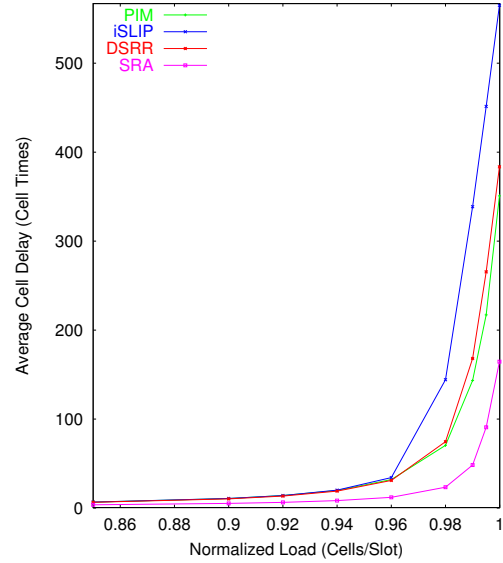


Figure 2. Cell delay under uniform traffic (four iterations).

### 4.2. Bursty Traffic

We modeled bursty traffic using interrupted Bernoulli process (IBP). IBP is the discrete version of interrupted Poisson process. IBP is similar to two-state Markov-modulated Bernoulli process, exponential on/off process, and Pareto on/off process. Switch size is again  $16 \times 16$ .

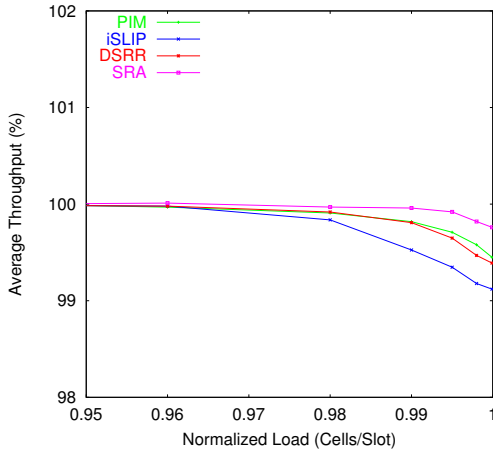
IBP has two states (on and off) and is characterized by three parameters  $\alpha$ ,  $p$ , and  $q$ . In each time slot, if the current state is on, the state remains on in the next time slot with probability  $p$ ; if the current state is off, the state remains off in the next time slot with probability  $q$ . In the on state, a cell arrives in a time slot with probability  $\alpha$ . The length of on state,  $X$ , and the length of off state,  $Y$ , have geometric distributions:

$$P\{X = x\} = (1 - p)p^{(x-1)},$$

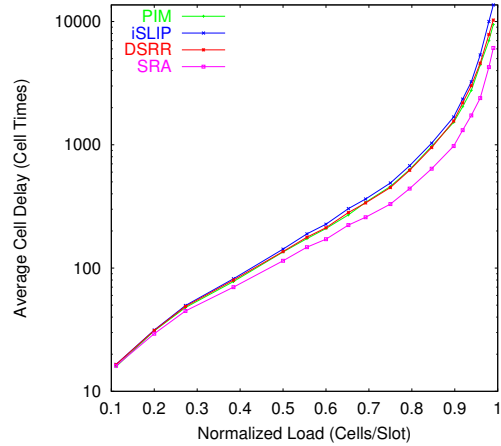
$$P\{Y = y\} = (1 - q)q^{(y-1)}.$$

The mean arrival rate or offered load,  $\rho$ , is

$$\rho = \frac{\alpha(1 - q)}{2 - p - q}.$$



**Figure 3. Throughput under uniform traffic (four iterations).**



**Figure 4. Cell delay under bursty traffic.**

Average burst length is

$$b = E[X] = \frac{1}{1-p}.$$

In each burst period, arrived cells all go to the same destination. Thus  $b$  measures how bursty the traffic is. In our simulations, we set  $b = 128$  cells and  $\alpha = 1$ . When  $\alpha$  is set,  $p$  is set. To get various loads, we just vary the value of  $q$ .

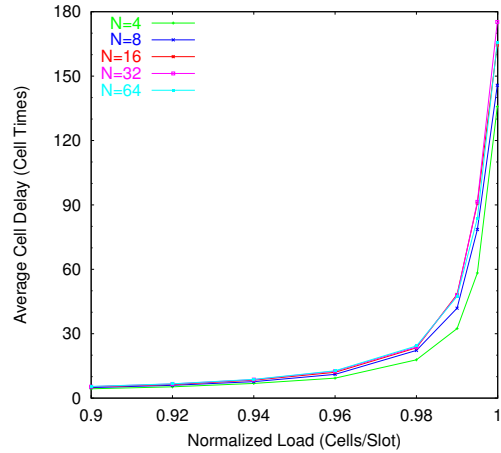
As shown in Figure 4, SRA is faster than the other three over all loads. PIM, iSLIP, and DSRR were run for four iterations. At load 95.92%, the delay values are 4453.010231, 5356.881259, 4596.592691, 2391.089810 cell times for PIM, iSLIP, DSRR, and SRA respectively. In all times, PIM, iSLIP, and DSRR are very close to each other, and SRA works 2 to 3 times faster than them. Also, SRA maintains the highest throughput under all loads. Its throughput dips when load passes 90% but still less than the other three schemes. Thus SRA is the most stable at providing high throughput.

### 4.3. Effect of Switch Size

The performance of iSLIP degrades considerably as switch size increases as shown in [2]. The switch slows down a time when  $N$  doubles. We saw little slowdown with SRA when switch size increases.

We ran a few simulations on a  $N \times N$  switch with  $N$  being 4, 8, 16, 32, and 64. Traffic is i.i.d. Bernoulli with uniformly distributed destinations. As shown in Figure 5, when  $N = 4$ , cell delay is the smallest. But when  $N$  takes higher values, cell delay remains just about the same. That

implies that the SRA scheduling scheme provides the same efficiency regardless of  $N$ . Thus SRA is scalable.



**Figure 5. SRA cell delay as a function of switch size (uniform traffic).**

## 5. The Problem of Input Blocking

SRA allows an input port to simultaneously send a cell in a time slot to each of up to  $N$  output ports. (That is, an input port can send  $k$  ( $1 \leq k \leq N$ ) cells in a time slot. We call  $k$  *cell multiplicity*.) This requires the input port memory be capable of concurrent read. Should the memory technology be unavailable, other means must exist to mitigate the delay. Contention of multiple outputs reading data at the same time from the same input memory is called *input blocking*. We discuss here how input blocking can be circumvented.

OQ scheduling requires  $N$  writes (plus  $N$  reads if the output ports are under one shared-memory) in one time slot. SRA transforms the  $N$  writes of OQ into  $N$  reads. A read operation is much easier to perform and needs minimal hardware support. Thus, SRA reduces the complexity while approaching OQ in speed. Note that actual OQ switches do exist despite the multiple writes problem.

The maximum input blocking delay occurs when  $N$  reads have to be performed at one time. For a given input traffic pattern, the probability of this occurrence is zero under SRA. We did simulations for a  $64 \times 64$  switch under various loads of uniform i.i.d. Bernoulli traffic and IBP traffic. An input port can send zero, one, or multiple cells in a time slot. When a send involves multiple cells, multiple reads to the same input memory, hence input blocking, occurs. Frequency of a cell multiplicity is calculated by dividing the number of sends of that particular cell multiplicity with the total number of sends during the simulation duration.

The simulations indicate that  $k$  tends to be far smaller than  $N$ . The chance for input blocking to happen at all is also low. Specifically, the data we obtained show that for  $N = 64$  the occurrence of  $k > 2$  is about 7% in the worst case. That  $k > 5$  virtually does not occur. Most input ports send only one cell or send none for a time slot. Some send two. Thus input blocking is slight. The above property of low  $k$  holds true indifferently of  $N$ . Our simulations show that  $k$  is nearly unchanged when  $N$  is 16, 64, and 128 under the same uniform and bursty traffic conditions.

Since SRA sometimes requires simultaneous multiple reads from an input port, additional hardware support is needed at the crossbar to handle the multiple cells arriving to the crossbar so that the cells can go to separate inputs of the crossbar. Splitting the cells to separate inputs, albeit easy, would incur some latency. A simple solution would be to make the crossbar to have  $kN$  rows with  $k$  rows of links connected to each input port. Parameter  $k$  may be set to be 3 as each input port sends 3 cells to the crossbar at most times. This way, the structural complexity of the crossbar is  $kN^2$ , which is still in the order of  $N^2$ . That would be of the same order of complexity as the typical crossbar used for traditional iterative algorithms. By the criterion of Li, Zheng, and Yang [9], the new architecture is still scalable.

With SRA, speed gain appears to outweigh speed loss due to possible input blocking. SRA gets  $N$  cells sent to the outputs by simply letting the queue head element at the output to send and sending one grant signal back to the inputs. Using an iterative matching scheme, this process would take at least 4 iterations of request, grant, and accept with numerous signals sent. SRA removes this complexity at the expense of a much smaller delay of multiple reads. In view of this and the other aforementioned facts, input blocking in this context is a benign tradeoff for simplicity and speed.

## 6. Conclusion

A switch implementing SRA is necessarily an IQ switch as it does not need egress memory for arbitration. The benefits of using SRA include high throughput and low delay. Cell delays incurred by the iterative PIM, iSLIP, and DSRR in simulations would be much higher if the time spent on iterating the algorithms were taken into account. SRA is scalable and reduces the complexity of switching. SRA could serve as an exemplar scheduler for IQ switches. Whether SRA can be useful to IQ switches with buffered crossbars, the other promising alternative to designing IQ switches, is yet to be investigated. Also, SRA is a best-effort architecture as is. Quality of service and multicast supports merit further study.

## References

- [1] N. McKeown, J. Walrand, and P. Varaiya, "Scheduling cells in an input-queued switch," *IEE Electronics Letters*, vol. 29, no. 25, pp. 2174–2175, Dec. 1993.
- [2] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.
- [3] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, Nov. 1993.
- [4] Y. Jiang and M. Hamdi, "A fully desynchronized round-robin matching scheduler for a VOQ packet switch architecture," in *Proceedings of IEEE HPSR 2001*, May 2001, pp. 407–411.
- [5] C. Koliass and L. Kleinrock, "Throughput analysis of multiple input-queueing in ATM switches," in *Proceedings of the International IFIP-IEEE Conference on Broadband Communications*, Apr. 1996, pp. 382–393.
- [6] K. L. Yeung and S. Hai, "Throughput analysis for input-buffered ATM switches with multiple FIFO queues per input port," *IEE Electronics Letters*, vol. 33, no. 19, pp. 1604–1606, Sept. 1997.
- [7] H. Kim, C. Oh, Y. Lee, and K. Kim, "Throughput analysis of the bifurcated input-queued ATM switch," *IEICE Transactions on Communications*, vol. E82-B, no. 5, pp. 768–772, May 1999.
- [8] H. Kim and K. Kim, "Performance analysis of the multiple input-queued packet switch with the restricted rule," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 478–487, June 2003.
- [9] C. Li, S. Q. Zheng, and M. Yang, "Scalable schedulers for high-performance switches," in *Proceedings of IEEE HPSR 2004*, Apr. 2004, pp. 198–202.