Cooperative Token-Ring Scheduling for Input-Queued Switches

Amir Gourgy, *Member*, *IEEE*, and Ted H. Szymanski, *Member*, *IEEE*

Abstract—We present a novel distributed scheduling paradigm for Internet routers with input-queued (IQ) switches, called cooperative token ring (CTR), that provides significant performance improvement over existing scheduling schemes with comparable complexity. Many classical schedulers for IQ switches employ round-robin arbiters, which can be viewed as noncooperative token rings, where a separate token is used to resolve contention for each shared resource (e.g., an output port), and each input port acquires a token oblivious of the state of other input ports. Although classical round-robin scheduling schemes achieve fairness and high throughput for uniform traffic, under nonuniform traffic, the performance degrades significantly. We show that by using a simple cooperative mechanism between the otherwise noncooperative arbiters, the performance can be significantly improved. The CTR scheduler dynamically adapts to nonuniform traffic patterns and achieves essentially 100 percent throughput. In addition, our proposed CTR scheduling paradigm can amortize the arbitration time over multiple time slots such that tokens are exchanged only on an as-needed basis. The proposed cooperative mechanism is conceptually simple and is supported by experimental results. To provide adequate support for rate guarantees in IQ switches, we present a weighted CTR, a simple hierarchical scheduling mechanism. Finally, we analyze the hardware complexity introduced by the cooperative mechanism and describe an optimal hardware implementation for an $N \times N$ switch with a time complexity of $\Theta(\log N)$ and a circuit size of $\Theta(N \log N)$ per port.

Index Terms—Switch scheduling, quality of service, input-queued switch, parallel prefix.

1 Introduction

Most commercial high-performance switches and routers (e.g., CISCO 1200 [1] and BBN [2]) employ input-queued (IQ) switches because output-queued switches are difficult to build in practice. Although output queuing provides optimal performance, for an $N \times N$ switch, it requires the switching fabric and memory to run up to N times faster than the line rate; unfortunately, for large or for high-speed data lines, memories with sufficient bandwidth are not available. In contrast, the fabric and the memory of an IQ switch need to run only as fast as the line rate, which makes input queuing very appealing for switches with fast line rates or with a large number of ports. Unfortunately, IQ switching can suffer from head-of-line (HOL) blocking, which limits the throughput under uniform traffic to just 58.6 percent, if each input maintains a single FIFO queue [3].

The virtual output queuing (VOQ) architecture is commonly used for avoiding HOL blocking such that each input port maintains a separate queue for each output port [4]. Fixed-length switching technology is widely accepted for achieving high switching efficiency such that variable-length IP packets are segmented into fixed-length cells at each input port and are reassembled at each output port. We assume fixed-length cell scheduling for the remainder of this paper.

In the VOQ architecture, incoming cells are queued at the input ports, and a scheduling algorithm configures the switch fabric during each time slot. The scheduler essentially computes a bipartite graph matching in each time slot. The graph corresponding to the scheduling problem has N source vertices that correspond to the N inputs of the switch, and N sink vertices that correspond to the outputs. An input and an output are connected by an edge if the corresponding VOQ is not empty; thus, there are at most N^2 edges between source and sink vertices. A matching $\mathcal{M}$ on this graph is any subset of the edges such that no two edges in $\mathcal{M}$ have a common vertex; that is, at most one cell is transferred from each input, and at most one cell is received at each output.

In an $N \times N$ switch, the scheduler must examine the contents of N^2 VOQs and determine a conflict-free matching. Although IQ scheduling can be optimally solved using a maximum-weight matching algorithm [5], it requires a runtime complexity of $\Theta(N^3)$ on a sequential model, which makes the optimum algorithm prohibitively expensive. Instead, most practical algorithms are based on simple heuristics that aim at maximizing the number of connections between inputs and outputs and attempt to achieve a *maximal match* (e.g., iFair [6], iDRR [7], and FIFOMS [8]). A maximal matching can be achieved by adding edges incrementally, without removing edges made earlier in the process. These schemes may use multiple iterations to converge on a maximal matching and require N iterations in the worst case. Although these maximal matching algorithms provide fairness, QoS support, and good throughput for uniform traffic, the performance degrades for realistic nonuniform traffic.

- *A. Gourgy is with Research in Motion Inc., Waterloo, ON N2L 3W8, Canada. E-mail: agourgy@ieee.org.*
- *T.H. Szymanski is with the Department of Electrical and Computer Engineering, McMaster University, Hamilton, ON L8S 4K1, Canada. E-mail: teds@mail.ece.mcmaster.ca.*

Manuscript received 18 Mar. 2006; revised 12 July 2007; accepted 29 Oct. 2007; published online 17 Sept. 2008.
Recommended for acceptance by A. Mei.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0108-0306.
Digital Object Identifier no. 10.1109/TC.2008.178.

In summary, it is a challenge to find a scheduling scheme for IQ switches that meets the following requirements:

1. It provides high throughput, essentially 100 percent, for both uniform and nonuniform traffic.
2. It provides rate guarantees for QoS traffic and proportional bandwidth sharing.
3. It is readily implemented in hardware. Most practical schedulers are iterative with a hardware time complexity of $\Theta(\log N)$ per iteration, where N is the size of the switch; usually, $\log(N)$ iterations are used in practice [9].

A precise statement of our objective is the following: to find an iterative scheduling algorithm for IQ switches that achieves all the previous three requirements; i.e., it achieves essentially 100 percent throughput for uniform and nonuniform traffic, it provides rate guarantees, and it has a complexity comparable to existing iterative schedulers.

We present a solution, called the cooperative token ring (CTR), that meets all these requirements. We emphasize that almost all practical scheduling policies in the literature can provide high throughput under uniform traffic; however, under nonuniform traffic, the throughput usually degrades significantly. In contrast, the proposed CTR solution dynamically adapts to nonuniform traffic patterns and achieves essentially 100 percent throughput.

This paper is organized as follows: Section 2 provides an overview of related work on scheduling for IQ switches. Section 3 provides an overview of the proposed CTR scheduling paradigm. In Section 4, we present the algorithmic details of the proposed CTR scheduler. We present a parallel implementation of CTR in Section 5. The performance of CTR, for best effort traffic, is evaluated by simulation in Section 6. In Section 7, we examine the fairness of the proposed CTR scheduler. We propose a two-level hierarchical scheduler, the weighted CTR (WCTR) scheduler, which supports rate guarantees and proportional bandwidth sharing in Section 8. In Section 9, we provide an optimal hardware implementation for the proposed cooperative mechanism. Finally, Section 10 provides our conclusions.

2 Related Work

In an IQ switch, there are essentially two shared resources: the switch fabric and the outgoing link. Arriving packets are queued at the input ports of the switch, and they must first contend for access to the switch fabric, before contending for the outgoing link. A simple paradigm that is commonly employed in implementing maximal matching is using an input arbiter at each input port to resolve input contention and an output arbiter at each output port to resolve output contention such that a maximal match is achieved by iteratively pairing inputs to outputs. Specifically, two schemes can be classified under this iterative paradigm: the two-phase and three-phase schemes with different implementation trade-offs. Initially, all the inputs and outputs are not matched. A three-phase algorithm comprises the following phases:

1. *Request phase.* Each unmatched input arbiter sends a request to every output arbiter for which it has a queued cell.
2. *Grant phase.* Each unmatched output arbiter resolves output port contention by selecting only one of the input requests and sending back a grant signal to the selected input port.
3. *Accept phase.* Each input arbiter resolves input port contention by accepting only one of the received grants. It sends back an accept signal to the corresponding output arbiter to confirm the match.

The previous phases are repeated until either a maximal matching is found or a fixed number of iterations are performed.

In a two-phase algorithm [10], each input arbiter sends at most one request; subsequently, it receives at most one grant signal, and the accept phase is not needed; for example, the dual round-robin (DRR) algorithm [11] performs the following two phases:

1. *Request phase.* Each unmatched input arbiter sends a request to an output arbiter corresponding to the first nonempty VOQ in a fixed round-robin order, starting from the current pointer position. The pointer remains at that nonempty VOQ if the request is not granted in the second phase.
2. *Grant phase.* If an output arbiter receives one or more requests, it grants the one that appears next in a fixed round-robin order starting from the current pointer position. The output arbiter notifies each input port whether or not its request was granted. The pointer of the output arbiter is incremented to one location beyond the granted input port. If there are no requests, the pointer's position does not change.

On one hand, a two-phase algorithm requires less communication and is simpler to implement than a three-phase algorithm; on the other hand, a three-phase algorithm tends to converge to a maximal matching faster than a two-phase algorithm. Consequently, with the same number of iterations, a three-phase algorithm usually provides better performance. For simplicity, we refer to all scheduling schemes based on either the two-phase or three-phase matching paradigm as ΠRGA.

Generally, most scheduling schemes based on the ΠRGA scheduling paradigm could be viewed as employing traditional token rings, where nodes in the ring correspond to input arbiters that perform the request and accept phases and where tokens correspond to output ports. A token that is acquired by a node corresponds to an input port being matched to an output port. The token rotation corresponds to the case where a match is not confirmed between an input port and an output port, and other potential matches are then considered.

Anderson et al. [4] proposed Parallel Iterative Matching (PIM), a three-phase algorithm that uses *random* selection at each input and output arbiter. Although finding a maximal matching using PIM may, in the worst case, take N iterations, it was shown that [4] under *uniform* independent identically distributed (i.i.d.) traffic, the algorithm converges to a maximal matching in $\Theta(\log N)$ iterations; however, for a single iteration, the throughput is limited to approximately 63 percent for uniform i.i.d. traffic.

McKeown proposed the iSLIP algorithm [9], which uses rotating priority round-robin arbiters. Under uniform traffic, the pointers used in the input and output arbiters (i.e., for the grant and accept phases) tend to point to different ports (*desynchronize*) such that each arbiter tends to point to a different port compared to other arbiters and the largest number of input and output ports tend to be matched. Consequently, under uniform Bernoulli i.i.d. traffic, iSLIP arbiters adapt to a time-division multiplexing scheme, providing a perfect matching and 100 percent throughput [9]. However, under nonuniform traffic, the pointers are not necessarily desynchronized and the performance potentially degrades—Chang et al. [12] showed, using a pathological traffic pattern for a 3 × 3 switch, how iSLIP can get trapped in "bad modes" such that the throughput is limited to 66.67 percent.

In [13], a scheduling algorithm, FIRM, is proposed to better approximate FCFS than iSLIP with a slight modification to the grant phase of iSLIP. Although FIRM reduces the absolute maximum waiting time for any request from $(N-1)^2 + N^2$ for iSLIP to N^2 at no additional implementation cost, its sustained throughput is essentially identical to iSLIP under different traffic models; that is, the throughput degrades under nonuniform traffic models.

McKeown [14] proposed a class of schedulers based on weighted ΠRGA such that rather than using a 1-bit request, a weight is assigned to each request. The grant phase selects the request with the highest weight rather than using the round-robin order. Different weight functions have been proposed for this class of schedulers; for example, iLQF uses the queue length as the weight, whereas iOCF assigns an arrival time stamp to each cell, and the grant phase selects the cell with the oldest age. The iLPF scheduling algorithm [15] uses a weight for every VOQ that equals that length of all cells at its input port (i.e., summation of the occupancies of all the VOQs at this input) plus the summation of all cells in the switch destined to the same output (i.e., summation of all occupancies of all the VOQs at other inputs destined to the same output). RPA [16] is also a weighted heuristic scheduler where cells are selected based on their urgency.

Although these weighted schedulers generally perform better than nonweighted algorithms, they require complex comparators rather than the simple round-robin arbiters used in the nonweighted schedulers. To maintain the occupancies of each VOQ, addition/subtraction computations need to be performed for every input port during every time slot. In addition, some of these schemes like iLQF could suffer from a starvation problem [14], and algorithms like iOCF require computing a time stamp for every single cell in the switch, which is expensive to implement in hardware at high speeds (see [14, pp. 84-87] for a discussion on the complexity of these weighted schedulers).

Duan et al. [17] proposed a matrix unit cell scheduler (MUCS) that is based on a weighted heuristic. In MUCS, a matrix is computed where rows corresponds to inputs and columns corresponds to outputs. The algorithm is iterative, and during every iteration, an entry weight is computed for each element in the matrix that is a weighted summation of its row (input port) and column (cells destined to the same output port). Subsequently, the element with the heaviest weight is selected as the winner, ties are broken randomly, and its corresponding row (input) is matched to its column (output) and is removed from subsequent iterations of the algorithm. The heaviest element in the matrix corresponds to the cell with the least contending candidates (input and output contention). By choosing the heaviest element first, each iteration of MUCS renders the remaining elements with the maximum number of scheduling opportunities. On one hand, this algorithm tends to perform well under several traffic models; on the other hand, the computation performed per iteration is relatively complex as a weighted addition of all inputs and outputs needs to be performed and a comparator tree is required to select the element with the heaviest weight. MUCS uses a mixed digital-analog core for computing the element with the heaviest weight, which alleviates some of this complexity but makes it difficult to support QoS by setting each element's initial value to a single binary bit value. Also, note that unlike ΠRGA algorithms, MUCS is inherently sequential in nature in the sense that every iteration matches exactly one input to one output, and consequently, it requires N iterations for computing a maximal matching. In contrast, ΠRGA usually matches several input-output pairs during every iteration and on the average requires $\Theta(\log N)$ iterations to converge to a maximal matching. The sequential time complexity of MUCS is $\Theta(N^3)$.

Randomized scheduling algorithms have been proposed for IQ switches [18] in an attempt to simplify the scheduling problem and provide fairness. The basic idea of randomized scheduling is to select the *best* matching from a set of random matches. The best matching is defined to be the matching with the maximum weight, where the weight of each edge typically corresponds to the length of the corresponding VOQ, and the weight of the entire matching is the summation of all the edge weights in the matching. The algorithms proposed in [19] start with an initial matching from a previous time slot. During each iteration, a maximal matching is randomly computed and *merged* with the initial matching to form a new matching with a weight that is at least equal to any of the two merged matchings. The algorithms described in [19] use the length of each VOQ and an arrival matrix to compute a new random matching. The merging of two matchings requires sequentially comparing the edge weights from both matchings and selecting the one with the larger weight. The computational complexity per iteration of these randomized algorithms is significantly higher compared to other nonweighted schemes. Each iteration requires the computation of a full maximal matching based on assigned weights and a merge with an existing matching. Furthermore, given the randomized nature of the algorithm, it is not clear how it can be extended to provide deterministic rate guarantees. At best, these schemes could be extended to provide probabilistic guarantees. Finally, the hardware complexity

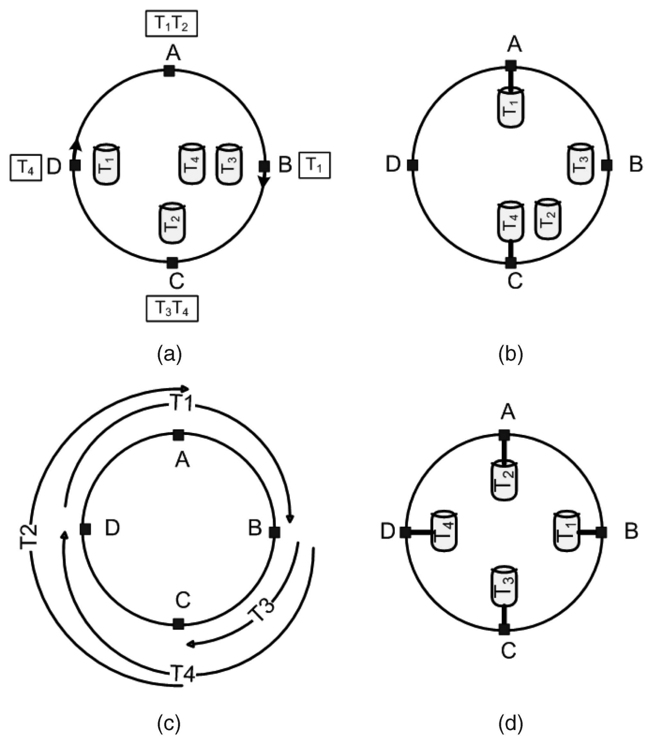

Fig. 1. Scheduling using classical token ring and CTR. (a) Initial state. (b) Classical token ring. (c) TRVs. (d) CTR.

of these randomized algorithms has never been thoroughly addressed in the literature, and the feasibility of their hardware implementation at high speeds is unknown.

Load-balanced Birkhoff-von Neumann [12] switches address the problem of scheduling nonuniform traffic using a two-stage scheduler: a load balancing stage followed by a second scheduling stage that essentially operates on uniform traffic. The main drawback of this architecture is that packets can be missequenced. Furthermore, providing a scalable solution that simultaneously provides QoS support and solves the packet missequencing problem is a major difficulty in the load-balanced router architecture.

To cope with degrading performance under nonuniform traffic, without increasing the scheduler's complexity, Li et al. [20] proposed coupling the ΠRGA paradigm with *exhaustive matching* (EM). In EM, after an input port is matched to an output port, the VOQ is served continuously until it becomes empty. Specifically, it was shown [20] that exhaustive iSLIP (EiSLIP) produces the best results compared to several proposed exhaustive scheduling algorithms and performs better than nonexhaustive matching algorithms, under some nonuniform traffic patterns.

3 Overview of Cooperative Token-Ring Scheduling

In this section, we informally describe the CTR scheduling policy. The goal is to provide an intuitive understanding of the concept rather than to list the algorithmic details, which are given in Section 4.

Consider the system shown in Fig. 1a. There are a set of four nodes (users) that are alphabetically labeled A, B, C, and D. There are four resources, which are represented by the four tokens T_1, T_2, T_3, and T_4. These tokens rotate clockwise in the ring and could be acquired by any of the nodes subject to the constraint that each node acquires at most one token simultaneously. Each node maintains a separate queue of requests for each token that represents backlogged work for that resource. We assume that time is slotted such that token arbitration is performed during each time slot, wherein each node may acquire an unclaimed token or release an acquired token. At the end of token arbitration, each node may acquire at most one token, in which case it consumes one request from the corresponding queue of backlogged requests. By definition, a node that acquires a token is considered matched; otherwise, it is unmatched.

Consider the configuration shown in Fig. 1a where each of the four nodes has backlogged queues for resources, which are represented by the rectangle boxes outside the ring: node A has requests for tokens T_1 and T_2, node B has requests only for token T_1, node C has requests for tokens T_3 and T_4, and node D has requests only for token T_4. The initial token position(s) are as shown in Fig. 1a: T_1 resides at node D, T_3 and T_4 reside at node B, and T_2 resides at node C. Note that a token can reside at a node without necessarily being matched to that node. Furthermore, multiple unacquired tokens can reside at one node. For the purposes of this discussion, we assume that in one iteration of the CTR algorithm, an unacquired and unrequested token can complete one revolution of the ring. If it is unclaimed by any node, it resides where it started.

In classical scheduling schemes employing round-robin arbiters, each node makes an independent token-selection decision oblivious of the state of other nodes; for example, given the initial state shown in Fig. 1a, each node could acquire the first available token to result in the matching state shown in Fig. 1b, where node A acquires token T_1, and node C acquires token T_4. The resource utilization in this example using classical round-robin arbiters is 50 percent. Note that nodes B and D do not require tokens T_2 and T_3.

CTR is an iterative scheme such that each iteration comprises two phases. In the first phase, a *token request vector* (TRV) for each token is computed, as shown in Fig. 1c. In the second phase, tokens propagate along the ring and may be acquired. The main idea of CTR is to create a TRV along the ring for each token, from its current position to the last unmatched downstream node in the ring that requires that token. The vectors for the initial token configuration in Fig. 1a are pictorially shown in Fig. 1c; for example, the vector for token T_1 starts at node D and ends at node B, which is the last unmatched downstream node that requires token T_1. The value of the TRV, for each token, at each node is a Boolean variable that indicates whether this token is requested by any unmatched downstream node along the ring. Subsequently, tokens propagate along the ring such that each node uses the TRVs to decide whether to acquire, swap, or release a token to improve the overall resource utilization. Specifically, the token propagation/acquisition in the second phase of the CTR algorithm is performed at each node to achieve the following two goals:

$\mathcal{G}1$. *Improve a node's resource utilization.* If a node is unmatched, then it acquires the first available token

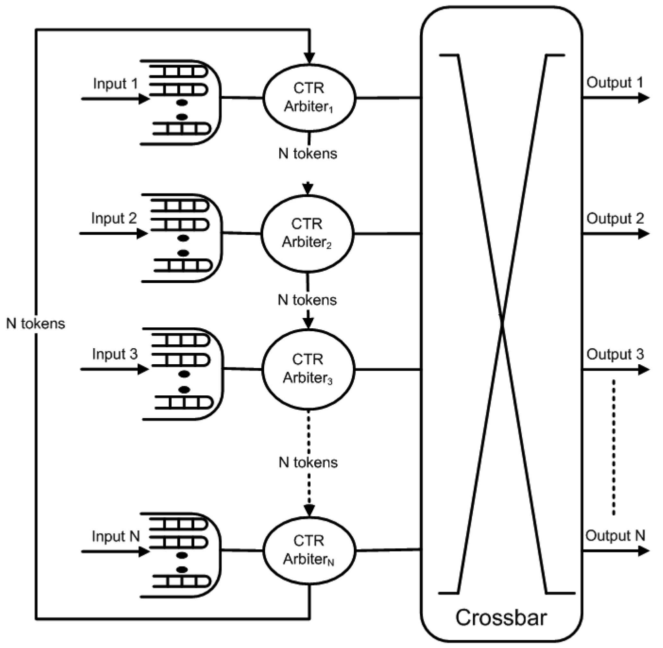

Fig. 2. Architecture of CTR switch.

that it needs, regardless of whether this token is requested by other downstream nodes along the ring.

$\mathcal{G}2$. *Improve the overall resource utilization of the ring by swapping an acquired token for another unrequested token, thereby breaking an existing match to create a new match.* This token swapping is performed using the TRVs that have been previously computed.

After computing the vectors in Fig. 1c, tokens propagate in the ring. When node A receives token T_1, it acquires it according to [$\mathcal{G}1$]. When tokens T_3 and T_4 arrive at node C, node C acquires T_3 according to [$\mathcal{G}2$]. Node C observes that T_4 is requested by a downstream node, whereas T_3 is not. Therefore, to maximize resource utilization, node C acquires token T_3.

Token T_4 subsequently propagates along the ring and is acquired by node D according to [$\mathcal{G}1$]. When token T_2 arrives at node A, node A swaps token T_1 for the token T_2 according to [$\mathcal{G}2$]. Token T_1 is thus released and propagates to node B, where it gets acquired according to [$\mathcal{G}1$]. The final state is shown in Fig. 1d, where each node is matched, and the resource utilization is 100 percent.

In essence, the main difference between traditional scheduling algorithms employing round-robin arbiters and the CTR algorithm is that each arbiter in the traditional scheme only considers its own resource utilization, whereas in the CTR scheme, each node additionally *cooperates* with other nodes in the ring to improve the overall resource utilization.

4 Description of Cooperative Token-Ring Scheduler

The basic architecture of a CTR switch is shown in Fig. 2. There are N tokens in the ring that correspond to the N output ports of the switch such that each CTR arbiter is allowed to acquire at most one token. When input i is matched to output j in a time slot, then it is allowed to transmit a cell to output j during that time slot.

CTR is an iterative algorithm such that each iteration comprises two phases:

1. *Computing the TRVs.* In this phase, a TRV is computed for *each* token. As shown in Fig. 1c, the TRV for each token is distributed among all nodes. At each node, the value of the TRV for a specific token indicates whether this token is requested by an unmatched downstream node(s): a 1 bit indicates that the token is requested, and a 0 bit indicates otherwise. Computing the TRV is described in Section 4.1, and its hardware complexity is examined in Section 9.1.
2. *Token propagation/acquisition.* In this phase, tokens propagate along the ring, and each CTR arbiter may acquire a token based on its VOQ status and TRV.

Our design strategy is to have a communication structure that is feasible to implement in hardware and that could be used to iteratively improve the throughput such that a trade-off could be made between the number of iterations performed and the achieved throughput. We would like the communication structure to reflect the dynamic nature of the traffic conditions such that the scheduler is able to dynamically adapt to time-varying traffic, which manifests itself in the status of VOQs, such that little or no exchange of tokens is performed between the different arbiters when the status of VOQs does not change and more communication is performed when the status of the VOQ changes and arbiters become unmatched. The communication structure in the CTR is the TRVs computed among all nodes. Unequivocally, the TRVs can be viewed as forming guided paths for the tokens to reach their intended destinations that lead to an overall performance improvement.

The computation of the TRVs and token propagation/acquisition phases are described in detail in Sections 4.1 and 4.2, respectively.

4.1 Computing the Token Request Vector

The TRV computations are distributed over all N nodes in the ring. Let $TRV_{i,j}$ be the value of the TRV at node i for token j. $TRV_{i,j}$ is set to true if there is an unmatched downstream node along the ring from node i that requests token j. Here, we make precise some terminology used for the remainder of this paper. We adopt a matrix representation to represent the switch's state. We use the following standard notations:

- $+$ denotes the standard Boolean OR operation.
- $\times$ denotes standard Boolean AND operation.
- 1 denotes true and 0 denotes false.
- $\overline{A}$ denotes the Boolean not operator applied to the Boolean parameter A.

Definition 1: The VOQ state matrix VOQ. *$VOQ_{i,j}$ is set to one if the VOQ at input i for output j is nonempty and is set to zero otherwise.*

Definition 2: The token position matrix TP. *Each row i represents the tokens that are residing at node i at the* *beginning of an iteration. Specifically, $TP_{i,j} = 1$ if the token for output port j is located at node i at the beginning of an iteration and is set to zero otherwise. Note that multiple tokens can reside at the same node, and also, when token j is located at node i, it does not necessarily imply that node i is matched to output j.*

Definition 3: The request matrix R. *$R_{i,j}$ is the request by node i for token j. $R_{i,j}$ is set to zero, if node i is matched and is set to $VOQ_{i,j}$ otherwise. The request matrix is used for computing the TRVs.*

Consider the computation of the TRV for token j in a ring with eight nodes labeled $0, \ldots, 7$, where token j is at node 4 initially. The token can only perform one revolution per iteration. From node 0's perspective, the set of downstream nodes that may request token j are nodes 1, 2, and 3. Therefore, the calculation of $TRV_{0,j}$ for node 0 needs to consider only nodes $1, \ldots, 3$; $TRV_{0,j} = 1$ if any of nodes $1, \ldots, 3$ requests token j. Furthermore, assume that token k is located at node 7 initially. From node 0's perspective, the set of downstream nodes that may request token k are nodes $1, \ldots, 6$. Therefore, the calculation of $TRV_{0,k}$ for node 0 needs to consider only nodes $1, \ldots, 6$; $TRV_{0,k} = 1$ if any of nodes $1, \ldots, 6$ requests token k.

Assume a token ring with N nodes and N tokens, labeled from 1 to N. Let $|k| = (k \ mod \ (N+1))$. Identify a downstream node, m, from node i, with respect to token j by the predicate $\prod_{t=1}^{t=m} \overline{TP}_{|i+t|,j}$; that is, the predicate evaluates to true (m is a downstream node) if token j is not at any intermediate node between i and m circularwise. The same algorithm is used to compute the TRV for each token in the ring. To simplify the notation, we focus on computing one element in the TRV at node i for token j and drop the second subscript. The value of the TRV at node i for token j is given by

$$TRV_{i,j} = R_{|i+1|} + \sum_{j=i+2}^{j=i+N-1} R_{|j|} \prod_{k=i+1}^{k=j-1} \overline{TP}_{|k|}. \quad (1)$$

The predicate $\prod_{t=1}^{t=m} \overline{TP}_{|i+t|,j}$ in (1) is interpreted as masking out the requests for upstream nodes such that only the downstream requests are ORed together.

Various implementation schemes could be used to compute the **TRV** based on (1). One possible implementation is to exploit the linear ring structure and propagate information along the ring. The time complexity using this technique is $\Theta(N)$. In Section 9, we describe how a binary tree structure could be used to evaluate (1) in $\Theta(\log N)$ time. We emphasize that computing TRV at each node requires simple Boolean operations that is readily implementable in hardware.

4.2 Phase 2: Token Propagation/Acquisition Phase

The CTR arbiter at each node performs token acquisition using the computed $TRVs$ for all tokens at this node. By definition, $TRV_{i,j} = 0$ indicates that token j is critical at node i; i.e., it is not requested by any other downstream nodes from node i; otherwise, it is noncritical. A precondition for an arbiter to acquire a token is that the corresponding VOQ is nonempty. Each CTR arbiter acquires a token according to the following rules:

$\mathcal{R}1$. An input that is not matched acquires the first available token regardless of whether this token is critical or not—this ensures that the matching converges.

$\mathcal{R}2$. The acquisition of a critical token takes precedence over the acquisition of a noncritical token.

$\mathcal{R}3$. An acquired token is swapped for a critical token, when possible.

$\mathcal{R}4$. An input arbiter that has backlogged cells for its acquired token can hold its acquired token for more than one time slot.

The prioritization according to [$\mathcal{R}2$] is done to provide other unmatched downstream inputs the chance to acquire noncritical tokens and improve the overall throughput.

[$\mathcal{R}3$] allows two cases for token swapping: swapping an acquired noncritical token for a critical token as in [$\mathcal{R}2$] and swapping an acquired critical token for *another* critical token. The swapping of critical tokens allows the breaking of cyclic dependencies; for example, consider a token ring with three nodes A, B, and C such that C is not matched and requests a token that is acquired by B. B would relinquish its acquired token only if it acquires the token that is acquired by node A. According to Definition 3, node B cannot send a request for the token acquired by node A because B is already matched; however, node A would swap its acquired token according to [$\mathcal{R}3$], which in turn would be acquired by B, thereby releasing the token required by node C to achieve 100 percent utilization. A detailed example that shows how swapping critical tokens could break a cyclic dependency and other detailed examples that show a step-by-step operation of the CTR scheduling policy are provided in [21].

[$\mathcal{R}4$] is based on the observation that the state of the VOQs changes slightly between time slots. Therefore, rather than starting each matching from scratch at the beginning of each time slot, [$\mathcal{R}4$] attempts to improve over the matching computed from the previous time slot.

There are various mechanisms for implementing a CTR scheduler with implementation trade-offs. We emphasize that our description so far has been only a logical description: any hardware implementation that logically implements the CTR scheduler could be used; for example, in Section 5, we describe how CTR could be physically implemented using a ΠRGA paradigm, which is typically employed in high-speed IQ switch implementations [22].

5 Parallel Implementation of Cooperative Token-Ring

In this section, we present a parallel implementation of the CTR scheduler that is tailored toward high-speed implementation with a hardware time complexity of $\Theta(\log N)$ per iteration based on the ΠRGA paradigm.

In the first phase of each iteration, a TRV is computed for each token as described in Section 4.1. In the second phase of each iteration, the token propagation/acquisition is performed using the ΠRGA paradigm as described next.

As shown in Fig. 3, a round-robin arbiter is used at each output port, and a CTR arbiter is used at each input port. The round-robin arbiter at each output implements the logical token ring for the corresponding output, whereas

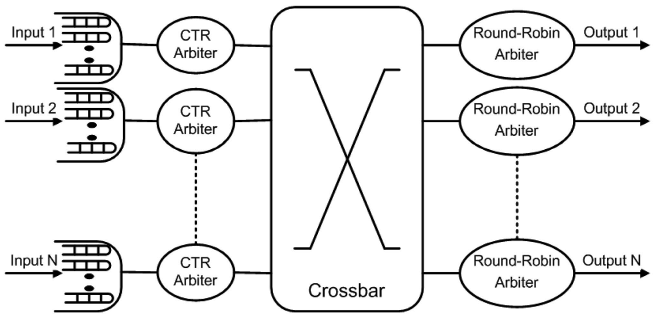

Fig. 3. Parallel implementation of CTR.

the CTR arbiter implements the token selection described in Section 4.2; the CTR arbiter implements the request and accept phases, and the round-robin arbiter implements the grant phase of the ΠRGA paradigm. Each CTR arbiter and round-robin arbiter uses a rotating round-robin priority encoder as described next. Specifically, each iteration of the CTR algorithm comprises the following steps:

1. Compute the TRV for each token as described in Section 4.1.
2. *Request step.* Each *unmatched* CTR arbiter sends a request to every output arbiter for which it has a queued cell, whereas each *matched* CTR arbiter sends a request to every *critical* and unmatched output[1] for which it has a queued cell.
3. *Grant step.* If an unmatched output arbiter receives any requests, it chooses the one that appears next in fixed round-robin priority order. The output notifies each input whether or not its request was granted through a grant signal. The pointer to the new highest priority element of the round-robin arbiter is incremented (modulo N) to one location beyond the granted input.
4. *Accept step.* Each CTR arbiter selects one of the grant signals and sends an accept signal to the corresponding output arbiter. The selection of a grant follows [$\mathcal{R}1$]-[$\mathcal{R}3$] as described in Section 4.2, which are reiterated here for completeness. There are two cases:

   a. *Unmatched input.* Select a grant for a critical output if possible; otherwise, select a grant for a noncritical output and send the accept signal starting with the highest priority element. The corresponding input and output are considered matched. The round-robin pointer is incremented (modulo N) to one location beyond the accepted output.
   b. *Matched input.* By definition, the received grants are from critical outputs (as matched inputs only sent requests to critical outputs in the request step). The input arbiter accepts one of these grants in a round-robin fashion starting with the highest priority element—the CTR arbiter uses a rotating round-robin priority. The matched input resets (breaks) its acquisition to its previously matched output and sends an accept signal (and becomes matched) to the critical output whose grant it is accepting. The round-robin pointer is incremented (modulo N) to one location beyond the accepted output.

1. Technically, it is irrelevant whether an input sends a request to a matched output because by definition a matched output ignores the requests it receives, but it helps simplify our presentation.

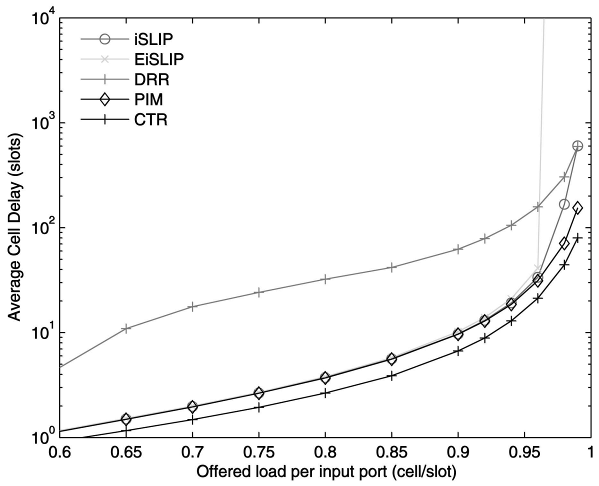

Fig. 4. Performance of CTR, iSLIP, DRR, PIM, and EiSLIP for uniform traffic.

6 Simulation Results for Best Effort Traffic

In this section, we evaluate the performance of CTR, iSLIP, EiSLIP, DRR, and PIM for a 16 × 16 switch with four iterations. All simulations were performed with 99 percent confidence and 1 percent accuracy; that is, the simulations were run until the relative width of the confidence interval equals 1 percent with probability $\geq$ 99 percent. We evaluate the performance for both Bernoulli traffic distributions and bursty traffic models.

6.1 Bernoulli Traffic Distribution

We use various traffic models recommended by the switching fabric benchmarking group [23]. The following arrival patterns are used with Bernoulli traffic distribution (note that ρ denotes the normalized load such that all inputs are equally loaded, and N is the switch size):

1. *Uniform.* $\lambda_{i,j} = \rho/N \; \forall i, j$.
2. *Diagonal.* $\lambda_{i,j} = 2\rho/3$, $\lambda_{i,|i+1|} = \rho/3 \; \forall i$, and $\lambda_{i,j} = 0$ for all other i and j. This is very skewed loading and is more difficult to schedule than uniform loading.
3. *Log diagonal.* $\lambda_{i,j} = 2\lambda_{i,|j+1|}$, and $\sum_i \lambda_{i,j} = \rho$; for example, the distribution of the load at input i across outputs is $\lambda_{i,j} = \frac{2^{N-j}\rho}{2^N-1}$. This type of load is more balanced than diagonal loading but more skewed than uniform loading.

Figs. 4, 5, and 6 show the average delay under uniform, log-diagonal, and diagonal traffic, respectively. The CTR scheduler provides the best performance, i.e., the lowest average cell delay and the highest throughput, under the three traffic patterns. The improvement achieved by the proposed CTR scheduling policy manifests itself clearly as the arrival pattern becomes more skewed. Under uniform arrivals in Fig. 4, all schemes can sustain up to essentially 100 percent traffic load,

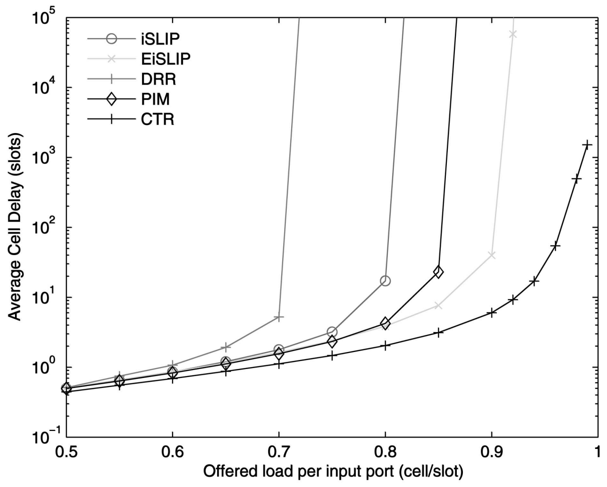

Fig. 5. Performance of CTR, iSLIP, and EiSLIP for log-diagonal traffic.

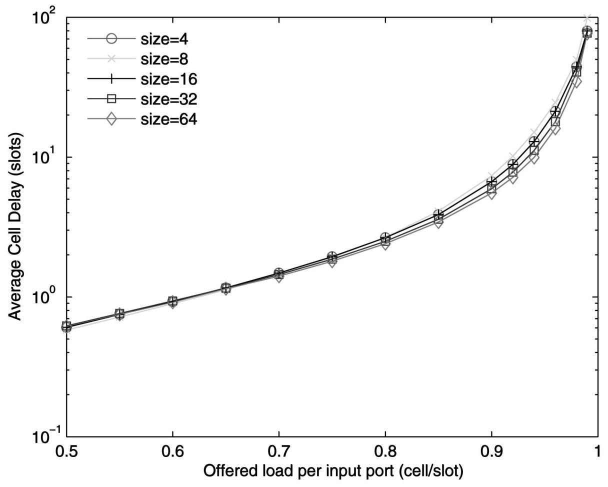

Fig. 7. The performance of CTR as a function of switch size for uniform i.i.d. Bernoulli arrivals.

and CTR provides the lowest delay. In Fig. 5, the arrival pattern becomes more skewed under log-diagonal traffic. The iSLIP, PIM, and EiSLIP schedulers saturate at approximately 80 percent, 85 percent, and 90 percent loads, respectively. In contrast, the CTR scheduler provides essentially 100 percent throughput for traffic loads exceeding 90 percent. Fig. 6 illustrates the diagonal arrival pattern, which is the most skewed pattern. The iSLIP, PIM, and EiSLIP schedulers saturate at 80-85 percent loads. Only the CTR scheduler is able to provide essentially 100 percent throughput for traffic loads larger than 85 percent.

6.2 Simulation as a Function of the Switch Size

Fig. 7 shows the average latency imposed by a CTR scheduler as a function of offered load for switches with 4, 8, 16, and 32 ports for Bernoulli uniform traffic with $\log(N)$ iterations. The performance is almost identical for the various switch sizes.

6.3 Bursty Traffic Distribution

Real Internet traffic is bursty [24], and bursty traffic models were considered in the simulations. Specifically, an on/off Markov modulated arrival process with a geometrically distributed burst size was used.

Each input port is connected to a burst source that generates traffic cells using a two-state Markov process that alternates between busy and idle states. The process remains in the busy and idle states for a geometrically distributed number of cell times. When the server is in the busy state, a cell arrives at the beginning of every time slot, and all cells in the burst have the same destinations. This traffic model is described in detail in [23]. An average burst size of 16 was used.

As shown in Fig. 8, the same trend occurs, and CTR provides the best performance.

6.4 Effects of Increasing the Number of Iterations

One of the main arguments for CTR is that its performance iteratively improves as more iterations are performed. The effect of increasing the number iterations was evaluated by simulation for CTR, iSLIP, EiSLIP, DRR, and PIM. A 16×16 switch was used, and the numbers of iterations executed were 1, 2, 4, 8, and 16.

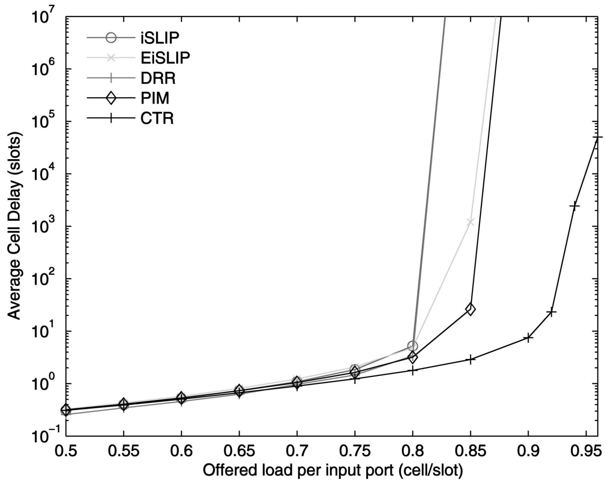

Fig. 6. Performance of CTR, iSLIP, DRR, PIM, and EiSLIP for diagonal traffic.

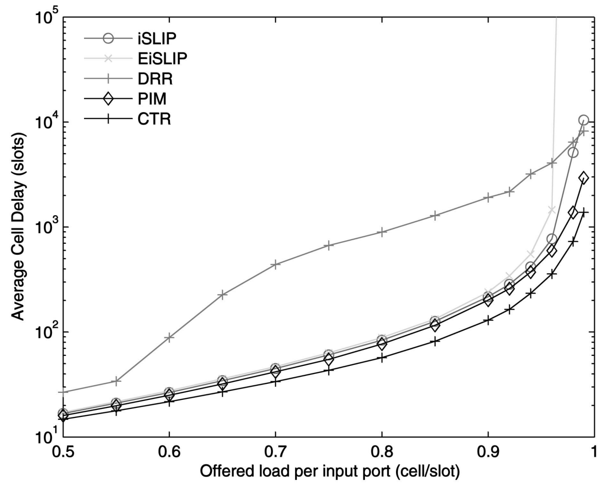

Fig. 8. Average delay under two-state Markov-modulated arrivals with an average burst size of 16.

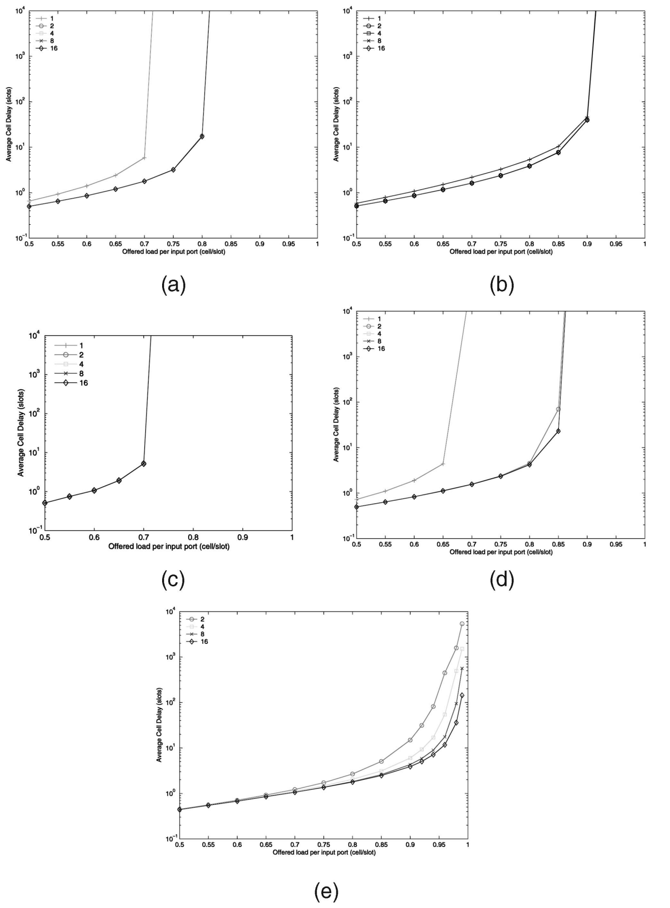

Fig. 9. Effects of increasing the number of iterations under log-diagonal Bernoulli i.i.d. traffic. (a) iSLIP. (b) EiSLIP. (c) DRR. (d) PIM. (e) CTR.

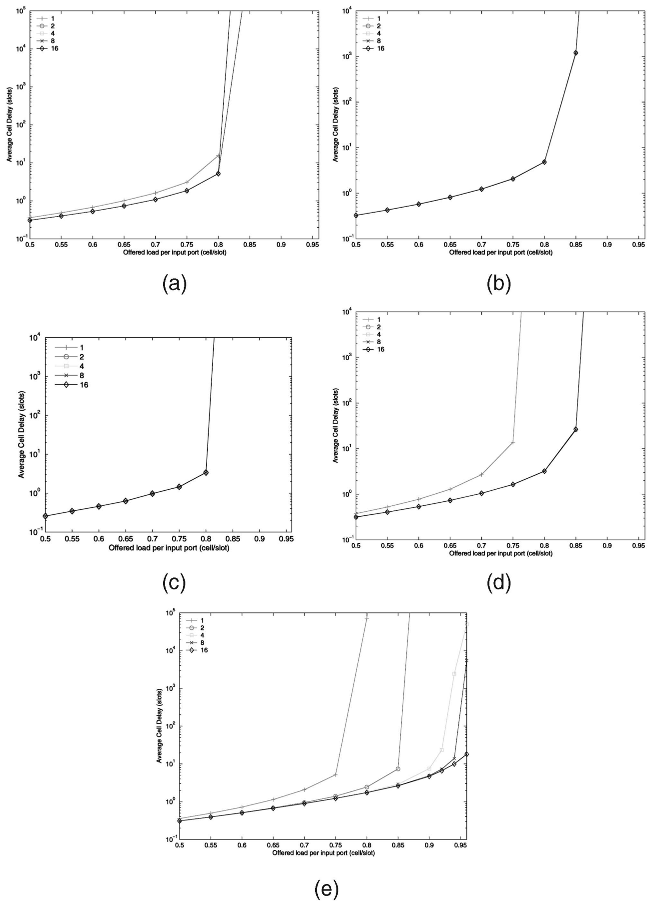

Fig. 10. Effects of increasing the number of iterations under diagonal Bernoulli i.i.d. traffic. (a) iSLIP. (b) EiSLIP. (c) DRR. (d) PIM. (e) CTR.

6.4.1 Log-Diagonal Bernoulli Traffic

As shown in Fig. 9, CTR outperforms all other schemes for the same number of iterations. CTR is the only scheme that sustains essentially 100 percent traffic load even with a single iteration. Observe that regardless of executing more iterations, none of the schemes other than CTR can sustain a traffic load larger than 90 percent.

6.4.2 Diagonal Bernoulli Traffic

As shown in Fig. 10, none of the schemes other than CTR sustains a traffic load higher than 85 percent, whereas CTR provides essentially 100 percent throughput with four iterations. In addition, the performance of CTR incrementally improves as more iterations are executed, as expected.

6.5 ON/OFF Markov-Modulated Traffic

As shown in Fig. 11, CTR outperforms all other schemes even with a single iteration under the bursty traffic model with a geometrically distributed burst size of 16.

7 Fairness of Cooperative Token-Ring Scheduling

Given that the CTR scheduling policy potentially violates the strict ordering of round-robin arbitration to achieve high throughput, it is expected that it could suffer from a fairness problem for an adversarial traffic pattern. We describe the fairness problem in the current design and then describe several augmenting schemes to the proposed CTR scheduling policy that address fairness.

We have chosen to address the fairness issue separately for several reasons. First, there are various possible solutions with trade-offs in their implementation complexities that depend on the desired granularity of fairness that we believe should be left to the designer. Second, the solutions to the fairness problem are complementary to the concept of CTR scheduling, and separating the issue helps simplify our presentation. Third and more importantly, we view the decoupling between achieving high throughput and providing fairness as one of our key contributions. It is our view that the tight coupling of a rigid fairness scheme in many scheduling policies, which almost dictates the next schedule, forces the scheduler to not adapt to the traffic dynamics, thus causing an overall performance degradation. On one hand, in a strict round-robin scheduler, the *uniform* selection of the next matching element tends to dovetail with *uniform* i.i.d. traffic, and the scheduler can provide essentially 100 percent throughput [25]. However, for nonuniform traffic, strict round-robin scheduling causes performance degradation. On the other hand, exhaustive scheduling policies (e.g., EiSLIP [26]) potentially provide better performance than strict round-robin schedulers for bursty traffic, but the scheduler could still get locked into "bad modes" because each arbiter makes its decision

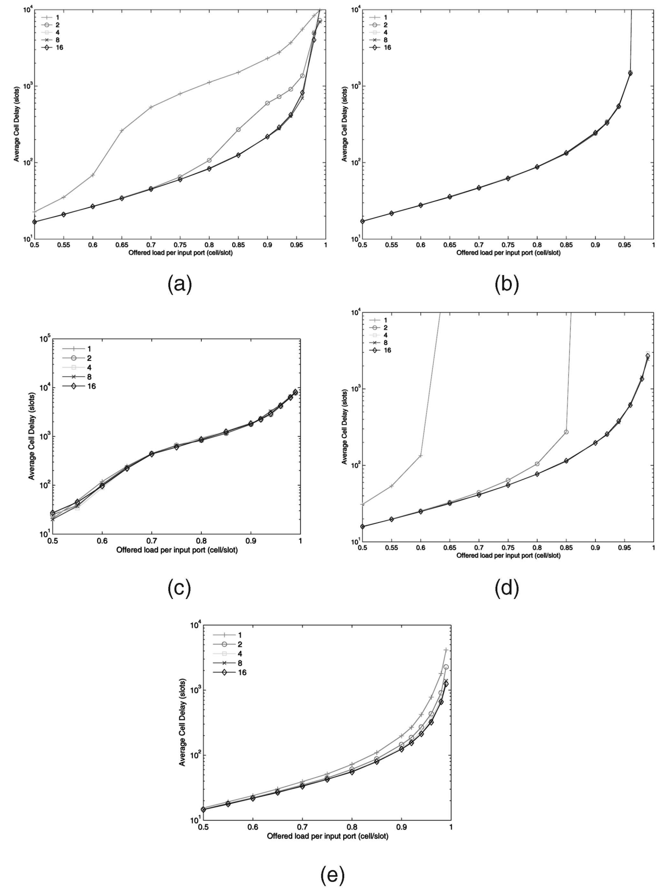

Fig. 11. Effects of increasing the number of iterations under two-state Markov-modulated arrivals with an average burst size of 16. (a) iSLIP. (b) EiSLIP. (c) DRR. (d) PIM. (e) CTR.

oblivious of the state of the other arbiters in the switch; that is, the scheduler does not necessarily adapt to traffic dynamics. Our scheme alleviates this problem by using the cooperative mechanism described earlier.

Although our proposed CTR provides excellent performance for all admissible traffic as previously shown, under *inadmissible* traffic, the CTR scheduler could lead to *starvation* of some queues. An example of starvation behavior is shown in Fig. 12 for a 2 × 2 switch. Because all three queues are permanently occupied, the algorithm will always select the "cross" traffic, input 1 to output 2 and input 2 to output 1, and $VOQ_{1,1}$ will starve.

Although in a real router, decongestion mechanisms (e.g., RED) are applied at the ingress ports to avoid buffer overflow associated with inadmissible traffic, it is still possible to construct an adversarial traffic pattern for CTR that leads to unfairness. There are several mechanisms for providing fairness in a CTR scheduler. One simple scheme is to set a threshold on the number of consecutive time slots for which a node (an input port) can hold an acquired token (e.g., k time slots). This threshold would ensure that each node gets the chance to acquire any token every $k(N-1)$ time slots. Conversely, a node with a VOQ that has not been served beyond a threshold period of time slots may send (broadcast) a "prioritized request," which must be honored by the node that currently acquires this token. If a higher granularity of fairness is desired, then a credit-based mechanism could be used such that a number of credits are allocated to each input-output pair, and each CTR arbiter is allowed to acquire a token only if there are available credits for the corresponding output port. We explore this credit-based scheme in Section 8 to provide rate guarantees in our proposed CTR scheduler.

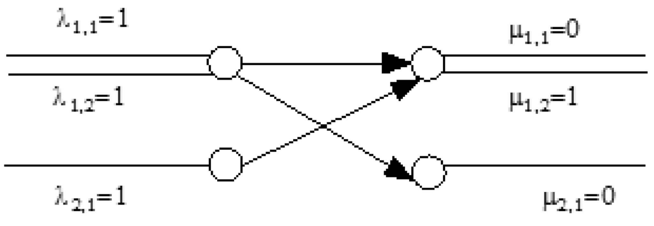

Fig. 12. Under an inadmissible workload, the CTR scheduler will cause $VOQ_{1,1}$ to starve.

8 Weighted Cooperative Token-Ring

To provide both rate guarantees and proportional bandwidth sharing, we propose a two-level scheduler, called the WCTR. In WCTR, QoS traffic is scheduled first, and best effort traffic then contends for the remaining bandwidth.

Scheduling QoS in WCTR is performed using frame-based scheduling such that time is divided into frames, and a counter is associated with each input-output pair. The counters are set according to their negotiated bandwidth shares at the beginning of each frame. Queues with positive counters compete with higher priority according to CTR. Then, the remaining queues contend according to CTR for the available bandwidth. During any time slot, an input can acquire a token for an output port to send either guaranteed-rate traffic or best effort traffic, and a flag is used to indicate the traffic type for which the token is acquired. Scheduling in each level (QoS or best effort) is performed similar to the original CTR description in Section 4: computing the TRV phase followed by a token propagation/acquisition phase. The main difference is that QoS traffic is prioritized over best effort traffic. First, the TRVs are computed for QoS traffic with the semantics that an arbiter sends a token request only if it has QoS traffic and available credit. Subsequently, if an arbiter had previously acquired a token for best effort traffic and the acquired token is now requested by another unmatched input for QoS traffic, then it must release it. Conversely, during the best effort scheduling level, a WCTR arbiter would not release a token that was acquired for a QoS traffic if this token is requested by another input.

There are many variations of the presented WCTR scheduler; for example, it is straightforward to generalize the scheme to multiple priority levels. In addition, a centralized module could be used allocate credits for best effort traffic to ensure fairness in the distribution of unreserved bandwidth among all input ports.

8.1 Simulation Results for the Weighted Cooperative Token-Ring Scheduler

To illustrate the fairness of WCTR in bandwidth allocation, a 4 × 4 switch was simulated such that each input has four flows, each going to a different output with a different

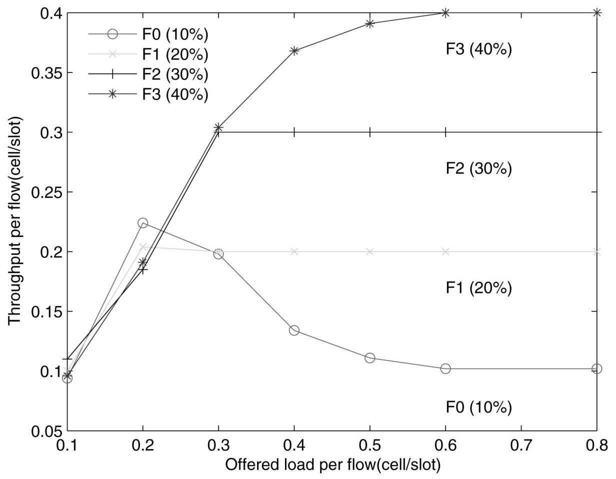

Fig. 13. Plot of throughput per flow for WCTR at the end of one frame.

bandwidth reservation. Let $f_k(i,j)$ represent flow k from input port i to output port j. In the simulated switch, $f_1(0,0)$, $f_2(1,0)$, $f_3(2,0)$, and $f_4(3,0)$ have reserved 10, 20, 30, and 40 percent of the bandwidth, respectively, but they always maintain the same actual arrival rate. Other flows have a load of 5 percent each. This traffic model has been used in [27] and [7]. We used a frame size of 1,000 slots and measured the throughput per flow at the end of one frame. As shown in Fig. 13, WCTR is able to provide to each flow its allocated rate.

9 Hardware Implementation of Cooperative Token-Ring

In this section, we analyze the hardware complexity of computing the TRVs and prove that its latency is $\Theta(\log N)$ and the circuit size per node is $\Theta(N \log N)$. Recall from Section 4.1 that each element in the TRV is given by

$$TRV_{i,j} = R_{|i+1|} + \sum_{j=i+2}^{j=i+N-1} R_{|j|} \prod_{k=i+1}^{k=j-1} \overline{TP}_{|k|}.$$

We make the following two observations:

1. Equation (1) cannot be directly implemented using a parallel prefix circuit [28] because each element is computed using all the other elements in the ring in a circular (modulo arithmetic) fashion.
2. Although it is simple to achieve an optimal latency of $\Theta(\log N)$ by using a separate binary tree to evaluate (1) for each element of a TRV, the circuit size per TRV element would be $\Theta(N \log N)$, and consequently, the circuit size per token (i.e., an entire vector) would be $\Theta(N^2 \log N)$, whereas we describe a technique with optimal latency and circuit size per token of $\Theta(N \log N)$.

Rather than providing a specific solution for computing (1), we generalize the problem and present a generic circuit for computing all elements in a list such that each element depends on other elements in the list, in a circular fashion,

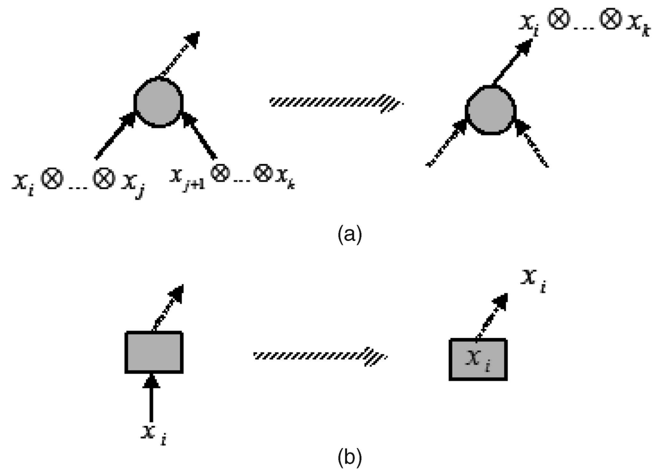
$x_i \otimes ... \otimes x_k$
$x_i \otimes ... \otimes x_j$
$x_{j+1} \otimes ... \otimes x_k$
(a)
x_i
x_i
x_i
(b)

Fig. 14. Action of (a) an internal node and (b) a leaf during the upward phase. Inputs to the internal nodes are concatenated and then passed upward. Inputs to leaves are stored and passed upward.

with a circuit size of $\Theta(N \log N)$ and with a time complexity of $\Theta(\log N)$, as described in Section 9.1.

9.1 Complete Symmetrical Prefix Problem

Let $\otimes$ be a binary associative operation. The complete symmetrical prefix problem is to compute, given $x_1, x_2, \ldots, x_n$, the results $y_1, y_2, \ldots, y_n$, where $y_k = x_{|k+1|} \otimes x_{|k+2|} \otimes \ldots \otimes x_{|k+n-1|}$, for $1 \le k \le n$.

The problem of computing the token request bit can be solved on a binary tree network in $2D$ steps, where D is the depth of the tree. The algorithm consists of essentially two interleaving phases: upward phase and downward phase. In the upward phase, each internal node computes the product of the entries in the leaves spanned by the node. In the downward phase, these products are passed downward the tree so that the leaf can form the result. We describe the algorithm in more detail in the next sections.

9.1.1 Upward Phase

During the first step, x_i is input to the ith leaf for $1 \le i \le N$. This value is both stored and passed upward to the node's parent. In subsequent steps, internal nodes receiving inputs from children concatenate the inputs and pass the product upward in the tree. After D steps, every node, other than the root, will have computed the product of the inputs to the leaves covered by the node. Fig. 14 shows the computation

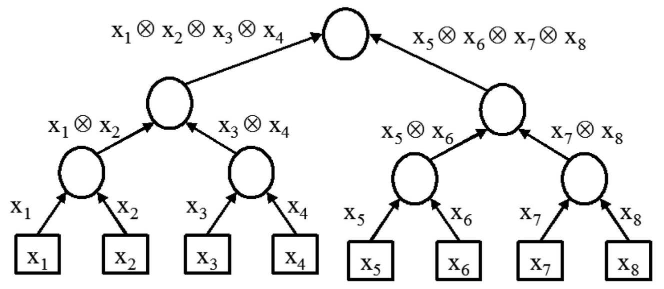
$x_1 \otimes x_2 \otimes x_3 \otimes x_4$
$x_5 \otimes x_6 \otimes x_7 \otimes x_8$
$x_1 \otimes x_2$
$x_3 \otimes x_4$
$x_5 \otimes x_6$
$x_7 \otimes x_8$
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8

Fig. 15. Concatenation performed by each node in the complete symmetrical parallel prefix algorithm for an eight-element string. Each node computes the product of the inputs that it spans.

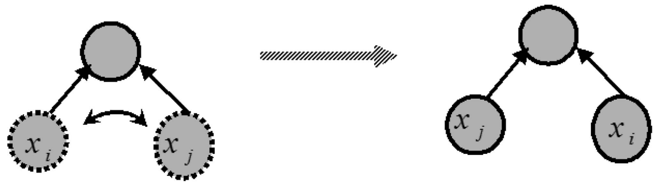

Fig. 16. Swapping the node values during the upward propagation phase.

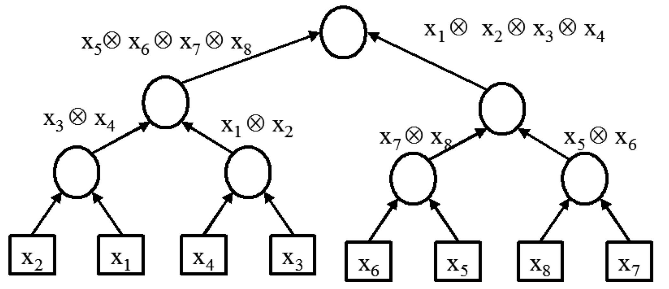

Fig. 17. The node values after swapping the left and right child for the eight-element string used in Fig. 15.

performed by each node, and Fig. 15 shows the actual products computed by each node for an eight-element string example.

As each nonleaf node receives its input from below, it swaps the values computed by the left and right child nodes. That is, it passes the value computed by the left child to the right child and conversely. Each node then stores the new value it receives from its parent, as shown in Fig. 16.

For example, Fig. 17 shows the node values after swapping for the eight-element string example.

9.1.2 Downward Phase

During the downward phase each node receives the node value of its parent and computes the result, as depicted in Fig. 18. The operation is performed for both leaf and nonleaf nodes.

In total, the algorithm takes 2D steps, where D is the depth of the tree, and the circuit size is determined by the number of nodes in the tree, which is $\Theta(N \log N)$ given that each node has a fixed degree of two.

9.2 Computing the Token Request Vector as a Complete Symmetrical Prefix Problem

Computing the TRV can be modeled as the complete symmetrical prefix problem by keeping track of whether each node stage *stops* a request, *propagates* a request, or *generates* a request. Specifically, let the state at node i be

- stop(s) $\equiv$ if the token is currently at node i,
- propagate(p) $\equiv$ if the token is not currently at node i and the node does not have a request for the token, and
- generate(g) $\equiv$ if node i has a request for the token.

Let x_i denote the s, p, or g value of ith node and let $|k| = (k \; mod \; N)$.

Next, let $TRV_i = x_{|i+1|} \otimes x_{|i+2|} \otimes x_{|i+3|}, \ldots, \otimes x_{|i+N-1|}$ for $1 \leq i \leq N$, where $\otimes$ is the binary associative operator defined in Table 1.

For example, the signal states for the nodes used in Fig. 19a are shown in Table 2. The token request bit at module x_3 is given by

$$\begin{aligned} x_3 &= x_4 \otimes x_5 \otimes x_6 \otimes x_7 \otimes x_8 \otimes x_1 \otimes x_2, \\ x_3 &= p \otimes g \otimes p \otimes p \otimes s \otimes p \otimes g, \\ x_3 &= g. \end{aligned}$$

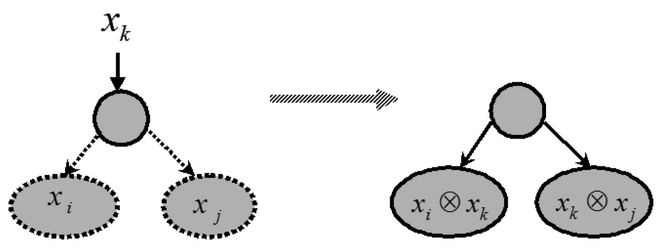

Fig. 18. Operation of nodes (leaf and nonleaf nodes) during the downward phase.

TABLE 1
Multiplication Table for the Operator Defined for Computing the TRV

$\otimes$	S	P	G
S	S	S	S
P	S	P	G
G	G	G	G

For example, $S \otimes G = S$.

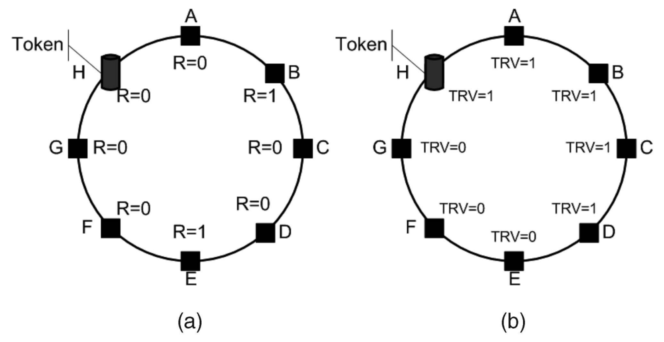

Fig. 19. (a) The request bit at each node. (b) The corresponding value of TRV at each node.

TABLE 2
The Signal States for the Modules in Fig. 19

Module	Signal
$x_1 \equiv x_A$	P
$x_2 \equiv x_B$	G
$x_3 \equiv x_C$	P
$x_4 \equiv x_D$	P
$x_5 \equiv x_E$	G
$x_6 \equiv x_F$	P
$x_7 \equiv x_G$	P
$x_8 \equiv x_H$	S

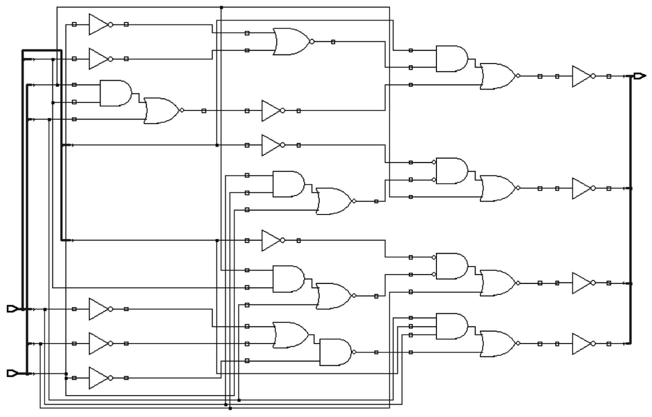

Fig. 20. Circuit for computing the TRV in a ring with four nodes.

Fig. 20 shows the circuit for computing the TRV for a ring with four nodes; this circuit was generated using Synopsys,[2] with the optimization setting for high speed. The circuit computes the TRV for single token in a ring with four nodes; i.e., it computes the 4 bit elements in the TRV.

10 Conclusion

We have proposed CTR scheduling, a novel scheduling paradigm that provides significant improvement over existing schedulers with comparable complexity. Our scheduling paradigm adapts to dynamically varying traffic, provides high throughput, and is easily implemented in hardware. We have analyzed the hardware complexity introduced by the cooperative mechanism and described an optimal hardware implementation for an $N \times N$ switch with a time complexity of $\Theta(\log N)$ and a circuit size of $\Theta(N \log N)$ per port. The CTR scheduling policy provides essentially 100 percent throughput over uniform and nonuniform traffic patterns, whereas the iSLIP and EiSLIP saturate at approximately 80-85 percent traffic load. We have proposed WCTR to provide rate guarantees in IQ switches and proportional bandwidth sharing. Finally, we note that although CTR was presented in the context of IQ switches, it is applicable to several other systems, including SONET all-optical circuit switches, which schedule cells in circuit-based frames by using delay lines and a star-based WDM broadcast-and-select optical system with tunable transmitters and fixed receivers. Generally, the proposed CTR scheme can be applied to solve any resource allocation problem with a set of nodes competing for exclusive access to a set of shared resources.

Acknowledgments

This work was supported by the Bell Canada Chair in Data Communications, Natural Sciences and Engineering Research Council of Canada (NSERC), and the Ontario Center of Excellence for Communication and Information Technology Ontario (OCE-CITO). This work was done while A. Gourgy was with McMaster University.

2. Synopsys is a trademark of Synopsys, Inc.

References

[1] *Cisco 12000 Series-Internet Routers,* http://www.cisco.com, 2004.

[2] C. Partridge, P.P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, T. Ma, J. Mcallen, T. Mendez, W.C. Milliken, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober, G.D. Troxel, D. Waitzman, and S. Winterble, "A 50-Gb/s IP Router," *IEEE/ACM Trans. Networking,* vol. 6, no. 3, pp. 237-248, June 1998.

[3] M. Karol, M. Hluchyj, and S. Morgan, "Input versus Output Queueing on a Space-Division Switch," *IEEE Trans. Comm.,* vol. 35, pp. 1347-1356, Dec. 1987.

[4] T. Anderson, S.S. Owicki, J.B. Saxe, and C.P. Thacker, "High Speed Switch Scheduling for Local Area Networks," *ACM Trans. Computer Systems,* vol. 11, no. 4, pp. 319-352, Nov. 1993.

[5] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *IEEE Trans. Comm.,* vol. 47, no. 8, pp. 1260-1267, Aug. 1999.

[6] N. Nan and L.N. Bhuyan, "Fair Scheduling in Internet Routers," *IEEE Trans. Computers,* vol. 51, no. 6, pp. 686-701, June 2002.

[7] X. Zhang and L.N. Bhuyan, "Deficit Round Robin Scheduling for Input-Queued Switches," *IEEE J. Selected Areas in Comm.,* vol. 21, no. 4, pp. 584-594, May 2003.

[8] D. Pan and Y. Yang, "FIFO-Based Multicast Scheduling Algorithm for Virtual Output Queued Packet Switches," *IEEE Trans. Computers,* vol. 54, no. 10, pp. 1283-1297, Oct. 2005.

[9] N. McKeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Trans. Networking,* vol. 7, no. 2, pp. 188-201, Apr. 1999.

[10] H.J. Chao, "Saturn: A Terabit Packet Switch Using Dual Round-Robin," *IEEE Comm. Magazine,* vol. 38, no. 12, pp. 78-84, Dec. 2000.

[11] Y. Li, "Design and Analysis of Scheduling for High Speed Input Queued Switches," PhD dissertation, Polytechnic Univ., Jan. 2004.

[12] C.S. Chang, D.S. Lee, and Y.S. Jou, "Load Balanced Birkhoff-Von Neumann Switches, Part I: One-Stage Buffering," *Computer Comm.,* vol. 25, pp. 611-622, 2002.

[13] D. Serpanos and P. Antoniadis, "Firm: A Class of Distributed Scheduling Algorithms for High-Speed ATM Switches with Multiple Input Queues," *Proc. IEEE INFOCOM '00,* vol. 2, pp. 548-555, Mar. 2000.

[14] N. McKeown, "Scheduling Algorithms for Input-Queued Cell Switches," PhD dissertation, Univ. of California, Berkeley, 1995.

[15] A. Mekkittikul, "Scheduling Non-Uniform Traffic in High Speed Packet Switches and Routers," PhD dissertation, Stanford Univ., Nov. 1998.

[16] M. Marsan, A. Bianco, E. Leonardi, and L. Milia, "RPA: A Flexible Scheduling Algorithm for Input Buffered Switches," *IEEE Trans. Comm.,* vol. 47, no. 12, pp. 1921-1933, Dec. 1999.

[17] H. Duan, J.W. Lockwood, and S.M. Kang, "Matrix Unit Cell Scheduler (MUCS) for Input-Buffered ATM Switches," *IEEE Comm. Letters,* vol. 2, no. 1, pp. 20-23, Jan. 1998.

[18] L. Tassiulas, "Linear Complexity Algorithms for Maximum Throughput in Radio Networks and Input Queued Switches," *Proc. IEEE INFOCOM '98,* pp. 533-539, 1998.

[19] P. Giaccone, B. Prabhakar, and D. Shah, "Randomized Scheduling Algorithms for High-Aggregate Bandwidth Switches," *IEEE J. Selected Areas Comm.,* vol. 21, no. 4, pp. 546-559, May 2003.

[20] Y. Li, S. Panwar, and J.H. Chao, "The Dual Round-Robin Matching Switch with Exhaustive Service," *Proc. Workshop High Performance Switching and Routing (HPSR '02),* pp. 58-63, May 2002.

[21] A. Gourgy, "On Packet Switch Scheduling in High-Speed Data Networks," PhD dissertation, McMaster Univ., Mar. 2006.

[22] P. Gupta and N. McKeown, "Design and Implementation of a Fast Crossbar Scheduler," *IEEE Micro,* vol. 19, no. 1, pp. 20-28, Jan./Feb. 1999.

[23] N.P. Forum, *Switch Fabric Benchmarking Group Documents: Switch Fabric Benchmark Test Suites (NPF 2002.276.08), Performance Testing Methodology for Fabric Benchmarking (NPF 2003.213.06), Fabric Benchmarking Traffic Models, Fabric Benchmarking Performance Metrics, Switch Fabric Benchmarking Framework,* http://www.npforum.org/, 2004.

[24] M.E. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Trans. Networking,* vol. 5, no. 6, pp. 835-846, Dec. 1997.

[25] Y. Li, S. Panwar, and H.J. Chao, "On the Performance of a Dual Round-Robin Switch," *Proc. IEEE INFOCOM '01,* pp. 1688-1697, Apr. 2001.

[26] Y. Kim and H.J. Chao, "Performance of Exhaustive Matching Algorithms for Input-Queued Switches," *Proc. IEEE Int'l Conf. Comm. (ICC '03),* vol. 3, pp. 1817-1822, May 2003.

[27] D. Stiliadis and A. Varma, "Providing Bandwidth Guarantees in an Input-Buffered Crossbar Switch," *Proc. IEEE INFOCOM '95,* pp. 960-968, Apr. 1995.

[28] R. Ladner and M. Fischer, "Parallel Prefix Computation," *J. ACM,* vol. 27, no. 4, pp. 831-838, Oct. 1980.

Amir Gourgy received the BEng degree in computer engineering from McMaster University, the MASc degree in computer engineering from University of Waterloo, and the PhD degree in electrical engineering from McMaster University in 2006. He is currently with Research in Motion Inc., Waterloo, Ontario. He is a member of the IEEE.

Ted H. Szymanski received the PhD degree from the University of Toronto. He currently holds the position of L.R. Wilson/Bell Canada Enterprises Chair in Data Communications in the Department of Electrical and Computer Engineering, McMaster University, Hamilton, Ontario, Canada. He has held professorial positions at Columbia University, New York, and McGill University, Montreal. From 1993 to 2003, he was a principal architect of a 10-year national research program on photonic systems, funded by the Canadian Networks of Centers of Excellence program. The program brought together significant industrial and academic collaborators, including Nortel Networks, Newbridge Networks (now Alcatel), Lucent Technologies, Lockheed-Martin/Sanders, McGill University, McMaster University, the University of Toronto, and Heriot-Watt University, and resulted in the demonstration of a free-space "intelligent optical backplane" exploiting emerging optoelectronic technologies. The four-node system demonstrated 1,024 microoptical laser channels operating in a ring architecture with 1 cm^2 of cross-sectional free space. Since 2003, his research interests have expanded to include network quality of services in support of emerging telemedicine and telerobotic control systems. He holds two patents and one pending in the area of networks, and he has consulted for several companies. He has also served as associate chair (undergraduate) in the ECE Department. He is a member of the IEEE.