

A Distributed 100% Throughput Algorithm for a Buffered Crossbar Switch

Shunyuan Ye, Yanming Shen, Shivendra Panwar

Abstract

The promise of a buffered crossbar switch - a crossbar switch with a packet buffer at each crosspoint - is that it can provide good delay performance with much less complex, practical scheduling algorithms. With today's technology, it is now possible to implement it in a single chip. Thus it has attracted great attentions recently. Though simple distributed algorithms like dual round robin can achieve 100% throughput under uniform traffic, so far there are no distributed algorithms which can achieve 100% throughput under general arrival patterns. In this paper, we propose a distributed scheduling algorithm which we prove to achieve 100% throughput for any admissible arrival matrix. To the best of our knowledge, this is the first distributed algorithm which can stabilize the system under any admissible input traffic. Our simulation results also show that the algorithm can provide good delay performance for different traffic patterns.

I. INTRODUCTION

The fast growing demand of Internet traffic requires that packet switches should be simple, fast and efficient. Due to the memory speed limit, most of switches now are using *input queueing* (IQ) or *combined input and output queueing* (CIOQ), with a bufferless crossbar switching fabric. The scheduler must find a matching between inputs and outputs. Such switches usually require centralized, sometimes complex, algorithms to achieve good performance, such as *maximal* [1], *maximum size* [2] and *maximum weight matching* [3], [4]. Maximum weight matching can achieve 100% throughput for any admissible arrivals, but it is not practical to implement due to its high complexity. Maximal matching, on the other hand, cannot achieve as high a throughput as maximum weight matching. A number of practical iterative algorithms have been proposed, such as PIM [5], iSLIP [6] and DRRM [7]. PIM and iSLIP use multiple iterations to converge on a maximal matching. DRRM can achieve 100% throughput only under i.i.d. and uniform traffic.

With today's ASIC technology, it is now possible to add some limited buffers at each crosspoint inside the crossbar. This makes the *buffered crossbar switch* a much more attractive architecture since its

scheduler is much simpler. The input and output schedulers can potentially be independent. First, each input picks a crosspoint buffer to send a packet to. Then, each output picks a crosspoint buffer to transmit a packet from, which is shown in Fig. 1. A centralized scheduler is not needed since the processing can be distributed at each input and output. It has been shown that simple algorithms such as *round robin* at both the inputs and outputs (RR-RR) [8], or *longest queue first* at the inputs and *round robin* at the outputs (LQF-RR) [9] can provide 100% throughput under uniform traffic.

The authors in [10] showed that with a speedup of 2, a buffered crossbar can provide 100% throughput for any admissible arrivals. The results were extended to variable packet sizes in [11]. In [12], the authors proved that the speedup requirement can be reduced to $2 - \frac{1}{N}$. But without speedup, these algorithms can only achieve maximum throughput under uniform traffic. In [13], the authors proposed a practical algorithm called *stable queue input-output scheduler with Hamiltonian walk (SQUISH)*, which can achieve 100% throughput for any admissible traffic. It requires finite buffers and no speedup. The complexity is $O(\log N)$. But it is a centralized algorithm which does not scale with the increase in the number of ports due to the communication complexity.

It is still an open problem whether there exists any distributed scheduling algorithms which can achieve 100% throughput under any admissible traffic. Recently, it has been shown that CSMA-like algorithms can achieve the maximum possible throughput in wireless ad hoc networks [14]–[16] in continuous time systems. The ideas are extended to discrete time system by Ni and Srikant in [17], [18]. Inspired by the CSMA-like algorithms, in this paper, we propose a distributed algorithm in buffered crossbar switches, which can stabilize the system under any admissible arrivals. Using our new algorithm, no message passing is required. Each input only uses its local queue information and the previous time slot schedule to make its scheduling decision. A Hamiltonian walk is used to implicitly coordinate the inputs and outputs. We prove the stability of the system and evaluate the performance of the algorithm by running extensive simulations. To the best of our knowledge, this is the first distributed algorithm which can achieve 100% throughput under any admissible Bernoulli traffic in buffered crossbar switches. The algorithm is simple and easy to implement in a real router. The simulation results also show that it can provide good delay performance, compared to output queued switches, under different types of traffic.

The rest of paper is organized as follows. We first briefly describe the buffered crossbar switch in Sec. II. The new scheduling algorithm is proposed in Sec. III, and an example is given to better describe the idea. We derive the stationary distribution of the system in Sec. IV and prove its stability in Sec. V.

Simulations results are presented in Sec. VI to verify our algorithm and show its delay performance under different traffic arrivals.

II. CROSSPOINT BUFFERED SWITCH

An $N \times N$ switch is shown in Fig. 1. We assume fixed size packet switching. Variable size packets can be segmented into cells before switching and be reassembled at the output ports. There are *virtual output queues (VOQs)* at the inputs to prevent the head-of-line blocking. Each input maintains N VOQs, one for each output. Let VOQ_{ij} represent the VOQ at input i for output j . Let $Q_{ij}(n)$ denote the queue length of VOQ_{ij} at time n .

Each crosspoint has a buffer of size one. Let CB_{ij} denote the buffer of the crosspoint between input i and output j . $B_{ij}(n) \in \{0, 1\}$ denotes the occupancy of CB_{ij} at time n .

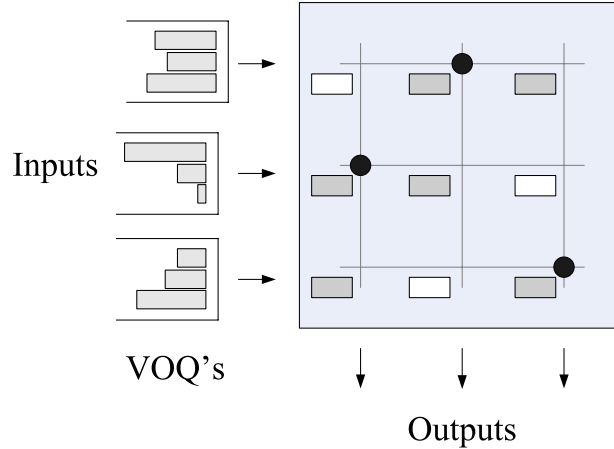


Fig. 1: Buffered Crossbar Switch

A schedule can be represented by $\mathbf{S}(n) = [\mathbf{S}^I(n), \mathbf{S}^O(n)]$. $\mathbf{S}^I(n) = [S_{ij}^I(n)]$ is the input schedule. Each input port can only transmit at most one cell at each time slot. Thus the input schedule is subject to the following constraints:

$$\sum_j S_{ij}^I(n) \leq 1, \quad S_{ij}^I(n) = 0 \text{ if } B_{ij}(n) = 1. \quad (1)$$

$\mathbf{S}^O(n) = [S_{ij}^O(n)]$ is the output schedule. It has to satisfy the following constraints:

$$\sum_i S_{ij}^O(n) \leq 1, \quad S_{ij}^O(n) = 0 \text{ if } B_{ij}(n) = 0. \quad (2)$$

We next define a class of scheduling policies called *priority schedule*. A priority schedule can be represented by a matrix $\mathbf{X} \in \{0, 1\}^{N \times N}$. $X_{ij} = 1$ if the crosspoint between input i and output j is in the

schedule; otherwise, $X_{ij} = 0$. With some abuse of notation, we also use \mathbf{X} to represent a set, and write $(i, j) \in \mathbf{X}$ if $X_{ij} = 1$. As we will present in Sec. III, following the new scheduling algorithm, if $X_{ij} = 1$ and $Q_{ij}(n) > 0$, then $S_{ij}^I(n) = 1$ and $S_{ij}^O(n) = 1$. In order to satisfy the constraints of Eq. (1) and (2), a *feasible priority schedule* is defined as:

Definition 1: A *feasible priority schedule* $\mathbf{X}(n)$ is an $N \times N$ matrix, where $X_{ij}(n) \in \{0, 1\}$, and $\sum_i X_{ij}(n) \leq 1$, $\sum_j X_{ij}(n) \leq 1$.

Note that a feasible priority schedule \mathbf{X} has the property that if $X_{ij} = 1$, then $\forall i' \neq i, X_{i'j} = 0$ and $\forall j' \neq j, X_{ij'} = 0$. We define these conflicting crosspoints as its neighbors.

Definition 2: For a crosspoint (i, j) , its neighbors are defined as:

$$\mathcal{N}(i, j) = \{(i', j) \text{ or } (i, j') \mid \forall i' \neq i, \forall j' \neq j\} \quad (3)$$

So for a feasible priority schedule \mathbf{X} , if $(i, j) \in \mathbf{X}$, then $\forall (k, l) \in \mathcal{N}(i, j), (k, l) \notin \mathbf{X}$. Let \mathcal{X} represent the set of all feasible priority schedules.

Let λ_{ij} represent the arrival rate of traffic between input i and output j . We assume that the arrival is a Bernoulli process.

Definition 3: An arrival process is said to be *admissible* if it satisfy:

$$\sum_j \lambda_{ij} < 1, \text{ and } \sum_i \lambda_{ij} < 1. \quad (4)$$

An algorithm is said to *achieve 100% throughput* if it can stabilize the system for any admissible traffic.

III. SCHEDULING ALGORITHM

A. The Basic Algorithm

In our algorithm, each input i only needs to keep track of the priority schedule in the previous slot, i.e. for which output j was $X_{ij}(n-1) = 1$. Similarly, each output only needs to keep track of for which input i was $X_{ij}(n-1) = 1$.

Let $\mathbf{X}(n)$ be the priority schedule at time n . At each time slot, generate a Hamiltonian walk schedule $\mathbf{H}(n)$ [19].

- At each input port i , assume VOQ_{ij} is the queue selected by $\mathbf{H}(n)$, then if there exists a j' , with $X_{ij'}(n-1) = 1$,
 - If $j = j'$, then $X_{ij}(n) = 1$ with probability p_{ij} ; 0 with probability of $\bar{p}_{ij} = 1 - p_{ij}$.

– Else, $X_{ij}(n) = 0$.

Else, if there is no j' such that $X_{ij'}(n-1) = 1$,

– If $X_{i'j}(n-1) = 0, \forall i'$ (we will explain how an input port can learn this next), then $X_{ij}(n) = 1$ with probability p_{ij} ; 0 with probability $\bar{p}_{ij} = 1 - p_{ij}$.

– Else, if there exists an i' such that $X_{i'j}(n-1) = 1$, $X_{ij}(n) = 0$.

• At each output port j , assume VOQ_{ij} is selected by $\mathbf{H}(n)$, then if there exists an i' , with $X_{i'j}(n-1) = 1$,

– If $i = i'$, then $X_{ij}(n) = 1$ if input i transmits a packet to crosspoint buffer CB_{ij} ; 0 otherwise.

– Else, if $i \neq i'$, $X_{ij}(n) = 0$.

Else, if there is no i' such that $X_{i'j}(n-1) = 1$, output j has to learn input i 's decision by observing crosspoint buffer CB_{ij} .

– If $B_{ij}(n-1) = 0$ and input i sends a packet to CB_{ij} at time n , output j learns that $X_{ij}(n) = 1$.

If $B_{ij}(n-1) = 1$, output j has to transmit this packet from CB_{ij} at time n . If input i sends a packet to CB_{ij} at the beginning of time $n+1$, output j learns that $X_{ij}(n) = 1$.

– Else, $X_{ij}(n) = 0$.

The key point in our algorithm is that by using crosspoint buffers, an input and an output can learn each other's decision. If an input i decides to set $X_{ij}(n) = 1$, it has to make sure that there does not exist an i' such that $X_{i'j}(n-1) = 1$. It can do this by observing whether the crosspoint buffer CB_{ij} is served by output port j at time n . Similarly, an output can learn about an input decision by observing if the input sends a packet to the crosspoint buffer. Note that in our algorithm, $X_{ij}(n)$ can change only when the VOQ_{ij} is selected by $\mathbf{H}(n)$.

After a new priority schedule is generated, each input and output port decides their schedules using the following rules:

• For input i , if $X_{ij}(n) = 1$ and $Q_{ij}(n) > 0$, then $S_{ij}^I(n) = 1$. If $X_{ij}(n-1) = 1$ and VOQ_{ij} is selected by $\mathbf{H}(n)$, according the algorithm above, $X_{ij}(n) = 1$ with probability p_{ij} . In this case, if $X_{ij}(n) = 1$ and $Q_{ij}(n) = 0$, a dummy packet should be transmitted so that output j can learn input i 's decision. Dummy packets are discarded at output ports.

• For output j , if $X_{ij}(n) = 1$ and $B_{ij}(n) > 0$, $S_{ij}^O(n) = 1$.

• For input i , or output j , if $\sum_j X_{ij}(n) = 0$ or $\sum_i X_{ij}(n) = 0$, they are considered as *free* input/output ports. Free input/output ports can randomly pick an eligible VOQ or non-empty crosspoint buffer to

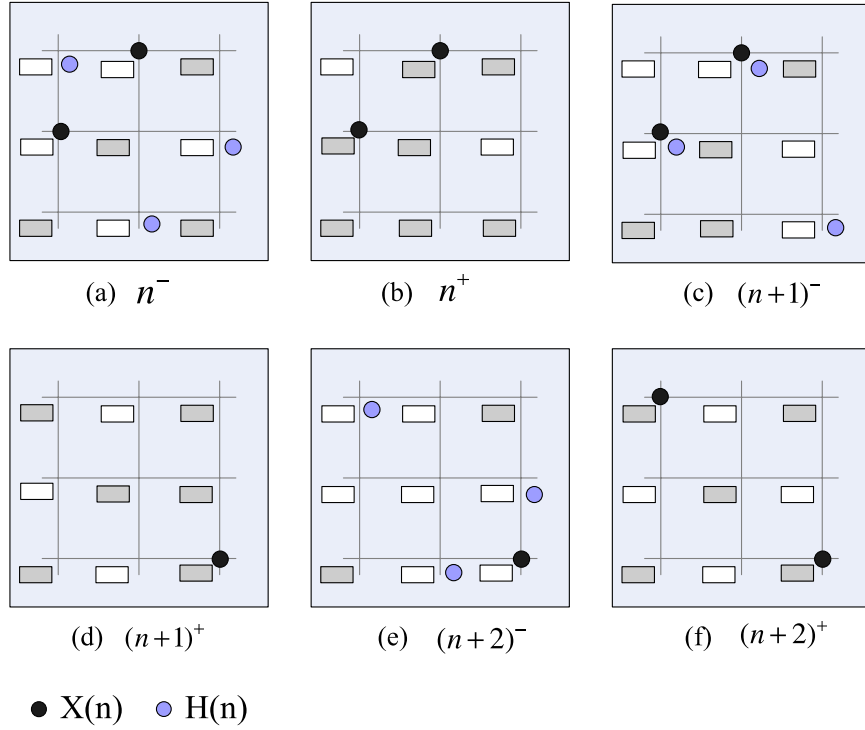


Fig. 2: Example

serve.

B. An Example

In order to better describe the scheduling algorithm, we give the following example, as shown in Fig. 2. Suppose that the input actions are done at the beginning of each time slot and outputs transmit packets from the crosspoint buffers before the end of each time slot.

- In Fig. 2(a), the priority schedule is $\mathbf{X}(n-1) = \{(1, 2), (2, 1)\}$ and the Hamiltonian walk schedule is $\mathbf{H}(n) = \{(1, 1), (2, 3), (3, 2)\}$. For input 1, $X_{12}(n-1) = 1$ and $(1, 2)$ is not selected by $\mathbf{H}(n)$, so $X_{12}(n) = X_{12}(n-1) = 1$. Similarly, $X_{21}(n) = 1$. Input 3 is free and since $(3, 2)$ is selected, it decides to send a packet to CB_{32} with probability p_{32} . But output 2 is not free, thus the packet in CB_{32} will not be transmitted. Input 3 can observe this by the end of time n . $X_{32}(n) = 0$. So the priority schedule at time n is still $\mathbf{X}(n) = \{(1, 2), (2, 1)\}$.
- At time $n+1$, $\mathbf{H}(n+1) = \{(1, 2), (2, 1), (3, 3)\}$. Both $(1, 2)$ and $(2, 1)$ are selected by $\mathbf{H}(n+1)$. So input 1 and 2 change their scheduling decisions with probability $\bar{p}_{12} = 1 - p_{12}$ and $\bar{p}_{21} = 1 - p_{21}$. In the example, they both decide to change their scheduling decisions. Therefore, $X_{12}(n+1) = 0$ and $X_{21}(n+1) = 0$. Input 3 is free and $(3, 3)$ is selected by $\mathbf{H}(n+1)$. So input 3 sends a packet to CB_{33} with probability p_{33} . Output 3 is free and it learns that $X_{33}(n+1) = 1$ by observing CB_{33} .

Then the priority schedule becomes $\mathbf{X}(n+1) = \{(3, 3)\}$, as shown Fig. 2(d).

- At time $n+2$, $\mathbf{H}(n+2) = \{(1, 1), (2, 3), (3, 2)\}$. Input 1 is free, and $(1, 1)$ is selected by $\mathbf{H}(n+2)$. So it sends a packet to CB_{11} with probability p_{11} . Output 1 is also free, and it learns that $X_{11}(n+2) = 1$ by observing CB_{11} . Input 2 is free and since $(2, 3)$ is selected by $\mathbf{H}(n+2)$, it has to decide whether to send a packet to CB_{23} or not with probability p_{23} . As we can see, input 2 decides not to send a packet to CB_{23} , $X_{23}(n+2) = 0$. $(3, 3)$ is not in $\mathbf{H}(n+2)$, so $X_{33}(n+2) = X_{33}(n+1) = 1$. The priority schedule then is $\mathbf{X}(n+2) = \{(1, 1), (3, 3)\}$.

IV. STATIONARY DISTRIBUTION

Lemma 1: If $\mathbf{X}(n-1) \in \mathcal{X}$, then $\mathbf{X}(n) \in \mathcal{X}$.

Proof: As defined, \mathbf{X} is a feasible priority schedule if and only if $\forall(i, j)$ such that $X_{ij} = 1: \forall(k, l) \in \mathcal{N}(i, j), (k, l) \notin \mathbf{X}$.

For any (i, j) such that $X_{ij}(n) = 1$, it belongs to one of those two cases below:

- 1) $X_{ij}(n-1) = 1$.
- 2) $X_{ij}(n-1) = 0$ and $(i, j) \in \mathbf{H}(n)$.

If $X_{ij}(n-1) = 1, \forall(k, l) \in \mathcal{N}(i, j), X_{kl}(n-1) = 0$. According to the scheduling algorithm, no matter $(k, l) \in \mathbf{H}(n)$ or not, we have $X_{kl}(n) = X_{kl}(n-1) = 0$.

If $X_{ij}(n-1) = 0$ and $(i, j) \in \mathbf{H}(n)$, according to the scheduling algorithm, $\forall(k, l) \in \mathcal{N}(i, j), X_{kl}(n-1) = 0$. Since $\mathbf{H}(n)$ is also a feasible schedule $\forall(k, l) \in \mathcal{N}(i, j), (k, l) \notin \mathbf{H}(n)$. Therefore, we have $X_{kl}(n) = X_{kl}(n-1) = 0$.

For any (i, j) such that $X_{ij}(n) = 1$, we have proved that $\forall(k, l) \in \mathcal{N}(i, j), X_{kl}(n) = 0$. So $\mathbf{X}(n)$ is feasible if $\mathbf{X}(n-1)$ is feasible. **QED** ■

Lemma 2: A priority schedule $\mathbf{X} \in \mathcal{X}$ can transit to a schedule \mathbf{X}' if and only if $\mathbf{X} \cup \mathbf{X}' \in \mathcal{X}$.

Proof: (Necessity): Suppose that $\mathbf{X} \cup \mathbf{X}' \notin \mathcal{X}$. According to the definition of feasible priority schedule, there exists at least one $(i, j) \in \mathbf{X} \cap \overline{\mathbf{X}'}$ and $(k, l) \in \overline{\mathbf{X}} \cap \mathbf{X}'$ such that $(k, l) \in \mathcal{N}(i, j)$. As shown in Fig. 3, in a transition from \mathbf{X} to \mathbf{X}' , both $(\mathbf{X} \cap \overline{\mathbf{X}'})$ and $(\overline{\mathbf{X}} \cap \mathbf{X}')$ change their states. According to the scheduling algorithm, a VOQ can change its scheduling decision only when it is selected by $\mathbf{H}(n)$. Since both (i, j) and (k, l) change their states, they should be both in $\mathbf{H}(n)$. But $\mathbf{H}(n)$ is a feasible schedule generated by Hamiltonian walk, which means if $(i, j) \in \mathbf{H}(n)$ and $(k, l) \in \mathcal{N}(i, j)$, (k, l) can not be in $\mathbf{H}(n)$. Contradiction occurs. So, a priority schedule $\mathbf{X} \in \mathcal{X}$ can not transit to a schedule \mathbf{X}' when $\mathbf{X} \cup \mathbf{X}' \notin \mathcal{X}$.

(*Sufficiency*): Suppose that \mathbf{X}' is a priority schedule that $\mathbf{X} \cup \mathbf{X}' \in \mathcal{X}$. Since $(\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}') \subseteq \mathbf{X} \cup \mathbf{X}'$, we then have $(\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}') \in \mathcal{X}$. Therefore, there exists at least one $\mathbf{H}(n)$ such that $(\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}') \subseteq \mathbf{H}(n)$. When \mathbf{X} is the current priority schedule and the $\mathbf{H}(n)$ selected satisfies $(\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}') \subseteq \mathbf{H}(n)$, following the scheduling algorithm, the system can make a transition to \mathbf{X}' if both $(\mathbf{X} \cap \overline{\mathbf{X}'})$ and $(\overline{\mathbf{X}} \cap \mathbf{X}')$ decide to change their scheduling decisions, and other selected elements in $\mathbf{H}(n)$ decide to keep their scheduling of previous slot. This transition probability is greater than 0, which we will give in Lemma 3. **QED** ■

Lemma 3: Suppose that $\mathbf{X} \cup \mathbf{X}' \in \mathcal{X}$. Then the transition probability from \mathbf{X} to \mathbf{X}' is:

$$p(\mathbf{X}, \mathbf{X}') = \sum_{\mathbf{H}: \mathbf{X} \Delta \mathbf{X}' \in \mathbf{H}} a(\mathbf{H}) \prod_{(i,j) \in \mathbf{X} \cap \overline{\mathbf{X}'}} \bar{p}_{ij} \prod_{(k,l) \in \overline{\mathbf{X}} \cap \mathbf{X}'} p_{kl} \prod_{(u,v) \in \mathbf{X} \cap \mathbf{X}' \cap \mathbf{H}} p_{uv} \prod_{(x,y) \in \mathbf{H} \cap \overline{\mathbf{X}} \cup \overline{\mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X} \cup \mathbf{X}')}} \bar{p}_{xy}, \quad (5)$$

where $a(\mathbf{H})$ is the probability that \mathbf{H} is selected and $\mathbf{X} \Delta \mathbf{X}' = (\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}')$.

Proof: Since $\mathbf{X} \cup \mathbf{X}' \in \mathcal{X}$, according to Lemma 2, the system can make a transition from \mathbf{X} to \mathbf{X}' . The transition occurs only when the VOQs of the selected \mathbf{H} satisfy the conditions below:

- 1) For any $(i, j) \in \mathbf{X} \cap \overline{\mathbf{X}'}$: the VOQ is selected by \mathbf{H} and decides to change its scheduling decision from 1 to 0, which happens with probability \bar{p}_{ij} .
- 2) For any $(k, l) \in \overline{\mathbf{X}} \cap \mathbf{X}'$: the VOQ is selected by \mathbf{H} and decides to change its scheduling decision from 0 to 1, which happens with probability p_{kl} .
- 3) For any $(u, v) \in \mathbf{X} \cap \mathbf{X}' \cap \mathbf{H}$: the VOQ was in the priority schedule of previous time slot, and even though selected by \mathbf{H} it decides to keep its schedule, which occurs with probability p_{uv} .
- 4) For any $(x, y) \in \mathbf{H} \cap \overline{\mathbf{X}} \cup \overline{\mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X} \cup \mathbf{X}')}$: neither the VOQ or any of its neighbors was in the priority schedule of previous time slot, and though selected by \mathbf{H} it decides to keep its schedule, which occurs with probability \bar{p}_{xy} . Since \mathbf{H} is a feasible schedule and $\overline{\mathbf{X}} \cap \mathbf{X}' \in \mathbf{H}$, $\mathbf{H} \cap \mathcal{N}(\overline{\mathbf{X}} \cap \mathbf{X}') = \emptyset$.

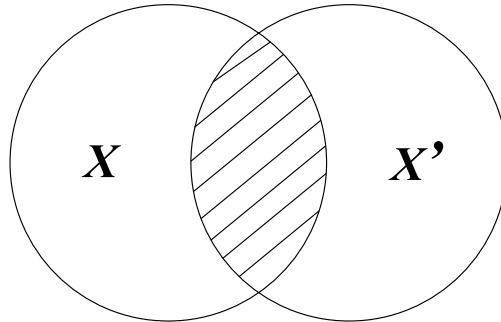


Fig. 3: States Overlap

Thus $\mathbf{H} \cap \overline{\mathbf{X} \cup \mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X})} = \mathbf{H} \cap \overline{\mathbf{X} \cup \mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X} \cup \mathbf{X}')}.$ We replace $\mathbf{H} \cap \overline{\mathbf{X} \cup \mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X})}$ by $\mathbf{H} \cap \overline{\mathbf{X} \cup \mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X} \cup \mathbf{X}')}$ in Eq. (5) for the proof of the stationary distribution in the following.

Since \mathbf{H} is a feasible schedule, elements of it can make their decisions independently. We then can multiply the probabilities of all the four categories above, which leads to the transition probability given by Eq. (5). **QED** ■

As we can see from Lemma 3, the priority schedule $\mathbf{X}(n)$ only depends on the previous time slot $\mathbf{X}(n-1)$, thus $\mathbf{X}(n-1), \mathbf{X}(n), \mathbf{X}(n+1) \dots$ is a Markov chain. The state transition probability is given in Eq. (5). So we define the Markov chain of the system on the priority schedule and will derive its stationary distribution in the following.

Lemma 4: The Markov chain of the system is positive recurrent.

Proof: Suppose that \mathbf{X} is a feasible priority schedule, and it has k non-zero elements: $(i_1, j_1), (i_2, j_2) \dots (i_k, j_k) \in \mathbf{X}$. Let \mathbf{X}_l represent a priority schedule which has l non-zero elements: $(i_1, j_1), (i_2, j_2) \dots (i_l, j_l) \in \mathbf{X}_l \subseteq \mathbf{X}, 0 \leq l \leq k$. We can see that $\mathbf{X}_0 = \mathbf{0}$ and $\mathbf{X}_k = \mathbf{X}$. Since \mathbf{X} is feasible, \mathbf{X}_l is also feasible and $\mathbf{X}_{l-1} \cup \mathbf{X}_l = \mathbf{X}_l \in \mathcal{X}$. Therefore, the system can make a transition from \mathbf{X}_{l-1} to \mathbf{X}_l with positive probability, as we already proved in Lemma 3. Hence, state \mathbf{X}_0 can reach any state $\mathbf{X} \in \mathcal{X}$ with positive probability in a finite number of steps and vice versa. Thus, the Markov chain is positive recurrent. **QED** ■

Since the Markov chain is positive recurrent, it has a unique stationary distribution. Let us associate each VOQ of a switch with a non-negative weight $w_{ij}(n)$ (i.e. $w_{ij}(n) = Q_{ij}(n)$) at time n and define the probability $p_{ij} = \frac{e^{w_{ij}(n)}}{e^{w_{ij}(n)} + 1}$. We have the following result.

Lemma 5: The Markov chain of the system has the following product-form stationary distribution:

$$\pi(\mathbf{X}) = \frac{1}{\mathcal{Z}} \prod_{(i,j) \in \mathbf{X}} \frac{p_{ij}}{\bar{p}_{ij}} = \frac{1}{\mathcal{Z}} \prod_{(i,j) \in \mathbf{X}} e^{w_{ij}(n)}, \quad (6)$$

where

$$\mathcal{Z} = \sum_{\mathbf{X} \in \mathcal{X}} \prod_{(i,j) \in \mathbf{X}} \frac{p_{ij}}{\bar{p}_{ij}} = \sum_{\mathbf{X} \in \mathcal{X}} \prod_{(i,j) \in \mathbf{X}} e^{w_{ij}(n)}. \quad (7)$$

Proof: If a state \mathbf{X} can make a transition to \mathbf{X}' , we can check that the distribution in Eq. (6) satisfies the detailed balance equation:

$$\pi(\mathbf{X})p(\mathbf{X}, \mathbf{X}') = \pi(\mathbf{X}')p(\mathbf{X}', \mathbf{X}), \quad (8)$$

hence the Markov chain is reversible and Eq. (6) is the stationary distribution (see [20], Theorem 1.2). ■

V. SYSTEM STABILITY

One of the most popular algorithm which has been proved stable is the *Maximum Weight Matching* (*MWM*) algorithm. The **MWM** algorithm selects a feasible schedule with the maximum weight:

$$\mathbf{S}^*(n) = \arg \max_{\mathbf{S} \in \mathcal{S}} \sum_{(i,j) \in \mathbf{S}} w_{ij}(n). \quad (9)$$

The stability result has been shown for bufferless crossbar switches. Following the algorithm we presented in Sec. III, if $X_{ij}(n) = 1$ and $Q_{ij}(n) > 0$, $S_{ij}^I(n) = 1$ and $S_{ij}^O(n) = 1$. A priority schedule is a feasible schedule in a bufferless crossbar switch. So we can define the weight on a priority schedule as:

$$W(\mathbf{X}) = \sum_i \sum_j X_{ij}(n) w_{ij}(n). \quad (10)$$

The **MWM** algorithm has been widely studied and it has been proved to stabilize the system. For **MWM**, the result below has been established in [21].

Lemma 6: For a scheduling algorithm, if given any ϵ and δ such that $0 \leq \epsilon, \delta < 1$, there exists a $B > 0$ such that the scheduling algorithm satisfies the condition: in any time slot t , with a probability greater than $1 - \delta$, the scheduling algorithm can choose a schedule $\mathbf{X} \in \mathcal{X}$ which satisfies the following condition:

$$\sum_{(i,j) \in \mathbf{X}(n)} w_{ij}(n) \geq (1 - \epsilon) \sum_{(k,l) \in \mathbf{X}^*(n)} w_{kl}(n), \quad (11)$$

whenever $\|\mathbf{Q}(n)\| \geq B$, where $\mathbf{Q}(n) = (Q_{ij}(n))$ and $\|\mathbf{Q}(n)\| = (\sum_{i,j} Q_{ij}^2(n))^{1/2}$. Then the scheduling algorithm can stabilize the system.

Since we already derived the stationary distribution of the Markov chain, we will prove the system stability using Lemma 6. Before the proof of Theorem 1, we first have to prove the lemmas below.

Lemma 7: Suppose that $T(\cdot)$ is a function defined on a set \mathcal{X} . For any probability distribution μ on \mathcal{X} , define the function:

$$F(\mu, T(\mathbf{X})) = E_{\mu}[T(\mathbf{X})] + H(\mu), \quad (12)$$

where $H(\mu)$ is the entropy: $H(\mu) = -\sum_{\mathbf{X} \in \mathcal{X}} \mu(\mathbf{X}) \log \mu(\mathbf{X})$. Then $F(\cdot)$ is uniquely maximized by the distribution:

$$\mu^*(\mathbf{X}) = \frac{1}{Z} \exp(T(\mathbf{X})), \quad (13)$$

where $Z = \sum_{\mathbf{X} \in \mathcal{X}} \exp(T(\mathbf{X}))$.

Proof: For any probability distribution μ , we have:

$$\begin{aligned}
F(\mu, T(\mathbf{X})) &= E_\mu[T(\mathbf{X})] + H(\mu) \\
&= \sum_{\mathbf{X} \in \mathcal{X}} \mu(\mathbf{X})T(\mathbf{X}) - \sum_{\mathbf{X} \in \mathcal{X}} \mu(\mathbf{X}) \log \mu(\mathbf{X}) \\
&= \sum_{\mathbf{X} \in \mathcal{X}} \mu(\mathbf{X})(\log \mu^*(\mathbf{X}) + \log Z) - \sum_{\mathbf{X} \in \mathcal{X}} \mu(\mathbf{X}) \log \mu(\mathbf{X}) \\
&= \sum_{\mathbf{X} \in \mathcal{X}} \mu(\mathbf{X}) \log Z + \sum_{\mathbf{X} \in \mathcal{X}} \mu(\mathbf{X}) \log \frac{\mu^*(\mathbf{X})}{\mu(\mathbf{X})} \\
&\leq \log Z \sum_{\mathbf{X} \in \mathcal{X}} \mu(\mathbf{X}) + \log \left(\sum_{\mathbf{X} \in \mathcal{X}} \mu(\mathbf{X}) \frac{\mu^*(\mathbf{X})}{\mu(\mathbf{X})} \right) \text{ (from Jensen's inequality)} \\
&= \log Z,
\end{aligned} \tag{14}$$

with equality holding only when $\mu = \mu^*$. **QED** ■

Note that when $T(\mathbf{X}) = 0$, uniform distribution maximizes $F(\mu, 0)$, and we have:

$$F(\mu, 0) = H(\mu) \leq \log Z = \log |\mathcal{X}|. \tag{15}$$

Lemma 8: Let $W(\cdot)$ be the weight function and $W^*(\mathbf{X})$ the maximum weight. Define the set:

$$\mathcal{K} = \{\mathbf{X} \in \mathcal{X} : W(\mathbf{X}) \leq (1 - \epsilon)W^*(\mathbf{X})\}. \tag{16}$$

Then, we have:

$$\pi(\mathcal{K}) \leq \frac{\log |\mathcal{X}|}{\epsilon W^*(\mathbf{X})} \tag{17}$$

Proof: As shown in Eq. (6), for a schedule $\mathbf{X} \in \mathcal{X}$, its stationary distribution is: $\pi(\mathbf{X}) = \frac{1}{Z} \prod_{(i,j) \in \mathbf{X}} e^{(w_{ij}(n))} = \frac{1}{Z} e^{W(\mathbf{X})}$. According to Lemma 7, π maximizes $F(\mu, W(\mathbf{X}))$.

Let \mathbf{X}^* be the schedule which can give the maximum weight. Let π' be the distribution that assigns all probability on \mathbf{X}^* such that:

$$\pi'(\mathbf{X}) = \begin{cases} 1 & \text{if } \mathbf{X} = \mathbf{X}^* \\ 0 & \text{otherwise} \end{cases}$$

Then we have:

$$\begin{aligned}
F(\pi', W(\mathbf{X})) &= E_{\pi'}[W(\mathbf{X})] + H(\pi') \\
&= W^*(\mathbf{X}) + H(\pi') \\
&\leq F(\pi, W(\mathbf{X})) = E_\pi[W(\mathbf{X})] + H(\pi)
\end{aligned}$$

$$\leq W^*(\mathbf{X})(1 - \epsilon\pi(\mathcal{K})) + H(\pi) \quad (18)$$

So,

$$\begin{aligned} W^*(\mathbf{X}) + H(\pi') &\leq W^*(\mathbf{X})(1 - \epsilon\pi(\mathcal{K})) + H(\pi) \\ \epsilon\pi(\mathcal{K})W^*(\mathbf{X}) &\leq H(\pi) - H(\pi') \leq H(\pi) \leq \log |\mathcal{X}| \\ \pi(\mathcal{K}) &\leq \frac{\log |\mathcal{X}|}{\epsilon W^*(\mathbf{X})} \end{aligned} \quad (19)$$

■

Theorem 1: The distributed scheduling algorithm can stabilize the system if the input traffic is admissible.

Proof: For any $\delta > 0$, we have $\pi(\mathcal{K}) < \delta$, if the maximum weight satisfies the condition:

$$W^*(\mathbf{X}) > \frac{N^2 \log 2}{\epsilon \delta} > \frac{\log |\mathcal{X}|}{\epsilon \delta}. \quad (20)$$

So, for any $\epsilon, \delta > 0$, there exists a $B > 0$ such that whenever $\|\mathbf{Q}(n)\| > B$, Eq. (20) holds and then $\pi(\mathcal{K}) < \delta$. Hence the scheduling algorithm can stabilize the system according to Lemma 6. ■

VI. SIMULATIONS

In this section, we run simulations for different scenarios to determine the performance of our algorithm. We also study the delay performance of the scheduling algorithm under different traffic patterns, including uniform and non-uniform traffic with Bernoulli and bursty arrivals. For bursty traffic, the burst length is exponentially distributed over $[1, 1000]$, following the truncated Pareto distribution:

$$P(l) = \frac{c}{l^\alpha}, \quad l = 1, 2, \dots, 1000, \quad (21)$$

where l is the burst length, α is the Pareto parameter and c is the normalization constant. In the simulations, $\alpha = 1.7$. The average burst length then is about 11.6. All inputs are equally loaded and we measure the packet delay.

A. Uniform Traffic

For uniform traffic, a new cell is destined uniformly for all output ports. Let λ represent the traffic load, the arrival rate between input i and output j is $\lambda_{ij} = \frac{\lambda}{N}$. The delay performance of our distributed

scheduling (DS) algorithms under uniform Bernoulli and bursty traffic is shown in Fig. 4. We can see that the packet delay of our algorithm is very close to the output-queued switch (OQ). Actually, it has been shown that under uniform traffic, even an algorithm as simple as dual round-robin can have a delay performance close to an output-queued switch [8]. However, the dual round-robin scheduling algorithm cannot achieve 100% throughput when the traffic is non-uniform. Therefore, we will study the performance of our algorithm under non-uniform traffic next.

B. Non-uniform Traffic

We run the simulations for the following traffic patterns:

- Lin-diagonal: Arrival rates at the same input differ linearly, i.e, $\lambda_{i(i+j \pmod{N})} - \lambda_{i(i+j+1 \pmod{N})} = 2\lambda/N(N+1)$.
- Hot-spot: For input port i , $\lambda_{ii} = \omega\lambda$ and $\lambda_{ij} = (1-\omega)\lambda/(N-1)$, for $i \neq j$. We can get different traffic patterns by varying the hot-spot factor ω .

The delay performance for lin-diagonal and hot-spot traffic are shown in Fig. 5 and 6, respectively. We can see that under Bernoulli traffic, the delay performance of our algorithm is still very close to the output-queued switch. Packets have low delay even when the load is as high as 0.99. Note that the RR-RR scheduling algorithm can have a throughput of only around 85% under hotspot traffic and the SBF-LBF [22] has a throughput of around 87%. When the traffic arrival is bursty, our algorithm has a delay a little larger than the output-queued switch. But the difference is not large

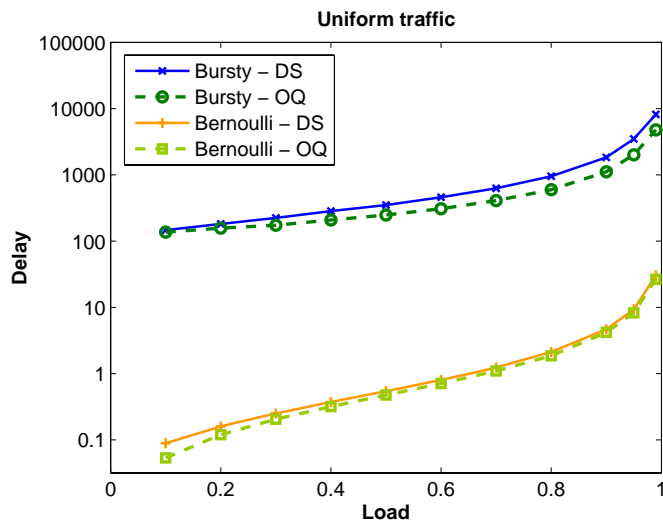


Fig. 4: Switch size $N=32$, uniform traffic

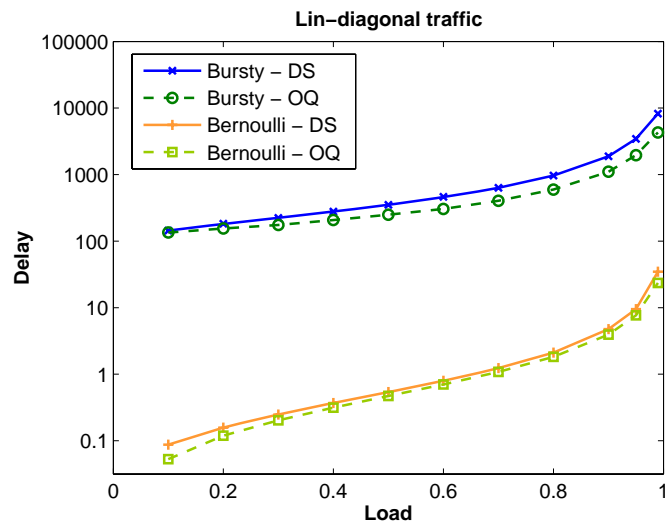


Fig. 5: Switch size $N=32$, lin-diagonal traffic

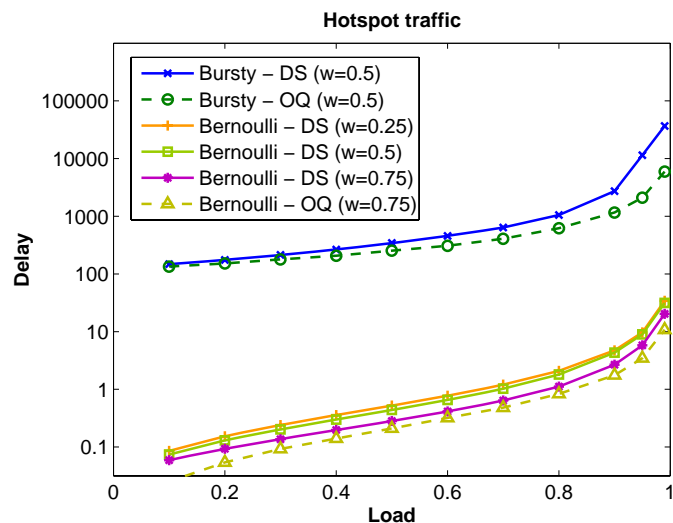


Fig. 6: Switch size $N=32$, hot-spot traffic

C. Impact of Switch Size

In this section, we will study the impact of switch size on the delay performance. Generally, for input-queued switch, the average delay increases linearly with the switch size. For output-queued switches, delay is independent of the size. Fig. 7 shows the delay performance of switches with different sizes under hot-spot traffic, for which ω is 0.5. We can see that, for Bernoulli traffic, the delay is almost the same for different switch sizes. As the size increases, the delay even decreases slightly. This shows that our algorithm is feasible for large scale packet switches, while providing good delay performance.

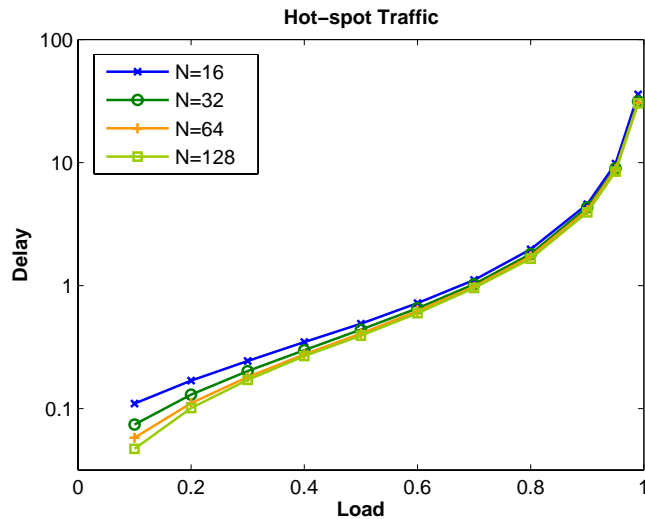


Fig. 7: Impact of switch size, hot-spot traffic, $w=0.5$

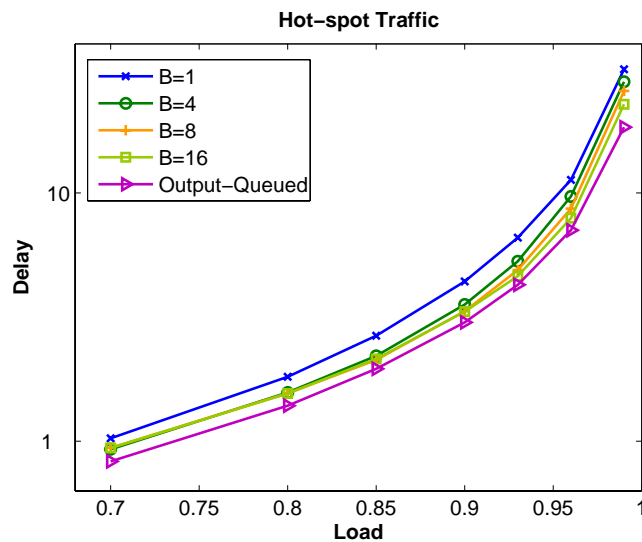


Fig. 8: Impact of buffer size, hot-spot traffic, $w=0.5$

D. Impact of Buffer Size

If the buffer at each crosspoint increases to infinite, then the buffered crossbar switch is then equivalent to an output-queued switch. So as we increase the buffer size, the average delay will decrease and slowly converge to the delay of an output-queued switch. As we already showed in previous simulation results, the delay performance of the new algorithm with buffer size 1 is already very close to that of an output-queued switch. Therefore, by increasing the buffer size, we can get a very marginal improvement on the delay performance. Fig. 8 shows the delay performance of our new algorithm with different buffer sizes, under hot-spot traffic. We can see that exactly as we analyzed, the delay is decreasing with the increase of the buffer size and slowly converge to the output-queued switch. So using the new algorithm, we only

need to implement a one-packet buffer at each crosspoint and still provide good delay performance.

VII. CONCLUSION

In this paper, we propose a distributed scheduling algorithms for buffered crossbar switch. We prove that it can achieve 100% throughput under any admissible arrival traffic with only a one-packet buffer at each crosspoint. Our simulation results show that it can provide very good delay performance under different traffic arrivals, which makes it practical to implement in real packet switching systems. Our simulation results also show that, by using our new algorithm, packet delay is not dependent on the switch size, which means that the distributed algorithm can scale with the number of ports of a switching system.

REFERENCES

- [1] J. G. Dai and B. Prabhakar, "The Throughput of Data Switches with and without Speedup," in *Proc. of IEEE INFOCOM*, (Tel Aviv, Israel), March 2000.
- [2] E. Lonardi, M. Mellia, F. Neri, and M. A. Marsan, "On The Stability of Input-Queued Switches with Speed-up," *IEEE/ACM Transactions on Networking*, vol. 9, February 2001.
- [3] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queuing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," *IEEE Transactions on Automatic Control*, vol. 37, pp. 1936–1949, December 1992.
- [4] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *IEEE Transactions on Communications*, vol. 47, pp. 1260–1267, August 1999.
- [5] T. E. Anderson, S. S. Owichi, J. B. Saxe, and C. P. Thacher, "High Speed Switch Scheduling for Local Area Networks," *ACM Transactions on Computer Systems*, vol. 11, pp. 319–352, November 1993.
- [6] N. Mckeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 188–201, April 1999.
- [7] Y. Li, S. Panwar, and H. J. Chao, "On the Performance of a Dual Round-Robin Switch," in *Proc. of IEEE INFOCOM*, April 2001.
- [8] R. Rojas-Cessa, E. Oki, and H. J. Chao, "On the Combined Input-Crosspoint Buffered Packet Switch with Round-Robin Arbitration," *IEEE Transactions on Communications*, vol. 53, pp. 1945–1951, November 2005.
- [9] T. Javidi, R. Magill, and T. Hrabik, "A High Throughput Scheduling Algorithm for a Buffered Crossbar Switch Fabric," in *Proc. of IEEE ICC*, (Helsinki, Finland), June 2001.
- [10] S.-T. Chuang, S. Iyer, and N. McKeown, "Practical Algorithms for Performance Guarantees in Buffered Crossbars," in *Proc. of IEEE INFOCOM*, (Miami, Florida), March 2005.
- [11] J. Turner, "Strong Performance Guarantees for Asynchronous Crossbar Schedulers," in *Proc. of IEEE INFOCOM*, (Spain), April 2006.
- [12] M. Berger, "Delivering 100% Throughput in a Buffered Crossbar with Round Robin Scheduling," in *Proc. of IEEE HPSR*, (Poznan, Poland), 2006.
- [13] Y. Shen, S. S. Panwar, and H. J. Chao, "Providing 100% Throughput in a Buffered Crossbar Switch," in *Proc. of IEEE HPSR*, (Brooklyn, New York), May-June 2007.
- [14] S. Rajagopalan and D. Shah, "Distributed Algorithm and Reversible Networks," in *Proc. of CISS*, March 2008.

- [15] L. Jiang and J. Walrand, "A Distributed Algorithm for Optimal Throughput and Fairness in Wireless Networks with a General Interference Model," *submitted*, June 2008.
- [16] S. Rajagopalan and D. Shah, "Aloha That Works," *submitted*, Nov. 2008.
- [17] J. Ni and R. Srikant, "Distributed CSMA/CA Algorithms for Achieving Maximum Throughput in Wireless Networks," *submitted*, March 2009.
- [18] J. Ni and R. Srikant, "Q-CSMA: Queue-Length Based CSMA/CA Algorithms for Achieving Maximum Throughput and Low Delay in Wireless Networks," *submitted*, August 2009.
- [19] P. Giaccone, B. Prabhakar, and D. Shah, "Toward Simple, High Performance Schedulers for High-Aggregate Bandwidth Switches," in *Proc. of IEEE INFOCOM*, (New York), 2002.
- [20] F. Kelly, *Reversibility and Stochastic Networks*. Wiley, 1979.
- [21] A. Eryilmaz, R. Srikant, and J. R. Perkins, "Stable Scheduling Policies for Fading Wireless Channels," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 411–424, April 2005.
- [22] L. Mhamdi and M. Hamdi, "MCBF: a High-Performance Scheduling Algorithm for Buffered Crossbar Switches," *IEEE Communications Letters*, vol. 7, pp. 451–453, September 2003.