Localized Independent Packet Scheduling for Buffered Crossbar Switches

Deng Pan, *Member*, *IEEE*, and Yuanyuan Yang, *Fellow*, *IEEE*

Abstract—Buffered crossbar switches are a special type of crossbar switches. In such a switch, besides normal input queues and output queues, a small buffer is associated with each crosspoint. Due to the introduction of crosspoint buffers, output and input contention is eliminated, and the scheduling process for buffered crossbar switches is greatly simplified. Moreover, since different input ports and output ports work independently, the switch can easily schedule and transmit variable length packets. Compared with fixed length packet scheduling, variable length packet scheduling has some unique advantages: higher throughput, shorter packet latency, and lower hardware cost. In this paper, we present a fast and practical scheduling scheme for buffered crossbar switches called Localized Independent Packet Scheduling (LIPS). With LIPS, an input port or output port makes scheduling decisions solely based on the state information of its local crosspoint buffers, i.e., the crosspoint buffers where the input port sends packets to or the output port retrieves packets from. The localization feature makes LIPS suitable for a distributed implementation and thus highly scalable. Since no comparison operation is required in LIPS, scheduling arbiters can be efficiently implemented using priority encoders, which can make arbitration decisions quickly in hardware. Another advantage of LIPS is that each crosspoint needs only L (the maximum packet length) buffer space, which minimizes the hardware cost of the switches. We theoretically analyze the performance of LIPS and, in particular, prove that LIPS achieves 100 percent throughput for any admissible traffic with speedup of two. We also discuss in detail the implementation architecture of LIPS and analyze the packet transmission timing in different scenarios. Finally, simulations are conducted to verify the analytical results and measure the performance of LIPS.

Index Terms—Buffered crossbar switches, packet scheduling, 100 percent throughput, priority encoders.

1 Introduction

Crossbar switches provide nonblocking capability and overcome the bandwidth limitation of bus-based switches. They have long been the preferred structures for high-speed switches and routers. With the ever-increasing demand for more bandwidth and higher throughput, it has become a more and more challenging task to design high-performance crossbar switches and efficient scheduling algorithms.

For crossbar switches, packets can be buffered at either output ports, input ports, or crosspoints of the crossbar. Output queued (OQ) switches only have buffer space at the output side, as shown in Fig. 1a, and new incoming packets must be immediately transferred through the crossbar and stored in the output queues. OQ switches achieve 100 percent throughput and can provide different levels of performance guarantees by running various fair scheduling algorithms, such as WFQ [1], DRR [2], and FMCF [3], at each output port. However, in order for an $N \times N$ switch to achieve 100 percent throughput, speedup of N is required. Consider that each input port has a packet arrived at the same time and all the N packets are destined for the same output port. Because there is no buffer space at the input side, all the N packets have to be simultaneously transmitted by the crossbar to the output port. This means that the crossbar must have N times bandwidth as that of an input port or output port. Thus, OQ switches are difficult to scale. On the contrary, as illustrated in Fig. 1b, input queued (IQ) switches only have buffer space at the input side and need no speedup. In order to make fast scheduling decisions, iterative maximal matching algorithms, such as PIM [4], iSLIP [5], and DRRM [6], were proposed for IQ switches, which can quickly converge on a maximal matching in multiple iterations. IQ switches are popular on the market due to their economical hardware architectures and efficient scheduling algorithms. Unfortunately, until now IQ switches are found to be able to achieve 100 percent throughput only when working with maximum matching algorithms or their variants, such as MWM [7], which have high time complexity [34], [35] and are not feasible for high-speed scheduling. In order to combine the advantages of both OQ switches and IQ switches, combined input-output queued (CIOQ) switches make a tradeoff between the crossbar speedup and the complexity of the scheduling algorithms. They usually have fixed small speedup of two and, thus, need buffer space at both the input side and output side, as shown in Fig. 1c. It has been shown that CIOQ switches with speedup of two can achieve 100 percent throughput with any maximal scheduling algorithms [8], [9] and can also emulate OQ switches with more complex algorithms [15], [16].

For buffering at the input side of a switch, usually the virtual OQ (VOQ) buffering technique [7] is used, since the

- *D. Pan is with the Department of Electrical and Computer Engineering, 1055 W. Flagler Street, EC 2945, Florida International University, Miami, FL 33174. E-mail: pand@fiu.edu.*
- *Y. Yang is with the Department of Electrical and Computer Engineering, State University of New York, Stony Brook, NY 11794. E-mail: yang@ece.sunysb.edu.*

Manuscript received 25 July 2007; revised 16 July 2008; accepted 28 July 2008; published online 6 Aug. 2008.
Recommended for acceptance by S. Dolev.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-07-0382.
Digital Object Identifier no. 10.1109/TC.2008.140.

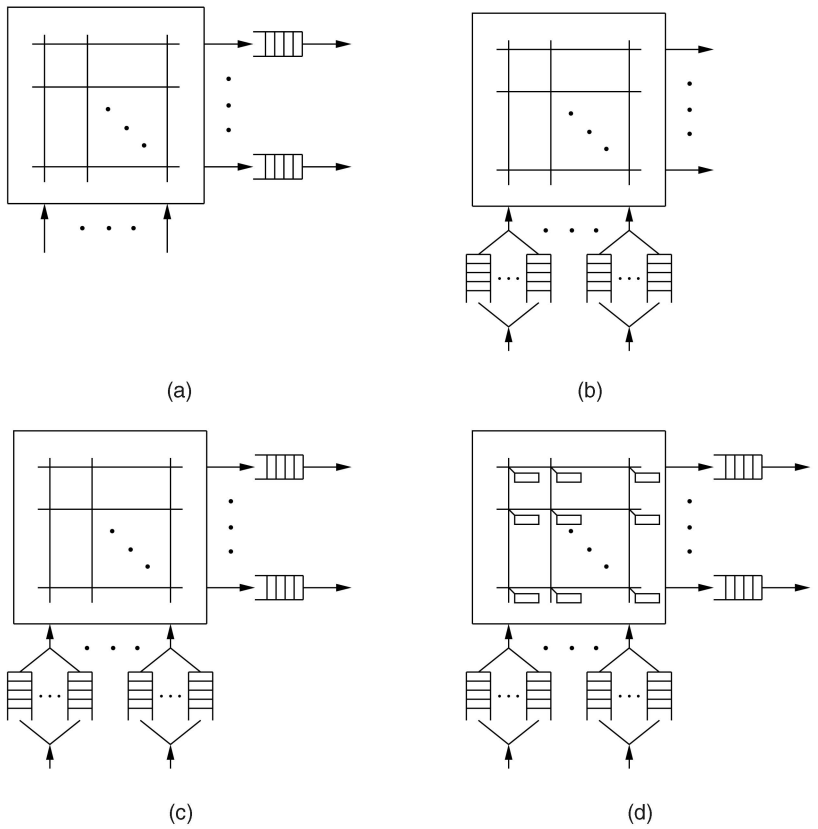

Fig. 1. Structures of different crossbar switches. (a) OQ switches. (b) IQ switches. (c) CIOQ switches. (d) Buffered crossbar switches.

traditional single FIFO queue suffers from the head of line (HOL) blocking, i.e., even though the destination output ports of the packets behind the HOL packet may be free, these packets cannot be scheduled to transmit because the HOL packet is blocked. It was proved in [10] that the HOL blocking limits the maximum throughput of the switch to about 58.6 percent. On the contrary, VOQ buffering maintains a logically separate queue or virtual queue for each output port at every input port, so that a packet will no longer be held up by another packet ahead of it that goes to a different output port.

With the development of modern VLSI technology, it has been feasible to integrate on-chip memory to the crossbar switching fabric. Buffered crossbar switches, or combined input-crosspoint-output queued (CICOQ) switches [17], [18], [28], are a special type of CIOQ switches, where each crosspoint of the crossbar is equipped with a small buffer, as illustrated in Fig. 1d. Due to the introduction of crosspoint buffers, the scheduling process is greatly simplified. First, output contention is eliminated. Assume that multiple input ports have packets destined for the same output port. In a traditional unbuffered crossbar switch, since packets are directly sent from input ports to output ports, the transmission has to be carefully scheduled so that no two input ports will simultaneously send packets to the same output port. On the other hand, in a buffered crossbar switch, these packets can be first sent to the crosspoint buffers, and then the output port can retrieve the packets from the crosspoint buffers one by one. Furthermore, since output contention has been eliminated, different input ports no longer need to cooperate with each other, and their scheduling can be conducted independently. As a result, the complexity of the scheduling algorithm is greatly reduced.

Previous research on scheduling algorithms for crossbar switches mainly focused on fixed length packet scheduling (or cell scheduling for short) [11]. Without crosspoint buffers, packets have to be directly sent from input ports to output ports. In order to maximize throughput and make fast scheduling decisions, all the scheduling and transmission units must have the same length, and all the input ports and output ports have to work in a synchronized mode, i.e., all input ports send cells at the same time, and all output ports receive cells at the same time. When variable length packets arrive, they must be segmented into fixed length cells at input ports. The cells are then used as the scheduling units and transmitted to output ports, where they are reassembled into original packets and sent to the output lines. In contrast, buffered crossbar switches have removed the necessity of synchronization due to crosspoint buffers. They can work in an asynchronous mode and directly handle variable length packets. In other words, each input port or output port periodically chooses a packet of an arbitrary length to send to or receive from the crosspoint buffer and does not need to wait for other input ports or output ports.

Compared with cell scheduling, variable length packet scheduling (or packet scheduling for short) has some unique advantages. First, packet scheduling can better utilize available bandwidth and achieve higher throughput. For

cell scheduling, when a packet is segmented into cells, its length may not be a multiple of the cell length, and the last segment has to be padded with empty bits to reach the cell length. The padding bits do not contain useful information and waste transmission capacity of the switch. In the worst case, if all packets happen to have a slightly longer length than the cell length, each packet has to be segmented into two cells, and the switch can only achieve about half of the maximum throughput. Second, since there is no segmentation and reassembly in packet scheduling, newly arrived packets at input ports can be immediately transferred through the crossbar, and similarly, transmitted packets at the output ports can be immediately sent to the output lines. Especially when the load is light, packets have short queuing delay, and the time for segmentation and reassembly can hardly be overlapped with other waiting time. Therefore, packet scheduling reduces the latency that a packet experiences in the switch. Third, no extra buffer space is needed at the input or output side to segment and reassemble the packets, which lowers hardware cost. Finally, cell scheduling can be regarded as a special case of packet scheduling, or in other words, packet scheduling can also handle fixed length cells.

Two packet scheduling algorithms for asynchronous buffered crossbar switches, Packet GVOQ (PGV) and Packet LOOFA (PLF), were proposed in [12], and their performance guarantees were analyzed. It was proved that, with speedup of two and $2L$ or more buffer space at each crosspoint, where L is the maximum packet length, PGV and PLF can provide work-conserving guarantees or emulate push-in-first-out (PIFO) scheduling algorithms for OQ switches. In order to be work conserving, the algorithms proposed in [12] usually impose an order on buffered packets and make scheduling decisions by sorting the packets. With slightly more buffer space at each crosspoint, they can emulate any PIFO fair scheduling algorithm for OQ switches by ordering the packets based on the departure sequence of the packets in the reference algorithm and, thus, provide bandwidth and delay guarantees. Scheduling algorithms delivering strong performance guarantees are certainly important, especially considering that nowadays many broadband-based multimedia applications have quality of service (QoS) requirements. On the other hand, it is also worth studying scheduling algorithms that are simple and easy to implement.

The objective of this paper is to design packet scheduling algorithms for buffered crossbar switches with low time complexity and easy hardware implementation. We present a packet scheduling scheme called Localized Independent Packet Scheduling (LIPS). LIPS conducts scheduling in an independent and distributed mode. An input port or output port makes scheduling decisions solely based on the state information of its local crosspoint buffers, i.e., the crosspoint buffers where the input port sends packets to or the output port retrieves packets from. Since no comparison operation is required, scheduling arbiters can be efficiently implemented using priority encoders, which can make decisions quickly in hardware. LIPS requires only L buffer space at each crosspoint. Considering that on-chip buffers are expensive resources, LIPS minimizes the hardware cost of switches. We theoretically analyze the performance of LIPS and, in particular, prove that LIPS achieves 100 percent throughput for any admissible traffic with speedup of two. We also present the detailed implementation architecture of LIPS and analyze the packet transmission timing in different scenarios. Finally, simulations are conducted to verify the analytical results and to measure the performance of LIPS.

The rest of this paper is organized as follows: In Section 2, we provide an overview of the scheduling algorithms proposed in the literature for buffered crossbar switches. In Section 3, we present the LIPS scheduling scheme and analyze its performance. In Section 4, we discuss the hardware architecture of LIPS and other implementation issues. In Section 5, we present the simulation results to verify the analytical results obtained in Section 3 and test the performance of LIPS. In Section 6, we conclude this paper.

2 Related Work

Scheduling algorithms for buffered crossbar switches are generally designed with two possible objectives: to achieve high throughput or to emulate scheduling algorithms for OQ switches. The latter is a stronger requirement than the former, i.e., an algorithm that emulates an OQ switch-based algorithm usually delivers 100 percent throughput, but the reverse is not always true. On the other hand, if 100 percent throughput is the only objective, an algorithm can be simpler or have less time complexity.

A buffered crossbar switch architecture called CIXB-1 was proposed in [18], where each crosspoint has a one-cell buffer. CIXB-1 offers several advantages for feasible implementation such as scalability and timing relaxation. It was shown that, in conjunction with round-robin (RR) arbitration, CIXB-1 can provide 100 percent throughput under uniform traffic. CIXOB-k [19] is the extended version of CIXB-1 with a k-cell buffer at each crosspoint and small speedup for the crossbar. CIXOB-k was shown to be able to achieve 100 percent throughput under uniform traffic as well as nonuniform traffic. Kornaros [20] presented a buffered crossbar switch architecture called BCB to reduce storage requirements and offer better memory utilization. The architecture can provide superior performance in situations of bursty traffic and in variable size packet-based environments. A cell scheduling scheme for buffered crossbar switches called Most Critical Buffer First (MCBF) was proposed in [21]. It conducts scheduling based on the crosspoint buffer information and has low hardware complexity. MCBF exhibits good performance and shows optimal stability in simulations. Shortest Crosspoint Buffer First (SCBF) [22] is another cell scheduling scheme, which finds a matching with minimum weight in each time slot. It was proved that SCBF achieves 100 percent throughput for any admissible traffic without speedup requirement. In order to facilitate hardware implementation, a maximal solution of SCBF was also proposed in [22], which achieves low $O(\log N)$ time complexity and is shown to have almost identical performance. A rate-based flow control scheme called sMUX was proposed in [23]. It can utilize 100 percent of the switch capacity to provide deterministic guarantees of bandwidth and fairness, delay, and jitter bounds for each flow. In [24], it was proved that a buffered crossbar switch

with a large number of ports asymptotically achieves 100 percent throughput for uniform Bernoulli i.i.d. traffic. The algorithms discussed above are cell scheduling algorithms targeting high throughput.

The emulation of OQ switches by buffered crossb0.ar switches was studied in [27]. It was proved that a buffered crossbar switch with speedup of two satisfying nonnegative slackness (NNS) insertion and lowest time to live (LTTL) blocking, and LTTL fabric scheduling can exactly emulate an OQ switch with PIFO scheduling policies. In particular, it was shown that the GBVOQ_OCF scheduling algorithm can exactly emulate an FIFO OQ switch, and the GBFG_SP scheduling algorithm can exactly emulate a strict priority OQ switch. In [25], the MCAF-LTF cell scheduling scheme for one-cell buffered crossbar switches was proposed. MCAF-LTF does not require costly time stamping mechanism and is able to emulate an OQ switch with speedup of two. Chuang et al. [26] studied practical scheduling algorithms for buffered crossbar switches. It was shown that with speedup of two, a buffered crossbar switch can mimic the restricted PIFO-OQ switch (a PIFO-OQ switch with the restriction that the cells of an input-output pair depart the switch in the same order as they arrive), regardless of the incoming traffic pattern, and that with speedup of three, a buffered crossbar switch can mimic an arbitrary PIFO-OQ switch and hence provide delay guarantees. It was also shown that buffered crossbar switches can achieve 100 percent throughput with speedup of two for any Bernoulli i.i.d. admissible traffic. The above algorithms also consider cell scheduling but were designed to emulate scheduling algorithms for 'OQ switches.

A buffered crossbar switch architecture supporting packet scheduling was proposed in [28]. The chip layout was presented and the hardware cost was analyzed. The simulation results demonstrate that the proposed architecture outperforms unbuffered crossbar switches. A segmentation-and-reassembly (SAR) scheme was proposed in [29]. It uses variable size segments while merging multiple packets into each segment. The proposed scheme eliminates padding overhead, reduces header overhead and crosspoint buffer size, and is suitable for use with external, modern DRAM buffer memory in ingress line cards. The simulation results show that it outperforms existing segmentation schemes in buffered as well as unbuffered crossbar switches. The performance guarantees of packet scheduling for asynchronous buffer crossbar switches were discussed in [12], and two algorithms were designed based on existing cell scheduling algorithms. It was theoretically proved that, with speedup of two, the PGV scheduling algorithm provides work-conserving guarantees with $2L$ crosspoint buffer space and can emulate a PIFO scheduling algorithm for OQ switches with $5L$ crosspoint buffer space. The PLF scheduling algorithm provides work-conserving guarantees with $16L/3$ crosspoint buffer space and can emulate a PIFO scheduling algorithm for OQ switches with $22L/3$ crosspoint buffer space. Stephens and Zhang [17] proposed the Distributed Packet Fair Queueing (DPFQ) architecture for physically dispersed line cards to emulate an OQ switch with fair queuing, and the simulation results demonstrate that the resulting system provides service that closely approximates an output buffered switch employing fair queuing with modest speedup. The above schemes use variable length packets as the scheduling and switching units.

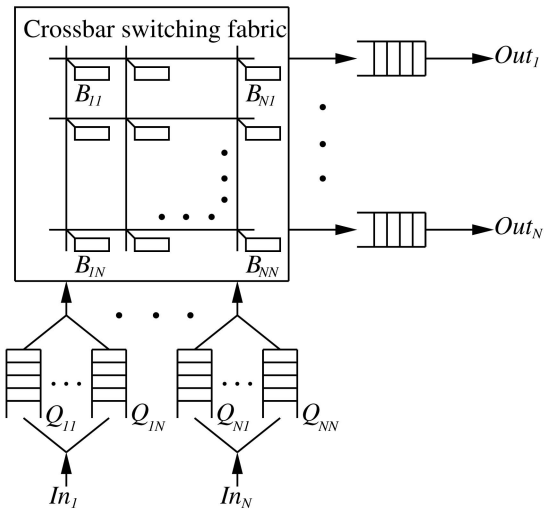

Fig. 2. The structure of buffered crossbar switches.

3 Localized Independent Packet Scheduling

In this section, we present our new LIPS scheme and analyze its performance.

3.1 Switch Model

The switch model considered in this paper is illustrated in Fig. 2, where N input ports and N output ports are connected by a crossbar switching fabric, which has speedup of two. We denote the bandwidth of an input port or output port by R, and the crossbar has bandwidth $2R$. An input port has N virtual queues to store packets destined for different output ports. Each crosspoint has an exclusive buffer of size L. Depending on the granularity level of performance guarantees, an output port may have a single queue, or multiple logical queues. For example, if bandwidth is to be fairly allocated among input ports, packets in output ports should be buffered on a per input port basis, and if different flows require different packet delay guarantees, an output port needs to set up as many queues as the number of flows. Different input ports and output ports work independently and asynchronously. After a packet arrives at the switch, it is first stored in the input queue. The packet is sent from the input queue to the crosspoint buffer, and then from the crosspoint buffer to the output queue and finally delivered to the output line.

Based on the locations of the packets to be scheduled, there are three types of scheduling involved in such a buffered crossbar switch, which we call input scheduling, crossbar scheduling and output scheduling, respectively. In input scheduling, an input port selects one of the virtual queues and sends its head packet to the crosspoint buffer. In crossbar scheduling, an output port selects one of the crosspoint buffers and retrieves the buffered packet to the output queue. In output scheduling, an output port selects a buffered packet from the output queue and sends the packet to the output line.

Over the last decade, output scheduling has been well studied, and a lot of output scheduling algorithms have been proposed, such as WFQ [1] and DRR [2]. By using different approaches, the algorithms provide different performance guarantees and have different time complexity. It was reported in [13] that a fundamental tradeoff exists between the performance guarantees that an algorithm can provide and its time complexity. It should be noted that output scheduling algorithms usually do not affect the throughput performance as long as they are work conserving. In other words, if an output scheduling algorithm sends packets to the output line whenever there is a packet in the output queue, 100 percent throughput can be achieved given that the input scheduling algorithm and crossbar scheduling algorithm deliver packets to the output queue in time. Therefore, in the rest of this paper, we will mainly consider input scheduling and crossbar scheduling and adopt a simple FIFO algorithm for output scheduling, which is work conserving.

TABLE 1
LIPS

```
Input Scheduling:
each input port independently does {
    while true do {
        select a backlogged virtual queue whose crosspoint buffer is empty;
        transfer the virtual queue head packet to the crosspoint buffer;
        if the channel to the output port is idle {
            transfer the packet to the output queue;
        }
    }
}

Crossbar Scheduling:
each output port independently does {
    while true do {
        select a crosspoint buffered packet;
        transfer the packet to the output queue;
        if the channel to the output line is idle {
            transfer the packet to the output line;
        }
    }
}
```

3.2 Algorithm Description

Input scheduling and crossbar scheduling of LIPS are conducted in an independent and distributed manner, and they only rely on the state information of local crosspoint buffers. Local crosspoint buffers of an input port or output port are the crosspoint buffers that the input port sends packets to or the output port receives packets from.

In input scheduling, when the transmission channel of an input port to the crosspoint buffers is idle, the input port selects one of its backlogged virtual queues whose corresponding crosspoint buffer is empty and sends the head packet to the crosspoint buffer. When there are multiple eligible virtual queues, different arbitration rules can be used, such as fixed priority (FP), random priority, or RR. In particular, the RR rule is able to avoid starvation by alternatively giving each virtual queue the highest priority. We will see later in this section that any work-conserving rule is able to achieve 100 percent throughput. It should be noted that since the crossbar has speedup of two, the packet is transferred from the virtual queue to crosspoint buffer with bandwidth of $2R$. After the last bit of the packet is sent to the crosspoint buffer, the scheduling and transmission process is repeated again. Crossbar scheduling is similar to input scheduling. When the transmission channel of an output port for receiving packets from the crosspoint buffers is idle, the output port selects a crosspoint buffered packet and saves it in its output queue. The transmission rate from the crosspoint buffer to the output queue is also $2R$, and different arbitration rules can be used as well.

In order to reduce packet latency and increase switch throughput, we use cut-through switching in the crossbar. In other words, when a packet is being sent from the virtual queue to the crosspoint buffer, if the transmission channel to its output is idle, the packet can be directly sent to the output queue without waiting for the whole packet to be buffered at the crosspoint buffer. Similarly, if the output port has cut-through switching capability, a packet can also be immediately sent to the output line as soon as its first bit arrives at the output queue. Thus, with the cut-through technique, it is possible that a packet is directly sent from the input queue to the output line without being fully buffered anywhere on the way. Since the bandwidth from the input queue to the crosspoint buffer and from the crosspoint buffer to the output queue is $2R$, and the bandwidth from the output queue to the output line is R, a packet can be safely delivered to the output line without being blocked in the middle. However, it should be noted that, because the bandwidth from the input line to the virtual queues is R, cut-throughput switching cannot be used at the input port. In other words, only after all the bits of a new incoming packet have been saved in the virtual queue, the packet can begin to be sent to the crosspoint buffer.

For easy understanding, the pseudo code description for the input scheduling and crossbar scheduling of LIPS is presented in Table 1. Note that, in input scheduling, the scheduling candidates of an input port are only the virtual queues whose crosspoint buffers are empty. This restriction seems to be unnecessary, because a crosspoint buffer may be able to contain more than one packets of shorter length. However, instead of calculating the remaining free space of the buffer, testing only whether it is empty or occupied greatly simplifies the operation. In this way, only one bit of information needs to be tested for each crosspoint buffer during input scheduling, and the testing of all the crosspoint buffers of the same input port can be conducted in parallel to minimize the time cost. Similarly, in crossbar scheduling, an output port only needs to test whether a crosspoint buffer is occupied or empty due to the cut-through switching capability of the crossbar.

3.3 Performance Analysis

In this section, we analyze the performance of LIPS. First, we use the fluid theory in [8] to prove that LIPS is able to achieve 100 percent throughput for any admissible traffic with speedup of two, regardless of the arbitration rules used by input scheduling and crossbar scheduling.

Before starting the analysis, we define some notations and variables. Let In_i denote the ith input port, and Out_j denote the jth output port. Q_{ij} represents the virtual queue of In_i to buffer the packets destined for Out_j, and B_{ij} represents the crosspoint buffer connecting In_i and Out_j. The following variables are used to represent the status of

the switch. Their initial values (at time 0) are conventionally assumed to be zero:

- $Q_{ij}(t)$: the number of bits buffered in Q_{ij} at time t;
- $B_{ij}(t)$: the number of bits buffered in B_{ij} at time t;
- $A_{ij}(t)$: the number of bits arrived at Q_{ij} up to time t;
- $D_{ij}(t)$: the number of bits left from Q_{ij} up to time t;
- $E_{ij}(t)$: the number of bits left from B_{ij} up to time t.

$A_{ij}(t)$ satisfies a strong law of large numbers (SLLN), i.e.,

$$\lim_{t\to\infty}\frac{A_{ij}(t)}{t}=\lambda_{ij}, \tag{1}$$

where λ_{ij} is called the arrival rate of Q_{ij}.

A traffic is said to be admissible if it has no oversubscription at any input port or output port, i.e.,

$$\forall i, \sum_k \lambda_{ik} < R, \tag{2}$$

$$\forall j, \sum_k \lambda_{kj} < R. \tag{3}$$

Following the definition in [8], we say that a scheduling scheme achieves 100 percent throughput if the following equation holds for any admissible traffic:

$$\forall i, \forall j, \lim_{t\to\infty}\frac{E_{ij}(t)}{t}=\lambda_{ij}. \tag{4}$$

$\lim_{t\to\infty}\frac{\sum_j E_{ij}(t)}{t}$ is the average rate that packets are transmitted to the jth output port. $\lim_{t\to\infty}\frac{\sum_j E_{ij}(t)}{t}=\sum_j \lambda_{ij}$ means that all traffic to the jth output port is delivered to the output queue. Thus, 100 percent throughput can be achieved, as long as the output scheduling algorithm is work conserving. Intuitively, traffic arrives at Q_{ij} and leaves from B_{ij} with the same speed, and thus packets will not infinitely accumulate at either Q_{ij} or B_{ij}.

A packet is saved to the virtual queue only after it has been fully received by the input port, and thus the value of $A_{ij}(t)$ changes only at some specific time points $\{t_0, t_1, \ldots, t_n, \ldots\}$, where $t_0 = 0$ and $t_n (n > 0)$ is the time that the nth packet is saved into the virtual queue. For $t_n < t < t_{n+1}$, we have $A_{ij}(t) = A_{ij}(t_n)$. Similar to [8], $A_{ij}(t)$ is right continuous and has left limit in $[0, \infty)$.

In reality, $D_{ij}(t)$ and $E_{ij}(t)$ are discrete functions, because a packet is removed from the buffer only after the whole packet has been fully transmitted. In order to make $D_{ij}(t)$ and $E_{ij}(t)$ continuous as in [8], in the following analysis, we assume that a bit is immediately released from the buffer after it has been transmitted. In other words, suppose that s_n $(s_0 = 0)$ and r_n $(r_0 = 0)$ are the time that the nth packet is removed from Q_{ij} and B_{ij}, respectively. For $s_n \le s < s_{n+1}$,

$$D_{ij}(s) = D_{ij}(s_n) + \frac{s - s_n}{s_{n+1} - s_n}\big(D_{ij}(s_{n+1})) - D_{ij}(s_n)\big), \tag{5}$$

and for $r_n \le r < r_{n+1}$,

$$E_{ij}(r) = E_{ij}(r_n) + \frac{r - r_n}{r_{n+1} - r_n}\big(E_{ij}(r_{n+1})) - E_{ij}(r_n)\big). \tag{6}$$

When $f(t)$ is differentiable at t, use $\dot{f}(t)$ to denote the derivative. Notice that $\dot{D}_{ij}(t)$ is equal to $2R$ when Q_{ij} is sending a packet to B_{ij}, and is zero otherwise. Similarly, $\dot{E}_{ij}(t)$ is equal to $2R$ when B_{ij} is sending a packet to the output queue and is zero otherwise.

Regarding the relationship of the defined variables, we have the following fluid equations:

$$Q_{ij}(t) = \lambda_{ij}t - D_{ij}(t), \tag{7}$$

$$B_{ij}(t) = D_{ij}(t) - E_{ij}(t), \tag{8}$$

$$Q_{ij}(t) + B_{ij}(t) = \lambda_{ij}t - E_{ij}(t). \tag{9}$$

It should be noted that, because $A_{ij}(t)$ satisfies an SLLN, we can use $\lambda_{ij}t$ to approximate $A_{ij}(t)$ when t is sufficiently large. In order to prove 100 percent throughput of LIPS, we need the following lemma from [8].

Lemma 1. *Let $f : [0, \infty) \to [0, \infty)$ be an absolutely continuous function with $f(0) = 0$. Assume that $\dot{f}(t) \le 0$ for almost every t (with respect to Lebesgue measure) such that $f(t) > 0$ and f is differentiable at t. Then, $f(t) = 0$ for almost every t.*

The basic idea to prove 100 percent throughput of LIPS is to first define the function $f(t)$ in Lemma 1 as follows:

$$V(t) = \sum_{ij} Q_{ij}(t)\left(\sum_k Q_{ik}(t)\right) + \sum_{ij} Z_{ij}(t)\left(\sum_k Z_{kj}(t)\right), \tag{10}$$

where $Z_{ij}(t)$ is the sum of $Q_{ij}(t)$ and $B_{ij}(t)$, i.e.,

$$Z_{ij}(t) = Q_{ij}(t) + B_{ij}(t). \tag{11}$$

Then, we prove that $V(t)$ has a negative derivative when it is positive, and thus by Lemma 1, $V(t) = 0$ for almost every t. Since $Z_{ij}(t) \le V(t)$, we know that $Z_{ij}(t) = 0$ for almost every t as well, which means that the length of each virtual queue is always bounded, or in other words, all incoming traffic is transmitted to the output queue.

Next, we introduce some supporting lemmas.

Lemma 2. *If B_{ij} is not empty at time t, $\sum_k Z_{kj}(t)$ has a negative derivative, i.e.,*

$$B_{ij}(t) > 0 \Rightarrow \sum_k \dot{Z}_{kj}(t) < 0. \tag{12}$$

Proof. The intuitive explanation for this lemma is that, if B_{ij} has buffered bits, Out_j must be receiving packets.

Since the crossbar scheduling of LIPS is work conserving and $B_{ij} > 0$, Out_j is either receiving bits from B_{ij} or another crosspoint buffer B_{kj}. Noticing that the bandwidth from the crosspoint buffer to the output queue is $2R$, we can obtain $\sum_k \dot{E}_{kj}(t) = 2R$.

According to the fluid equation $Z_{ij}(t) = \lambda_{ij}t - E_{ij}(t)$, we have

$$\begin{aligned}\sum_k \dot{Z}_{kj}(t) &= \sum_k \big(\lambda_{kj} - \dot{E}_{kj}(t)\big)\\ &= \sum_k \lambda_{kj} - \sum_k \dot{E}_{kj}(t)\\ &< R - 2R\\ &< 0.\end{aligned}$$

It should be noted that, due to the cut-through switching technique, even if B_{kj} has only received part of the packet but not the complete packet, B_{kj} can start to send the packet to the output queue of Out_j. □

Lemma 3. *If Q_{ij} is not empty at time t, $\sum_k Q_{ik}(t) + \sum_k Z_{kj}(t)$ has a negative derivative, i.e.,*

$$Q_{ij}(t) > 0 \Rightarrow \sum_k \dot{Q}_{ik}(t) + \sum_k \dot{Z}_{kj}(t) < 0. \tag{13}$$

Proof. The intuitive explanation for this lemma is that, when Q_{ij} has buffered bits, either a virtual queue of In_i is sending packets to its crosspoint buffer, or Out_j is receiving packets from one of the crosspoint buffers.

Based on the state of B_{ij}, we consider two possible cases.

Case 1. B_{ij} is empty. Since the input scheduling of LIPS is work conserving and $Q_{ij}(t) > 0$, either Q_{ij} is sending packets to B_{ij}, or another virtual queue Q_{ik} of the ith input port is sending packets to B_{ik}. For either case, $\sum_k \dot{D}_{ik} = 2R$.

Case 2. B_{ij} is occupied, including the case that B_{ij} has a fully buffered packet, and the case that B_{ij} is receiving a packet from Q_{ij} and simultaneously sending the packet to the output queue of Out_j and $B_{ij}(t) = 0$. Since the crossbar scheduling of LIPS is work conserving and the crossbar switching fabric uses cut-through switching, either B_{ij} or another crosspoint buffer B_{kj} is sending packets to Out_j. For either case, $\sum_k \dot{E}_{kj} = 2R$.

Note that $\dot{D}_{ij}(t) \geq 0$ and $\dot{E}_{ij}(t) \geq 0$. Thus, for both of the above cases, we can obtain $\sum_k \dot{D}_{ik}(t) + \sum_k \dot{E}_{kj}(t) \geq 2R$.

According to the fluid equations $Q_{ij}(t) = \lambda_{ij}t - D_{ij}(t)$ and $Z_{ij}(t) = \lambda_{ij}t - E_{ij}(t)$, we have

$$\begin{aligned}&\sum_k \dot{Q}_{ik}(t) + \sum_k \dot{Z}_{kj}(t)\\ &= \sum_k \big(\lambda_{ik} - \dot{D}_{ik}(t)\big) + \sum_k \big(\lambda_{kj} - \dot{E}_{kj}(t)\big)\\ &= \sum_k \lambda_{ik} + \sum_k \lambda_{kj} - \sum_k \dot{D}_{ik}(t) - \sum_k \dot{E}_{kj}(t)\\ &< R + R - 2R\\ &= 0.\end{aligned}$$

□

Theorem 1. *With speedup of two, LIPS achieves 100 percent throughput for any admissible traffic.*

Proof. We define

$$\begin{aligned}V(t) &= \sum_{ij} Q_{ij}(t)\left(\sum_k Q_{ik}(t)\right) + \sum_{ij} Z_{ij}(t)\left(\sum_k Z_{kj}(t)\right)\\ &= \sum_{ijk}\big(Q_{ij}(t)Q_{ik}(t) + Z_{ij}(t)Z_{kj}(t)\big).\end{aligned}$$

It is clear that $V(0) = 0$. In addition,

$$\begin{aligned}\dot{V}(t) &= \sum_{ijk}\big(\dot{Q}_{ij}(t)Q_{ik}(t) + Q_{ij}(t)\dot{Q}_{ik}(t) + \dot{Z}_{ij}(t)Z_{kj}(t)Z_{ij}(t)\dot{Z}_{kj}(t)\big)\\ &= 2\sum_{ijk}\big(Q_{ij}(t)\dot{Q}_{ik}(t) + Z_{ij}(t)\dot{Z}_{kj}(t)\big)\\ &= 2\sum_{ijk}\big(Q_{ij}(t)\dot{Q}_{ik}(t) + \big(Q_{ij}(t) + B_{ij}(t)\big)\dot{Z}_{kj}(t)\big)\\ &= 2\sum_{ijk} Q_{ij}(t)\big(\dot{Q}_{ik}(t) + \dot{Z}_{kj}(t)\big) + 2\sum_{ijk} B_{ij}(t)\dot{Z}_{kj}(t)\\ &= 2\sum_{ij} Q_{ij}(t)\left(\sum_k \dot{Q}_{ik}(t) + \sum_k \dot{Z}_{kj}(t)\right)\\ &\quad + 2\sum_{ij} B_{ij}(t)\left(\sum_k \dot{Z}_{kj}(t)\right).\end{aligned}$$

When $V(t) > 0$, there exists either $Q_{ij}(t) > 0$ or $B_{i'j'}(t) > 0$. By Lemma 3, we know that

$$2\sum_{ij} Q_{ij}(t)\left(\sum_k \dot{Q}_{ik}(t) + \sum_k \dot{Z}_{kj}(t)\right) \leq 0,$$

and it is strictly less than zero if $Q_{ij}(t) > 0$. Similarly, by Lemma 2,

$$2\sum_{ij} B_{ij}(t)\left(\sum_k \dot{Z}_{kj}(t)\right) \leq 0,$$

and it is strictly less than zero if $B_{i'j'}(t) > 0$. Thus, we can obtain that when $V(t) > 0$,

$$\dot{V}(t) < 0.$$

By Lemma 1, we know that $V(t) = 0$ for almost every t. Since $Q_{ij}(t) \leq V(t)$, $Q_{ij}(t) = 0$ for almost every t. Noticing that $B_{ij}(t) \leq L$, we can obtain

$$\begin{aligned}\lim_{t\to\infty}\frac{E_{ij}(t)}{t} &= \lim_{t\to\infty}\frac{A_{ij}(t) - Q_{ij}(t) - B_{ij}(t)}{t}\\ &= \lim_{t\to\infty}\frac{A_{ij}(t)}{t} - \lim_{t\to\infty}\frac{Q_{ij}(t) + B_{ij}(t)}{t}\\ &= \lambda_{ij} - 0\\ &= \lambda_{ij}.\end{aligned}$$

The above equation holds for any admissible traffic. Thus, by the definition, LIPS achieves 100 percent throughput. □

It should be pointed out that although it is desirable to analyze the switch throughput in a practical environment with bounded buffer size, no switch with bounded buffer size, independent of its structure and scheduling algorithm, can guarantee 100 percent throughput. Recall that λ_{ij} is the

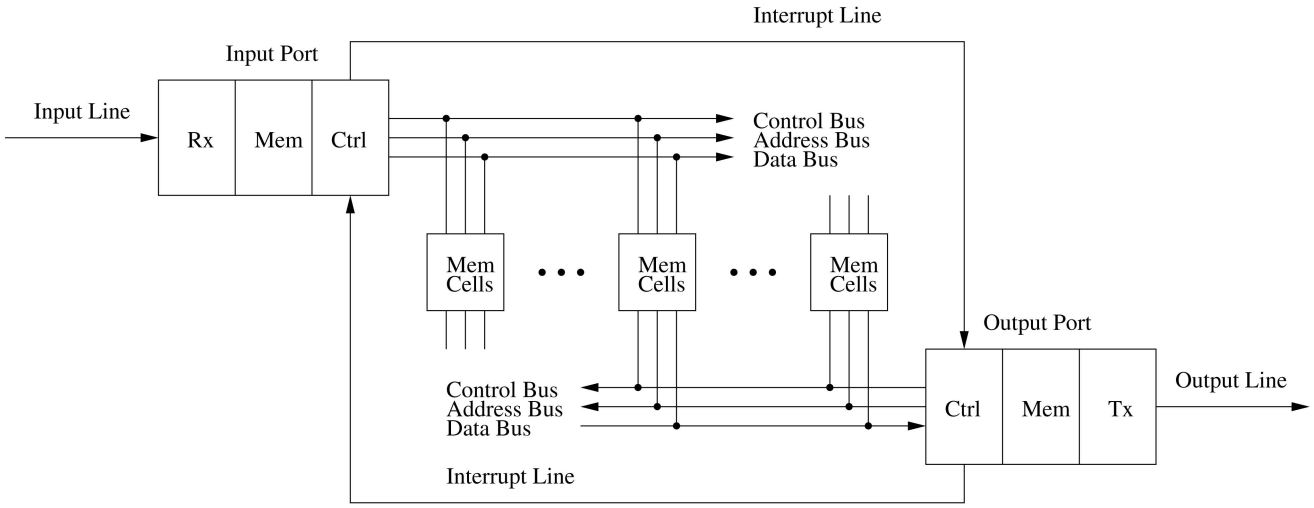

Fig. 3. The architecture diagram of LIPS.

long-term average traffic rate. Given bounded buffer size, it is possible to oversubscribe a specific output port for an arbitrarily long period but still satisfy the long-term average traffic rate, in which case there will be packet loss and a certain number of packets cannot be delivered to the output line.

In the following, we discuss the delay and queue length properties.

Before a packet is sent to the output line, it may be stored at the input buffer, the crosspoint buffer, and the output buffer. We define three types of delay to represent the waiting time of a packet at the three different locations. Input queuing delay is the interval from the time that the last bit of a packet arrives at the virtual queue to the time that the last bit of the packet leaves the virtual queue. In other words, input queuing delay measures the delay that a packet experiences at the input buffer. Similarly, we can define crossbar queuing delay and output queuing delay to be the time that a packet waits at the crosspoint buffer and at the output buffer, respectively. It should be noted that due to the cut-through switching technique, the last bit of a packet may leave the crosspoint buffer as soon as it arrives, which makes the crossbar queuing delay of the packet be zero.

Since the traffic arrival rate at an input port $\sum_i \lambda_{ij}$ is less than or equal to R, and the bandwidth of the crossbar is $2R$, most packets are immediately transmitted through the crossbar after they arrive but are buffered in output queues. This indicates short input and crossbar (IC) queuing delay and long output queuing delay. The observation is consistent with the simulation results obtained in Section 5. Assume that the traffic arrives according to a Poisson process and the packet length follows an exponential distribution with mean M. Then, In_i can be approximately modeled as an M/M/1 system, and accordingly,

$$\text{Average input queuing delay} = \frac{M}{2R - \sum_i \lambda_{ij}}. \qquad (14)$$

Applying Little's law, we can obtain

$$\text{Average input queue length} = \frac{M \sum_i \lambda_{ij}}{2R - \sum_i \lambda_{ij}}. \qquad (15)$$

4 Hardware Implementation

Practical scheduling algorithms are expected to be efficiently implemented in hardware to make fast decisions for high-speed switching. In this section, we discuss the implementation issues of LIPS.

4.1 Architecture and Timing

We first describe the implementation architecture and then give the timing for the entire transmission of a packet in the switch. The architecture diagram of LIPS is shown in Fig. 3. The left part depicts an input port, which consists of a receiver, a memory module, and a controller. The receiver is connected to the input line to accept packets, and the packets are stored into the corresponding virtual queues in the memory module based on their destinations. The controller makes scheduling decisions and sends packets stored in the memory to crosspoint buffers, which are shown in the middle of the figure. Since the input port sends a packet to only one crosspoint buffer at a time, the input port is connected to its crosspoint buffers through control, address, and data buses. Each crosspoint buffer contains a number of memory cells, so that it is sufficiently large to store a single packet of the maximum length. The output port, shown in the right part of the figure, is also connected to the memory cells of its crosspoint buffers through control, address, and data buses. It is symmetric to the input port, consisting of a controller, a memory module, and a transmitter. The controller is responsible for retrieving packets from the crosspoint buffers to the memory. The transmitter sends packets from the memory to the output line. The input port has an interrupt line connected to the output port and can send an interrupt to the output port when it starts sending a packet to the crosspoint buffer. Similarly, the output can send an interrupt to inform the

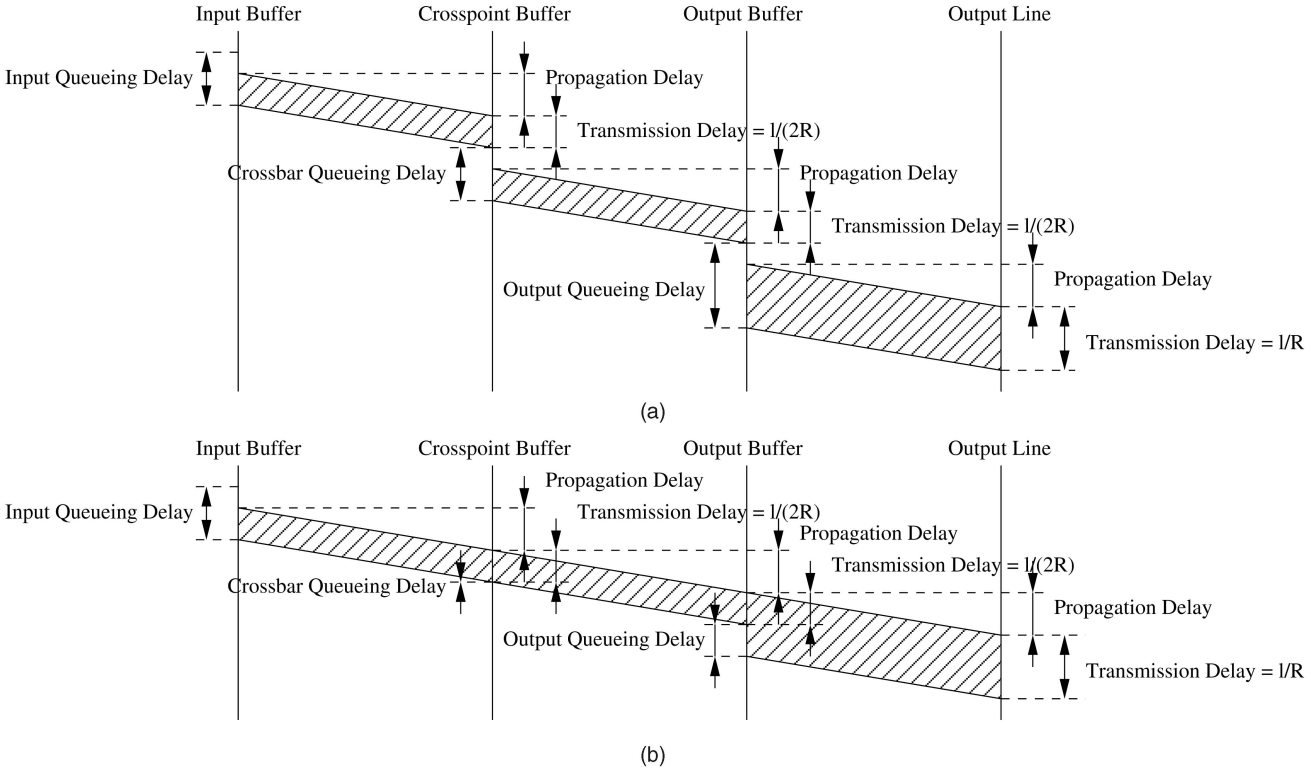

Fig. 4. The timing for the transmission of a packet. (a) Without cut-through switching. (b) With cut-through switching.

input port that it has finished retrieving a packet from the crosspoint buffer.

We now look at the transmission of a packet in LIPS and the timing of the transmission. A packet will be moved in sequence from the input buffer to the crosspoint buffer, the output buffer, and finally the output line. Fig. 4a illustrates the scenario that the packet is fully buffered at the crosspoint buffer and output buffer before being transmitted to the next stop. In Fig. 4b, as soon as the first bit of the packet arrives at the crosspoint buffer or output buffer, it is immediately sent out using cut-through switching. Figs. 4a and 4b represent the transmission timing of two extreme cases, but there are also other possible scenarios, such as the case that part of the packet has been received when the first bit is sent out.

We next explain in detail each step of the transmission along with the corresponding delay in the transmission timing. As the first step, the packet is received by the receiver and put into the input buffer, where it waits to be transmitted to the crosspoint buffer and experiences the input queuing delay. As defined in Section 3.3, input queuing delay is the interval from the time that the last bit of the packet arrives at the input buffer to the time that the last bit leaves the input buffer, as shown in the left side of both Figs. 4a and 4b.

The input port controller obtains the occupancy information of all its virtual queues from the memory module and knows whether the crosspoint buffers are empty from the interrupts sent by the output ports or by simple periodic polling. Based on a specific scheduling arbitration policy, the controller selects one backlogged virtual queue in the input buffer to send its head packet. The transmission of the packet from the input buffer to the crosspoint buffer can be done by either synchronous or asynchronous communication. To transmit the packet, the input port controller first puts the destination crosspoint buffer address on the address bus and asserts necessary control signals on the control bus indicating a write. The packet bits can then be put on the data bus from the input buffer and received by the crosspoint buffer. For synchronous communication, the control bus connecting the input port controller and the crosspoint buffer has a clock line to provide clock signals to synchronize all the operations. For asynchronous communication, acknowledgment-based handshaking protocols can be used between the sender and receiver. After the input port finishes sending one packet, it simply repeats the process for the next packet, thus variable packet length is no longer an issue. There are two parts in the transmission time of a packet from the input buffer to the crosspoint buffer: propagation delay and transmission delay, as shown in Figs. 4a and 4b. The propagation delay is the time for the signal of 1 bit to propagate from one end to the other end on the data bus. It is usually short because both ends are within the system. The transmission delay is the time for the input port to put all the bits of the packet on the data bus. Its value depends on the packet length and the data bus bandwidth. If we denote the packet length by l, the transmission delay is $l/(2R)$ because the bandwidth of the transmission channel connecting the input buffer and the crosspoint buffer is $2R$ due to the speedup of the crossbar.

Normally, when a packet is transmitted to the crosspoint buffer, it will wait some time before being sent to the output buffer, resulting in a nonzero crossbar queuing delay, as shown in Fig. 4a. However, when an input port starts transmitting a packet to the crosspoint buffer, it can send an interrupt request to the corresponding output port as a notification. If the output port is currently not receiving any packet, it handles the interrupt by immediately starting to retrieve the packet from the crosspoint buffer to achieve cut-through switching. In this case, the packet has a zero crossbar queuing delay, as indicated in Fig. 4b. Otherwise, if the output port is already busy in receiving another packet, it handles the interrupt by simply updating the status of the specific crosspoint buffer to occupied, so that it is eligible for crossbar scheduling later.

For an output port, it can obtain the statuses of its crosspoint buffers from the interrupts sent by the input ports or by periodic polling. The output port controller uses an arbitration policy to decide from which occupied crosspoint buffer to retrieve the next packet. Similarly, the communication between the crosspoint buffer and the output buffer can be either synchronous or asynchronous. The transmission time of the packet from the crosspoint buffer to the output buffer also includes the propagation delay and transmission delay, and the latter is $l/(2R)$, as shown in Figs. 4a and 4b. When the output port receives the first bit of a packet, it checks whether its transmitter is busy. If the transmitter is idle, it can immediately start sending the packet to the output line using cut-through switching. However, the transmitter has smaller bandwidth than the crossbar, and thus there is always a nonzero output queuing delay, as shown in Figs. 4a and 4b. Since the bandwidth of the transmitter is R, it can be calculated that the transmission delay for the transmitter to send the packet to the output line is l/R. After the output port finishes retrieving a packet from the crosspoint buffer, it can send an interrupt to the corresponding input port so that the input port can update its status of the crosspoint buffer to empty to be eligible to receive a new packet.

4.2 Implementation Advantages

We discuss in the following some unique implementation advantages of LIPS.

First, the localized feature makes LIPS suitable for a distributed implementation. In LIPS, each input port or output port makes scheduling decisions solely based on the state information of its local crosspoint buffers, and therefore, the scheduling of different input ports or output ports can be conducted in an independent and asynchronous mode. Since there is no information exchange between different arbiters, LIPS can be implemented in a distributed manner, which makes it highly scalable.

Moreover, because LIPS requires no comparison, the scheduling arbiters can be efficiently implemented using priority encoders [14] to make fast decisions in hardware. The theoretical time complexity to make an arbitration is $O(\log N)$. In practice, priority encoders perform all the operations in hardware and technically achieve constant time complexity for moderate switch size [36]. Depending on the arbitration rules, different types of priority encoders may be used. For example, if arbitration candidates are assigned different priorities at different time, such as the situation in an RR arbiter, a programmable priority encoder can be used to implement the arbiter.

At the first glance, the cost of crosspoint buffers may seem to be a problem for the implementation of buffered crossbar switches. Fortunately, with the recent development of VLSI technology, it has been feasible to integrate a small amount of memory to chips. For example, the latest Quad-Core Intel Xeon Processor 5400 Series have 12-Mbyte integrated on-chip memory used as L2 caches [30]. Several implemented prototypes of buffered crossbar switches have been reported in the literature [20], [31], [32], [33]. In addition, LIPS requires only L buffer space at each crosspoint and minimizes the switch hardware cost. If the switch size N is 32, and the maximum packet length L is equal to 1,500 bytes, thus 1.5-Mbyte on-chip memory is sufficient for all crosspoint buffers.

4.3 Comparison with Other Schemes

In this section, we compare LIPS with existing scheduling algorithms and summarize its characteristics.

First of all, LIPS is a packet scheduling algorithm dealing with variable length packets. Most existing scheduling algorithms operate on fixed length cells, for which packets have to be segmented into cells upon arrival, and the cells are then reassembled back to the original packets before departure. Based on the buffered crossbar structure, LIPS improves the switch efficiency by eliminating the SAR process and achieves higher throughput and shorter packet latency.

Second, LIPS has low time complexity. The time complexity of LIPS is $O(\log N)$, because each input port or output port makes scheduling decisions independently by selecting an arbitrary candidate from up to N candidates. Furthermore, as analyzed above, the arbiters can be implemented using priority encoders to make fast arbitration in hardware, which technically achieves constant time complexity for moderate switch size [36]. In comparison, most existing scheduling algorithms have higher time complexity. For example, maximum size matching and maximum weight matching algorithms have time complexity of $O(N^{2.5})$ [34] and $O(N^3)$ [35], respectively. Maximal matching algorithms usually work in iterative modes with $\ln N + e/(e-1)$ average convergence iterations [9], while the time complexity of each iteration is the same as that of LIPS. Algorithms emulating OQ schedulers need to maintain the reference scheduling systems and sort packets based on their leaving sequences in the reference systems, and are obviously more complex than LIPS.

Finally, LIPS is a truly distributed algorithm. In LIPS, the arbiter of an input port or output port only needs the state information of its local crosspoint buffers, and thus different arbiters can be distributed at different locations. On the contrary, most existing algorithms require different arbiters to exchange information when making scheduling decisions. Maximum size matching and maximum weight matching algorithms collect the information of all the queues to compute a globally maximum value. In iterative maximal matching algorithms, output ports need to receive requests from and send grants back to input ports. For algorithms emulating OQ schedulers, packets of different input ports destined for the same output port are compared so that their departure order is consistent with that in the reference system.

5 Simulation Results

We have conducted simulations to verify the 100 percent throughput of LIPS and to evaluate its performance.

For the input or crossbar scheduling of LIPS, when there are more than one eligible virtual queues or crosspoint buffers, different scheduling decisions can be made depending on the rules to make the arbitration. In the simulations, we consider five different LIPS implementation versions with different arbitration rules:

1. FP assigns an FP order to all the virtual queues of the same input port or all the crosspoint buffers to the same output port and always picks the candidate with the highest priority. In our implementation, higher priorities are assigned to virtual queues to output ports with smaller indices (e.g., Q_{ij} has higher priority than Q_{ij+1}), or to crosspoint buffers from input ports with smaller indices (e.g., B_{ij} has higher priority than B_{i+1j}).
2. Random (RD) does not favor any particular candidate but makes arbitration on a random basis.
3. RR sets up an RR pointer for the virtual queues of the same input port or the crosspoint buffers to the same output port and grants to the first candidate that is equal to or larger than the RR pointer (in a modular manner). After making an arbitration, the RR pointer is updated to the next candidate of the current assigned one (in a modular manner).
4. Oldest packet first (OPF) uses the packet arrival time as the arbitration criterion. In input scheduling, the eligible virtual queue whose head packet arrives earliest at the input port is selected. In crossbar scheduling, the eligible crosspoint buffer whose packet arrives earliest at the crosspoint is selected.
5. Longest queue first (LQF) uses the queue length as the arbitration criterion. In input or crossbar scheduling, the eligible packet whose virtual queue or crosspoint buffer has the longest queue is selected.

FP, RD, and RR rely only on the state information, i.e., for input scheduling, whether a virtual queue has packets and its crosspoint buffer is available, and for crossbar scheduling, whether a crosspoint buffer has a buffered packet. As discussed earlier, since there is no comparison operation involved, these algorithms can be efficiently implemented using priority encoders. In particular, RR is also able to avoid starvation in the scheduling, by giving each candidate the chance to obtain the highest priority. On the other hand, LQF and OPF need to compare either the packet arrival time or the queue length when making arbitrations and require more sophisticated hardware support. Our purpose to include these two algorithms is that LQF and OPF demonstrate advantages in the scheduling for VOQ switches [7], and we want to study whether they are also superior in the scheduling for buffered crossbar switches.

In order to reflect the burst nature of real network traffic, we emulate the incoming traffic by a Markov modulated Poisson process, as illustrated in Fig. 5. The intensity of the Poisson process is defined by the state of a Markov chain. The Markov chain has two states: ON and OFF. In the ON state, the intensity of the Poisson process is λ_1, and in the OFF state the intensity is λ_2. The probability to switch from the ON state to the OFF state is p, and the probability to switch from the OFF state to the ON state is q. In the simulations, we set $p = q = 0.2$ and $\lambda_2 = 0$, and change the value of λ_1 to adjust the load.

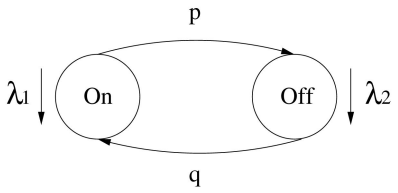

Fig. 5. In a Markov modulated Poisson process, the intensity of the Poisson process depends on the state of the Markov chain.

For the destination of the packets, we consider both uniform traffic and nonuniform traffic. For uniform traffic, the destination of a new incoming packet is uniformly distributed among all the output ports, i.e., $\lambda_{ij} = lR/N$, where l is the effective load. For nonuniform traffic, we use the same model as that in [18] and [21]. The traffic arrival rate λ_{ij} is defined by i, j, and an unbalanced probability w as follows:

$$\lambda_{ij} = \begin{cases} lR\left(w + \frac{1-w}{N}\right), & \text{if } i = j, \\ lR\frac{1-w}{N}, & \text{if } i \neq j. \end{cases} \quad (16)$$

The packet length in the simulation is uniformly distributed between [50, 1,500] bytes. We consider a 16×16 switch, and each input port or output port has a bandwidth of 1 Gbps. All packets to an output port are buffered in the same queue, and FIFO is used as the output scheduling policy for all the algorithms.

5.1 Throughput

In Section 3, we have theoretically proved that with speedup of two LIPS achieves 100 percent throughput for any admissible traffic. Now, we verify the analytical results by simulation.

Fig. 6a depicts the relationship between the throughput of different algorithms and the effective load under uniform traffic. As can be seen, all algorithms have similar curves and achieve 100 percent throughput. Fig. 6b shows the results under nonuniform traffic. We fix the load of the switch to one and adjust the unbalanced probability. Again, all the five algorithms achieve 100 percent throughput. As we have seen, both the simulation data under uniform traffic and nonuniform traffic support the previous analytical results well. It also can be noticed that, when speedup is equal to two, the five algorithms have no significant difference on the throughput performance.

In the rest of the simulations, the unbalanced probability of the nonuniform traffic is fixed to 0.5 if not specifically noted.

5.2 Average Delay

In this section, we study the delay performance of different algorithms. In Section 3.3, we defined the input, crossbar, and output queuing delay to be the waiting time of a packet at the input, crosspoint, and output buffer, respectively. In particular, we theoretically analyzed the input queuing delay and obtained an expression to compute it under the assumption of Poisson arrival and exponential packet

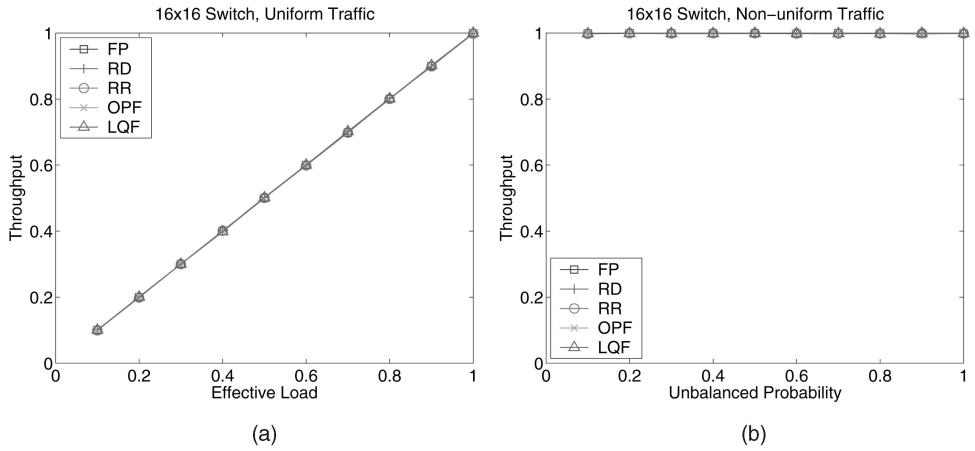

Fig. 6. Throughput of different algorithms. (a) Uniform traffic. (b) Nonuniform traffic.

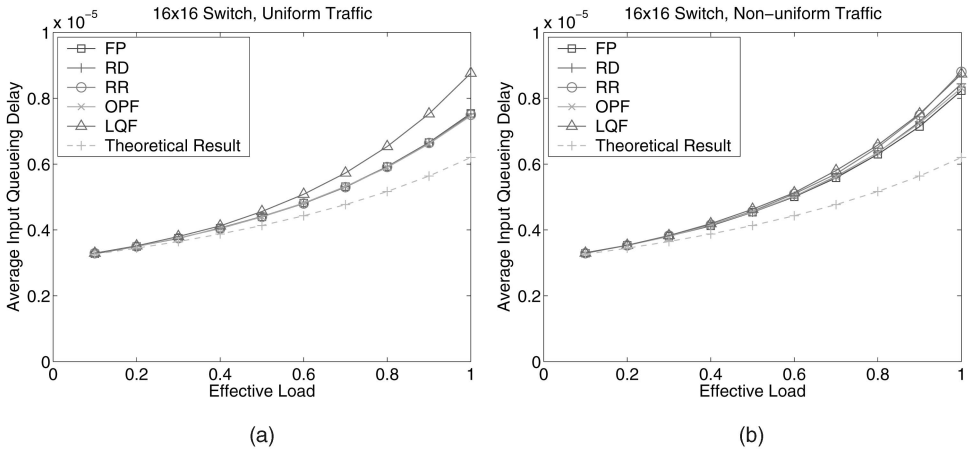

Fig. 7. Average delay of different algorithms. (a) Uniform traffic. (b) Nonuniform traffic.

length distribution. In the following, we verify the applicability of the expression by simulation.

Figs. 7a and 7b show the input queuing delay of different algorithms under uniform traffic and nonuniform traffic, respectively. We can see that all the five algorithms have similar input queuing delay, which grows gradually with the increase of the effective load. The theoretical results are also illustrated in the figures by dashed lines. Although the incoming traffic is not Poisson traffic and the packet length does not follow an exponential distribution, it can be seen that the theoretical results are reasonably consistent with the simulation results, especially when the effective load is small.

Next, we consider tw more delay measures. The first one is called nodal delay, which is the sum of the input, crossbar, and output queuing delay. Nodal delay is the total time that a packet stays in the switch and is an important performance criterion. The other measure is called IC queuing delay, which is the sum of the input and crossbar queuing delays. In this paper, we mainly discuss the input scheduling and crossbar scheduling of buffered crossbar switches, and all the simulation algorithms use the same output scheduling principle. Thus, IC queuing delay is a good measure to compare the different arbitration rules used in different algorithms. On the other hand, the nodal delay of a packet is equal to its IC queuing delay plus output queuing delay. With the above two measures, it is possible to determine the proportion of time that packets spend at different buffering locations in the switch.

The average nodal delay and IC queuing delay of the five algorithms under uniform traffic and nonuniform traffic are shown in Figs. 8a and 8b, respectively. The solid lines represent the nodal delay, and the dashed lines represent the IC queuing delay. First, we can notice that, for any algorithm, compared with the nodal delay (10^{-3} second), the IC queuing delay (10^{-5} second) is small enough to be neglected. Second, we can also see that the IC queuing delay of different algorithms does not have significant difference. Combining the above two observations, we can draw the conclusion that, for buffered crossbar switches with speed-up of two, input scheduling and crossbar scheduling do not significantly affect the nodal delay of the packet. Thus, when considering implementation cost, the simplest algorithms, such as RR and FP, are the preferred choices. On the other hand, since the IC queuing delay is very small, most packets are immediately transmitted to their output buffers after they arrive at input ports. Thus, we can expect LIPS to exhibit similar performance as OQ switch-based scheduling algorithms. Moreover, in conjunction with LIPS, existing fair scheduling algorithms, such as WFQ and DRR, can be used as the output scheduling principles to provide deterministic performance guarantees.

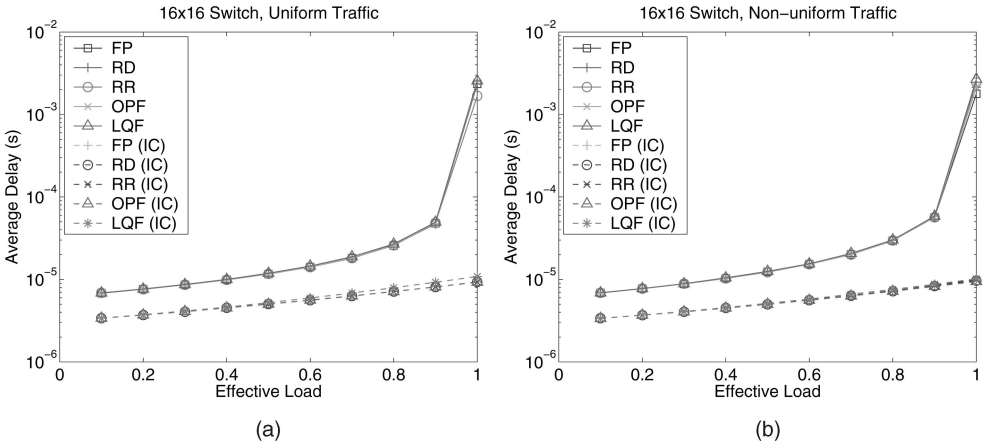

Fig. 8. Average delay of different algorithms. (a) Uniform traffic. (b) Nonuniform traffic.

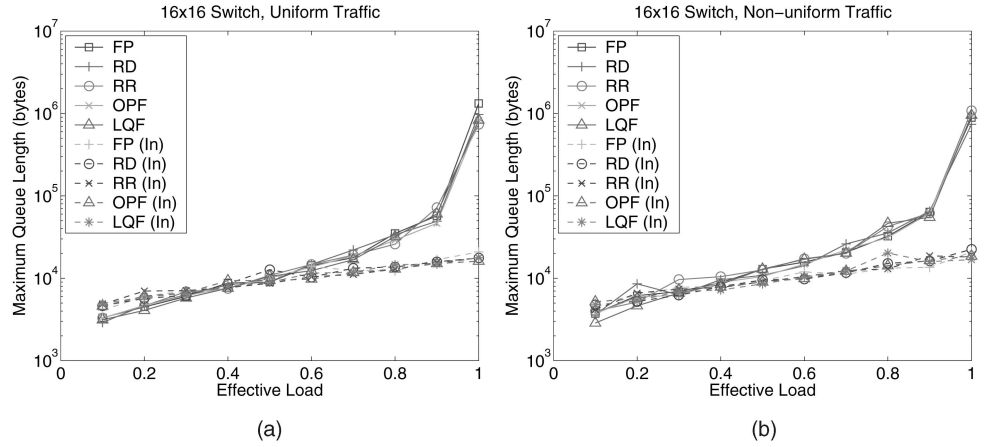

Fig. 9. Maximum queue length of different algorithms. (a) Uniform traffic. (b) Nonuniform traffic.

5.3 Maximum Queue Length

In order to achieve 100 percent throughput for admissible traffic, input ports and output ports must have enough buffer space to avoid packet overflow. We also collect the maximum queue length at both the input side and output side during the simulations to reveal the buffer requirement of the algorithms. The maximum output queue length is defined to be the maximum number of bytes buffered at any output queue during the entire simulation time. The maximum input queue length is defined to be the maximum number of bytes buffered at all the virtual queues of any input port.

Figs. 9a and 9b show the maximum queue length of the algorithms under uniform traffic and nonuniform traffic, respectively. The solid lines represent the maximum output queue length, and the dashed lines represent the maximum input queue length. It can be seen that all the algorithms exhibit similar buffer requirement at both the input side and the output side. On the other hand, the input ports have much shorter maximum queue length than the output ports. This indicates that, with speedup of two, packets can be quickly transferred through the crossbar, and more packets are buffered at output ports than at input ports.

5.4 Speedup Less than Two

In the above, we have theoretically proved and verified by simulation that LIPS achieves 100 percent throughput with speedup of two. With two being the theoretical bound, we are interested in finding out whether it is possible for LIPS to achieve 100 percent throughput with speedup less than two in practice. In the following, we measure the throughput performance of LIPS under different speedup values.

Among the five algorithms, we choose two for this experiment, in which RR is a representative of the algorithms without sorting and OPF of the algorithms with sorting involved. We fix the load to be one and adjust the speedup value from 1 to 2 with a step of 0.1 and adjust the unbalanced probability from 0 to 1 with a step of 0.1. The throughput of the RR and OPF is depicted in Figs. 10a and 10b, respectively.

First, we look at the Y-axis, which denotes the speedup value. As can be seen, when the speedup is approximately 1.3, both RR and OPF have reached 100 percent throughput under all different unbalanced probabilities. This does not necessarily mean that speedup of 1.3 can guarantee 100 percent throughput for LIPS, because there is no way to test it under all possible traffic. However, the results do indicate that the theoretical bound of two may be further tightened, which is an interesting problem for our future work.

We now look at the X-axis, which is the unbalanced probability. We can notice that, under the same speedup, the two algorithms achieve the lowest throughput when the unbalanced probability is around 0.5. This observation is consistent with the definition of the nonuniform traffic.

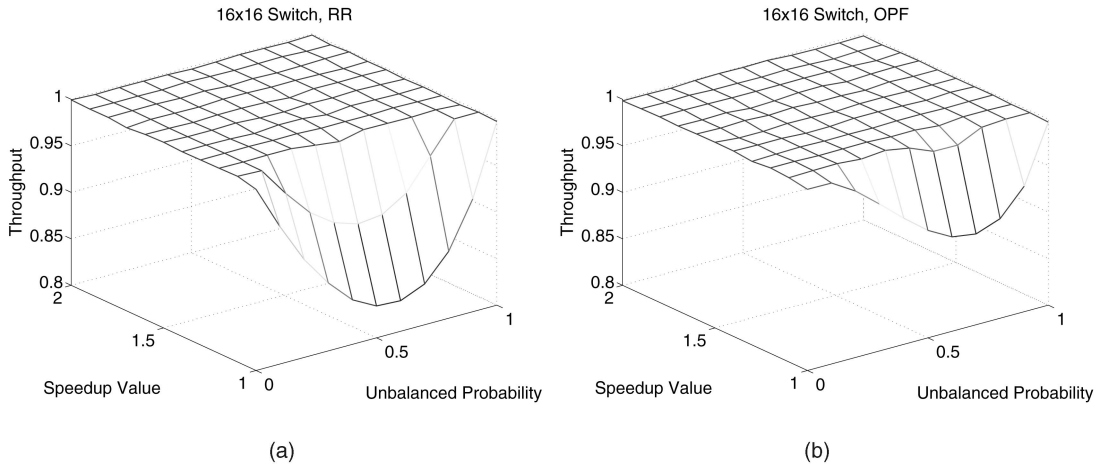

Fig. 10. Throughput with speedup less than two. (a) RR. (b) OPF.

When the unbalanced probability is equal to zero, the defined traffic is actually the uniform traffic, and the packets from any input port have equal probabilities to go to different output ports. The uniform traffic is friendly to scheduling algorithms and enables them to easily achieve 100 percent throughput. On the other hand, if the unbalanced probability is equal to one, all the packets of an input port are destined for the same output port, and packets from different input ports have different destinations. In this situation, no scheduling is necessary. By always connecting every input port with its corresponding output port, 100 percent throughput is guaranteed. When the unbalanced probability is between zero and one, packets are distributed to all output ports but one of them receives more packets than the rest, and the traffic is more difficult to schedule.

Comparing the two figures, we can see that, when the speedup value is less than 1.3, OPF almost always achieves higher throughput than RR with the same speedup value and unbalanced probability. This is easy to explain. RR, FP, and RD always equally treat all the scheduling candidates, while OPF and LQF take the traffic distribution information into consideration. The observation also suggests that, if the speedup bound of two is to be tightened, sorting-based algorithms may be first considered.

6 Conclusions and Future Work

In this paper, we have studied packet scheduling for buffered crossbar switches. Buffered crossbar switches are a special type of CIOQ switches, whose crosspoints are associated with small on-chip buffers. The introduction of crosspoint buffers eliminates output and input contention and greatly simplifies the scheduling process. Furthermore, the scheduling of different input ports or output ports are conducted in an independent and asynchronous mode, and variable length packets can be directly scheduled and transmitted without segmentation or reassembly. Compared with cell scheduling, packet scheduling has some unique advantages: higher throughput, shorter packet delay, and cheaper hardware cost. We have presented a packet scheduling scheme called LIPS for buffered crossbar switches. With LIPS, each crosspoint needs as little as L buffer space, which minimizes the hardware cost for switches. Another advantage of LIPS is that the scheduling of an input port or output port relies only on the state information of its local crosspoint buffers. The localization feature makes LIPS suitable for a distributed implementation and thus highly scalable. Since there is no comparison needed, priority encoders can be used to quickly make scheduling arbitrations in hardware. We theoretically proved that LIPS with speedup of two achieves 100 percent throughput for any admissible traffic and presented the detailed implementation architecture of LIPS. We also conducted extensive simulations to verify the analytical results and evaluate the performance of LIPS.

Finally, regarding future work, there are several ways to further extend the work in this paper, and the following problems are of particular interest. First, it might be possible to further reduce the size of crosspoint buffer. We showed that each crosspoint buffer only needs to store a single packet to achieve 100 percent throughput. However, when the incoming packets have an extremely large maximum packet length, the crossbar will require a large amount of buffer space. Moreover, if most packets have shorter lengths than the maximum length, the crosspoint buffers may have low utilization. It might be possible to break a large packet into smaller pieces (but not necessarily fixed length cells) to transmit and still guarantee 100 percent throughput, which will substantially reduce the total size of crosspoint buffers. Second, loosen or eliminate the interaction between input/output ports and crosspoint buffers. In the current design, input/output ports need to check whether the corresponding crosspoint buffers are empty or occupied before making scheduling decisions. Although a single bit is sufficient to indicate the status of a crosspoint buffer, the transmission of the status information still introduces delay to the switch scheduling. Potential techniques may be explored to compensate the delay or completely remove it. Third, tighten the speedup requirement. As indicated by the simulation results in Section 5.4, the speedup bound of two for 100 percent throughput might be further tightened. How to use sorting-based scheduling strategies to lower the speedup requirement is an interesting and important question, which will allow the same crossbar hardware to deliver a significantly higher capacity.

ACKNOWLEDGMENTS

This research work was supported in part by the US National Science Foundation under Grant CCR-0207999 and Grant CCF-0744234.

REFERENCES

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. ACM SIGCOMM '89,* vol. 19, no. 4, pp. 3-12, Sept. 1989
[2] M. Shreedhar and G. Varghese, "Efficient Fair Queuing Using Deficit Round Robin," *IEEE/ACM Trans. Networking,* vol. 4, no. 3, pp. 375-385, June 1996.
[3] D. Pan and Y. Yang, "Credit Based Fair Scheduling for Packet Switched Networks," *Proc. IEEE INFOCOM '05,* pp. 843-854, Mar. 2005.
[4] T. Anderson, S. Owicki, J. Saxe, and C. 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, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Trans. Networking,* vol. 7, no. 2, pp. 188-201, 1999.
[6] H.J. Chao, "Saturn: A Terabit Packet Switch Using Dual Round-Robin," *IEEE Comm. Magazine,* vol. 8, no. 12, pp. 78-84, Dec. 2000.
[7] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100 Percent Throughput in an Input Queued Switch," *IEEE Trans. Comm.,* vol. 47, no. 8, pp. 1260-1267, 1999.
[8] J.G. Dai and B. Prabhakar, "The Throughput of Data Switches with and without Speedup," *Proc. IEEE INFOCOM '00,* vol. 2, pp. 556-564, Mar. 2000.
[9] D. Pan and Y. Yang, "Pipelined Two Step Iterative Matching Algorithms for CIOQ Crossbar Switches," *Proc. ACM/IEEE Symp. Architectures for Networking and Comm. Systems (ANCS '05),* Oct. 2005.
[10] M.J. Karol, M.J. Hluchyj, and S.P. Morgan, "Input versus Output Queueing on a Space-Division Packet Switch," *IEEE Trans. Comm.,* vol. 35, pp. 1347-1356, 1987.
[11] N. McKeown, "A Fast Switched Backplane for a Gigabit Switched Router," *Business Comm. Rev.,* vol. 27, no. 12, 1997.
[12] J. Turner, "Strong Performance Guarantees for Asynchronous Crossbar Schedulers," *Proc. IEEE INFOCOM '06,* Apr. 2006.
[13] J. Xu and R. Lipton, "On Fundamental Tradeoffs between Delay Bounds and Computational Complexity in Packet Scheduling Algorithms," *Proc. ACM SIGCOMM '02,* Aug. 2002.
[14] M. Morris Mano, *Digital Design,* third ed. Prentice Hall, Aug. 2001.
[15] I. Stoica and H. Zhang, "Exact Emulation of an Output Queueing Switch by a Combined Input Output Queueing Switch," *Proc. Sixth IEEE/IFIP Int'l Workshop Quality of Service (IWQoS '98),* pp. 218-224, 1998.
[16] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhkar, "Matching Output Queueing with a Combined Input Output Queued Switch," *Proc. IEEE INFOCOM '99,* pp. 1169-1178, 1999.
[17] D. Stephens and H. Zhang, "Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture," *Proc. IEEE INFOCOM '98,* pp. 282-290, Mar. 1998.
[18] R. Rojas-Cessa, E. Oki, Z. Jing, and H.J. Chao, "CIXB-1: Combined Input-Once-Cell-Crosspoint Buffered Switch," *Proc. IEEE Workshop High Performance Switching and Routing (HPSR '01),* July 2001.
[19] R. Rojas-Cessa, E. Oki, and H.J. Chao, "CIXOB-k: Combined Input-Crosspoint-Output Buffered Packet Switch," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '01),* Nov. 2001.
[20] G. Kornaros, "BCB: A Buffered Crossbar Switch Fabric Utilizing Shared Memory," *Proc. Ninth EUROMICRO Conf. Digital System Design (DSD '06),* pp. 180-188, Aug. 2006.
[21] L. Mhamdi and M. Hamdi, "MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches," *IEEE Comm. Letters,* vol. 7, no. 9, pp. 451-453, Sept. 2003.
[22] X. Zhang and L. Bhuyan, "An Efficient Scheduling Algorithm for Combined-Input-Crosspoint-Queued (CICQ) Switches," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '04),* Nov. 2004.
[23] S. He, S. Sun, H. Guan, Q. Zheng, Y. Zhao, and W. Gao, "On Guaranteed Smooth Switching for Buffered Crossbar Switches," *IEEE/ACM Trans. Networking,* vol. 16, no. 3, June 2008.
[24] M. Lin and N. McKeown, "The Throughput of a Buffered Crossbar Switch," *IEEE Comm. Letters,* vol. 9, no. 5, pp. 465-467, May 2005.
[25] L. Mhamdi and M. Hamdi, "Output Queued Switch Emulation by a One-Cell-Internally Buffered Crossbar Switch," *Proc. IEEE Global Telecommunications Conf. (GLOBECOM '03),* vol. 7, pp. 3688-3693, Dec. 2003.
[26] S. Chuang, S. Iyer, and N. McKeown, "Practical Algorithms for Performance Guarantees in Buffered Crossbars," *Proc. IEEE INFOCOM '05,* Mar. 2005.
[27] B. Magill, C. Rohrs, and R. Stevenson, "Output-Queued Switch Emulation by Fabrics with Limited Memory," *IEEE J. Selected Areas in Comm.,* vol. 21, no. 4, pp. 606-615, May 2003.
[28] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos, "Variable Packet Size Buffered Crossbar (CICQ) Switches," *Proc. IEEE Int'l Conf. Comm. (ICC '04),* vol. 2, pp. 1090-1096, June 2004.
[29] M. Katevenis and G. Passas, "Variable-Size Multipacket Segments in Buffered Crossbar (CICQ) Architectures," *Proc. IEEE Int'l Conf. Comm. (ICC '05),* May 2005.
[30] ftp://download.intel.com/products/processor/xeon/dc54kprodbrief.pdf, 2008.
[31] L. Mhamdi, C. Kachris, and S. Vassiliadis, "A Reconfigurable Hardware Based Embedded Scheduler for Buffered Crossbar Switches," *Proc. 14th ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays (FPGA '06),* pp. 143-149, Feb. 2006.
[32] I. Papaefstathiou, G. Kornaros, and N. Chrysos, "Using Buffered Crossbars for Chip Interconnection," *Proc. 17th Great Lakes Symp. VLSI,* pp. 90-95, Mar. 2007.
[33] K. Yoshigoe, K. Christensen, and A. Jacob, "The RR/RR CICQ Switch: Hardware Design for 10-Gbps Link Speed," *Proc. 22nd IEEE Int'l Performance, Computing, and Comm. Conf. (IPCCC '03),* pp. 481-485, Apr. 2003.
[34] J. Hopcroft and R. Karp, "An $N^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs," *SIAM J. Computing,* vol. 2, no. 4, pp. 225-231, Dec. 1973.
[35] R. Tarjan, "Data Structures and Network Algorithms," *Proc. CBMS-NSF Regional Conf. Series in Applied Math.,* Dec. 1983.
[36] S. Ramabhadran and J. Pasquale, "Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay," *Proc. ACM SIGCOMM '03,* pp. 239-250, Aug. 2003.

Deng Pan received the BS and MS degrees in computer science from Xi'an Jiaotong University, Xi'an, China, in 1999 and 2002, respectively, and the PhD degree in computer science from the State University of New York, Stony Brook, in 2007. He is currently an assistant professor in the Department of Electrical and Computer Engineering, Florida International University. His research interests include high-speed networks and high-performance computer architectures. He is a member of the IEEE.

Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore. She is a professor of computer engineering and computer science at the State University of New York, Stony Brook. Her research interests include wireless networks, optical networks, high-speed networks, and parallel and distributed computing systems. Her research has been supported by the National Science Foundation (NSF) and US Army Research Office (ARO). She has published more than 200 papers in major journals and refereed conference proceedings and holds six US patents in these areas. She is currently an editor for *IEEE Transactions on Computers* and *Journal of Parallel and Distributed Computing*, and has served as an editor for *IEEE Transactions on Parallel and Distributed Systems*. She has also served on program/organizing committees of numerous international conferences in her areas of research. She is a fellow of the IEEE and a member of the IEEE Computer Society.