When is a Work-Conserving Switch Not?

Jonathan S. Turner
jon.turner@wustl.edu
WUCSE-2005-14

April 22, 2005

Department of Computer Science
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

Abstract

Crossbar-based switches are commonly used to implement routers with throughputs up to about 1 Tb/s. The advent of work-conserving crossbar scheduling algorithms now makes it possible to engineer systems that perform well, even under extreme traffic conditions. Unfortunately, all the published work-conservation results for crossbar scheduling apply only to systems that switch fixed-length cells, not variable length packets. Routers that use a cell-based crossbar with a nominally work-conserving scheduler to switch variable length packets can fail to be work-conserving at the external links, since the router cannot forward a packet until all of its constituent cells reach the output line card. Speedups as large as the number of inputs and outputs can be required to achieve work-conservation, using schedulers that operate only on cells. There appear to be fundamental obstacles to achieving practical work-conservation for variable length packet switches based on unbuffered crossbars. However, we show that adding buffers to crossbars allows work-conservation to be achieved for variable length packet switching, using modest speedups. In particular we define packet versions of the Group by Virtual Output Queue (GVOQ) scheduler of Chuang et. al. and the Least Occupied Output First (LOOFA) scheduler of Krishna et. al. and show that they are both work-conserving for speedups ≥ 2. Specific versions of both algorithms are also shown to be order-preserving for speedups ≥ 2, meaning that they can exactly emulate an ideal, output queued switch.

This work is supported by the National Science Foundation (grant #CNS-0325298).

When is a Work-Conserving Switch Not?

Jonathan Turner

Washington University
jon.turner@wustl.edu

1. INTRODUCTION

Crossbar switches have long been a popular choice for transferring data from inputs to outputs in mid-range performance switches and routers [1]. Unlike bus-based switches, crossbars can provide throughputs approaching 1 Tb/s, while allowing individual line cards to operate at speeds comparable to the external links. However the control of high performance crossbars is challenging, requiring *crossbar schedulers* that match inputs to outputs in the time it takes for a minimum length packet to be forwarded. The matching selected by the scheduler has a major influence on system performance, placing a premium on algorithms that can produce high quality matchings in a very short period of time.

Traditionally, crossbars schedulers have been evaluated largely on the basis of how they perform on random traffic arrival patterns that do not cause long term overloads at inputs or outputs. More recently, there has been growing interest in crossbar schedulers that are capable of performing well under worst-case conditions, including traffic patterns that involve extended overload periods at some outputs [2,5,8]. Since extended overloads are a common occurrence in internet routers, it is important to understand how crossbar schedulers perform in such situations.

There are two fundamental properties that are commonly used to evaluate crossbar schedulers in this worst-case sense. A scheduler is said to be *work-conserving* if an output link is kept busy so long there are packets addressed to the output, anywhere in the system. A scheduler is said to be *order-preserving* if it is work-conserving and it always forwards packets in the order in which they arrived. A crossbar with an order-preserving scheduler faithfully emulates an ideal non-blocking switch with FIFO output queues. In their seminal paper, Chuang, et. al. provided the first example of an order-preserving scheduler [2] for a crossbar with small speedup. (The *speedup* of a crossbar switch is the ratio of the ideal throughput of the crossbar to the total capacity of its external links. So a crossbar with a speedup of S has the potential to forward data S times faster than the input links can supply it.) Krishna, et. al. showed that a somewhat simpler scheduler is work-conserving for the same speedup (S=2) and Rodeheffer and Saxe [10] showed that this algorithm can be made order-preserving if the speedup is increased to 3.

The work-conservation properties that have been established to date, apply only to crossbars that forward fixed length data units, or *cells*. There is a sound practical justification for concentrating on such systems, since routers commonly use cell-based crossbars. Variable length packets are received at input line cards, segmented into fixed length cells for transmission through the crossbar and reassembled at the output line cards. This simplifies the implementation of the crossbar and allows for synchronous operation, which allows the scheduler to make better scheduling decisions than would be possible with asynchronous operation. Unfortunately, cell-based crossbar schedulers that are work-conserving when viewed from the edge of the crossbar, can fail to deliver work-conserving operation for the router as a whole. That is, an outgoing link may remain idle, even while complete packets for that link are present in the system. We show that much larger speedups are needed to make routers built around cell-oriented crossbar schedulers work-conserving. However, we also show that work-conservation can be restored, using crossbar schedulers that schedule packets, rather than cells, if the crossbars are equipped with a moderate amount of internal buffer space. Specifically, we define packet-oriented versions of the Group by Virtual Output Queue algorithm of [2] and the Least Occupied Output First algorithm of [5] and show that both are work-conserving with a speedup of 2. Moreover, we show that order-preservation can also be established for both algorithms using a speedup of 2. Unlike cell-

based crossbar schedulers, our packet schedulers operate asynchronously. This has required the development of new methods for analyzing their performance and these methods now make it possible to evaluate asynchronous crossbars in a way that is directly comparable to synchronous crossbars.

The use of buffered crossbars is not new. An early ATM switch from Fujitsu used buffered crossbars, for example [7]. However, most systems use unbuffered crossbars, because the addition of buffers to each of the n^2 crosspoints in an $n \times n$ crossbar has been viewed as prohibitively expensive. A recent paper by Chuang et. al. [3] advocates the use of buffered crossbars in cell-based switches in order to reduce the complexity of the scheduling algorithms. The authors argue that ongoing improvements in electronics now make it feasible to add buffering to a crossbar, without requiring an increase in the number of integrated circuit components. Hence, the cost impact of adding buffering is no longer a serious obstacle. Our results add further weight to the case for buffered crossbars, as the use of buffering allows inputs and outputs to operate independently *and asynchronously*, allowing variable length packets to be handled directly.

Section 2 provides a more detailed discussion of the issue of switching cells vs. packets. Our main results are given in sections 3 and 4, where we establish work-conservation and order-preservation properties for packet versions of two crossbar scheduling algorithms. In section 5, we briefly study how the packet version of the Least Occupied Output First scheduler performs when operated with speedups smaller than 2. Section 6 concludes the paper with a discussion of alternate buffering strategies, some practical implementation issues and an outline of ways this work can be extended.

2. SWITCHING PACKETS VS. CELLS

As noted in the introduction, most crossbar-based routers, segment packets into cells at input line cards, before forwarding them through the crossbar to output line cards, where they are reassembled into packets. This enables synchronous operation, allowing the crossbar scheduler to make decisions involving all inputs and outputs at one time. Unfortunately, cell-based crossbars have some significant drawbacks. First, the segmentation of packets into cells can lead to degraded performance if the incoming packets cannot be efficiently packed into fixed length cells. In the worst-case, arriving packets may be just slightly too large to fit in a single cell, forcing the input line cards to forward them in two cells. This effectively doubles the bandwidth that the crossbar requires in order to handle worst-case traffic. Increasing the crossbar bandwidth by this amount can be unacceptably expensive, particularly in systems where the line cards are connected to the crossbar via inter-chassis optical links. While one can reduce the impact of this problem by allowing parts of more than one packet to occupy the same cell, this adds complexity and does nothing to improve performance in the worst-case. For this reason, the ability to forward variable length packets through the crossbar is highly desirable.

However, the trouble with cell-based crossbars is not just the bandwidth loss due to segmentation. Crossbar schedulers that operate on cells can interfere with the objective of work-conservation, by delaying the delivery of complete packets to outputs. In a system that uses a cell-based crossbar scheduler, an output line card cannot begin transmission of a packet on its outgoing link until all cells of the packet have been received. Consider a scenario in which n input line cards receive packets of length L, all addressed to the same output, at time t. If the length of the cell used by the crossbar is C, each packet must be segmented into $\lceil L/C \rceil$ cells for transmission through the fabric. A crossbar scheduler that operates on cells will typically forward cells from each input in a fair fashion, meaning that close to $n(\lceil L/C \rceil - 1)+1$ cells will pass through the crossbar before the output line card has a complete packet that it can forward on the output link. While some delay between the arrival of a packet and its transmission on the output link, is unavoidable, delays that are substantially longer than the time it takes to receive a packet on the link are clearly undesirable, if not intolerable. In the given scenario, a speedup of n is needed to bring the delay down to this level.

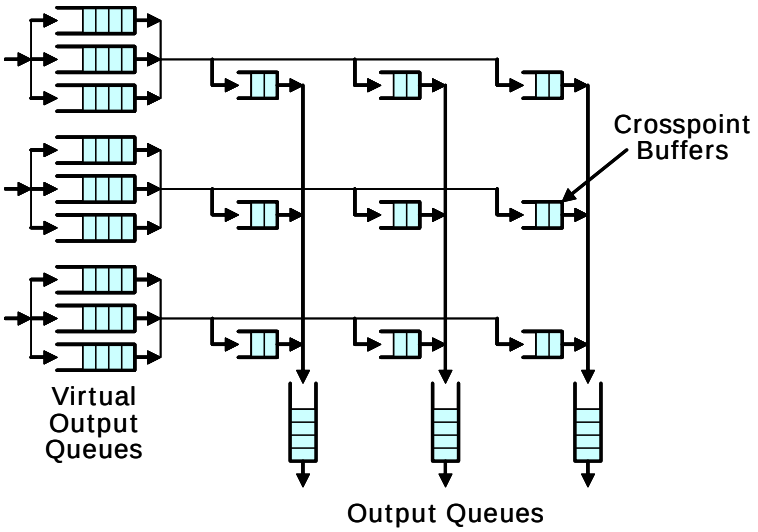

Figure 1. Packet switch using buffered crossbar

The basic problem with cell-based crossbar schedulers is their failure to consider which packets different cells belong to. There is no obvious solution to this problem in the context of unbuffered crossbars, since the ability of the scheduler to coordinate the movement of traffic through the system, seems to depend on its ability to make decisions involving all inputs and outputs at one time. A scheduler that operates on packets must deal with the asynchronous nature of packet arrivals, and must schedule packets as they arrive and as the inputs to the crossbar become available. In particular, if a given input line card finishes sending a packet to the crossbar at time t, it must then select a new packet to send to the crossbar. It may have packets that it can send to several different outputs, but its choice of output is necessarily limited to those outputs that are not currently receiving packets from other inputs. This can prevent it from choosing the output that it would prefer, were its choices not so constrained. One can conceivably ameliorate this situation by allowing an input to select an output that will become available in the near future, but this adds complication and sacrifices some of the crossbar bandwidth. Moreover, it is not clear that such a strategy can lead to a scheduling algorithm with good worst-case performance and small speedup.

The use of buffered crossbars offers a way out of this dilemma. The addition of buffers to each crosspoint of an $n \times n$ crossbar effectively decouples inputs from outputs, enabling the asynchronous operation that variable length packets seem to require. A diagram of a system using a buffered crossbar is shown in Figure 1. In addition to the now conventional *virtual output queues* at each input, and the single queue at each output, a buffered crossbar has a small buffer at each of its crosspoints. As pointed out in [3], the buffers allow inputs and outputs to operate independently, enabling the use of simpler crossbar scheduling mechanisms. But the buffers have an even greater import for packet-based scheduling, since they allow inputs and outputs to operate asynchronously. Specifically, whenever an input finishes sending a packet to the crossbar, it can select a packet from one of its virtual output queues, so long as the corresponding crosspoint buffer has room for the packet. We show in the next section that if the crosspoint buffers are large enough to hold two maximum length packets, we can achieve work-conservation with the same speedup required by cell-based schedulers.

3. WORK-CONSERVING SCHEDULERS

In this section, we show that two well-known cell-switching schedulers can be converted into packet-switching schedulers that provide comparable performance guarantees.

3.1 Preliminaries

To start, we introduce common notations that will be used in the analysis to follow. We say a packet x is an ij-packet if it is present at input i and is to be forwarded to output j. We let $s(x)$ denote the time at which the first bit of x is received on an input link and we let $f(x)$ be the time at which the last bit is received. We let $L(x)$ denote the number of bits in x and L_M denote the maximum packet length (in bits). The time

unit is the time it takes for a single bit to be transferred on an external link, so $f(x)–s(x) = L(x)$. For a packet x at the front of its VOQ, we let $R(x,t)$ denote the number of bits not yet sent to the crossbar at time t. The time at which a new packet is selected by an input and sent to the crossbar is referred to as a *scheduling event* or more simply, an *event*. We let $V_{i,j}$ denote the virtual output queue at input i that contains packets for output j and we let $V_{i,j}(t)$ denote the number of bits in $V_{i,j}$ at time t. Similarly, we let $B_{i,j}$ denote the crosspoint buffer for packets from input i to output j, $B_{i,j}(t)$ denote the number of bits in $B_{i,j}$ at time t, and B denote the capacity of the crosspoint buffers. We also, let Q_j denote the buffer at output j and let $Q_j(t)$ denote the number of bits it contains at time t. For all the quantities that include a time parameter, we sometimes omit the time parameter when its value can be understood from the context.

We say that a given packet switch scheduler is *T-work-conserving* for a given speedup S and crosspoint buffer size B, if whenever there is an idle output link, no input contains a packet x for that output link for which $s(x)+T$ is less than the current time. That is, a work conserving scheduler allows an output link to be idle only so long as there are no packets present for that output that are "older" than T.

The schedulers we focus on here are designed for systems in which packets are fully buffered at the input line cards where they arrive before they are sent to the crossbar. We say that a VOQ is *active*, if the last bit of the first packet in the VOQ has been received from the external link. Otherwise, it is *inactive*. Note that a VOQ can become inactive, even while it remains non-empty. For an active VOQ, we refer to the time period since it last became active as the *current active period* and for any packet x in a VOQ, we let $\tau_A(x)$ denote the time of the first event in the current active period. Once a packet has been selected by an input line card for transmission to the crossbar, it is sent at the rate allowed by the system's speedup S. Similarly, once an output line card selects a packet from a crosspoint buffer, it transfers bits from the crosspoint buffer at the rate allowed by the speedup, until the packet is fully transferred. Packets may be streamed through the crossbar buffer without fully buffering them and may be forwarded by an output line card to the external link as soon as the first bit is received by the output line card. Since the speedup is at least 1, the output line card is guaranteed that once it receives the first bit, the remaining bits will arrive in time to be sent on the outgoing link. (Note, this property is not shared by systems that use cell-based schedulers, forcing those systems to wait until the last cell of a packet has been received before starting transmission of a packet on the outgoing link.) In section 6 we discuss how similar results can be obtained for systems that place different conditions on where packets are buffered.

We consider only schedulers that keep the inputs and outputs busy whenever possible. In particular, if an input line card has any packet x at the head of one of its VOQs that is complete (all bits received from the input link) and the crosspoint buffer for x has room for it, then the input must be transferring bits to some crosspoint buffer at rate S. Similarly, if any crosspoint buffer for output j is not empty, then output j must be transferring bits from some crosspoint buffer at rate S. A scheduler that satisfies these properties is called a *prompt scheduler*. Prompt schedulers have the following useful property

Lemma 1. For a prompt scheduler,

$$B_{i,j}(t) \le (1+1/(S–1))Q_j(t)$$

for all values of i, j and t. In particular, if $S \ge 2$, $B_{i,j}(t) \le 2Q_j(t)$.

proof. For a prompt scheduler, whenever any of an output's crosspoint buffers is non-empty, the length of the output buffer increases at rate S–1. Since a crosspoint buffer can increase at rate no more than S, the length of the crosspoint buffer can be at most $S/(S–1) = (1+1/(S–1))$ times larger than the length of the output buffer. ■

3.2 Packet GVOQ

Group by Virtual Output Queue (GVOQ) is a cell switch scheduling algorithm first described in [2] and extended to buffered crossbars in [3]. We define the *Packet GVOQ* (PGV) packet switch scheduling algorithm as follows. The algorithm imposes a total order on the active VOQs at each input. This order is extended to an order on all the packets in the active VOQs (packets in the same VOQ are ordered by their position in the VOQ, while packets in different VOQs are ordered according to their VOQs). We say that one packet *precedes* another if it comes before the other in this ordering. The relative order of two VOQs does not change so long as they both remain non-empty. When a VOQ becomes active, it is placed first

in the VOQ ordering. When a VOQ becomes inactive, it is removed from the VOQ ordering. At each event, the PGV algorithm selects some VOQ for which the crosspoint buffer has enough space to accommodate the first packet in the VOQ. If multiple VOQs are eligible under this criterion, it selects the VOQ that comes first in the ordering. The work-conservation result we prove below does not depend on the specific policy used by the output line card to select a crosspoint buffer, so we leave the output policy undefined here. Hence, we are actually defining a class of PGV schedulers with a variety of specific instantiations.

For a packet x at an input, we let $p(x,t)$ = the number of bits in x's VOQ plus the number of bits in packets whose VOQs became *non-empty* after x's VOQ last became active. Note that $p(x,t)$ may include bits from packets that are in the process of arriving on the input link but have not yet been fully received. Also, note that any VOQ that became non-empty after x last became active, is either not yet active or became active after x's VOQ. Hence, $p(x)$ is an upper bound on the number of bits in the packets that precede x in the packet ordering (including x itself). So if $p(x,t)=L(x)$ and no bits of x have been sent to the crossbar yet, then x is the first packet in the ordering at input i at time t. More generally, if $p(x,t)=R(x,t)$, then x is the first packet in the ordering at input i at time t. We define $q(x,t)$ to be the number of bits in the output that x is going to, at time t and we define $slack(x,t) = q(x,t) - p(x,t)$.

Now, we show that PGV is work-conserving when operated with a speedup of 2. Our proof requires several lemmas. The first is similar to lemmas used in the work-conservation proofs for cell-switching algorithms.

Lemma 2. Let x be an ij-packet at input i at times t_1 and $t_2>t_1$, where $t_1 \geq f(x)$ and $t_1 \geq \tau_A(x)$. For any PGV scheduler with speedup S and $B \geq 2L_M$,

$$slack(x,t_2) \geq slack(x,t_1)+(S-2)(t_2-t_1).$$

That is, slack increases at rate at least S–2. Hence, for $S \geq 2$, *slack* does not decrease for any packet in a VOQ after the first event of the current active period.

proof. First, note that during any time period, $p(x)$ can increase by at most the duration of the time period, due to new bits arriving at input x at the link rate. Similarly, $q(x)$ can be decrease by at most the duration of the time period, as bits are sent on the outgoing link at the link rate. The resulting decrease in *slack* can be offset by increases caused by the transfer of bits from input i to the crossbar or from the crossbar to output j.

Now, consider what happens between any two consecutive events at input i. Let w be the first packet in the VOQ containing x. If, at the time of the first event, $B_{i,j}$ does not have room for w, then it contains at least L_M bits (recall that the crosspoint buffer size is $2L_M$). This means that output j will be transferring bits from some crosspoint buffer for at least the next L_M/S time units at rate S, by which time the next event at input i must have occurred. On the other hand, if $B_{i,j}$ does have room for w at the time of the first event, then either w must be selected for transmission to the crossbar or some packet that precedes w must be selected. In either case, this removes bits preceding x at rate S throughout this time period. Consequently, whether $B_{i,j}$ has room for w or not, the transfer of bits to/from the crossbar contributes to an increase in $slack(x)$ at rate S, giving a net rate of increase of at least S–2. ■

Lemma 3. Let x_1 be the first packet in $V_{i,j}$ at the time t_1, of some event at input i, and let x_2 be the first packet in $V_{i,j}$ at some time $t_2>t_1$ in the same active period for $V_{i,j}$. For any PGV scheduler with speedup $S \geq 2$, if $slack(x_1,t_1) \geq -L(x_1)$, then $slack(x_2,t_2) \geq -R(x_2,t_2)$.

proof. Suppose that t_2 is the time of the next scheduling event at input i after t_1. If the event at t_1 does not select a packet from $V_{i,j}$, the inequality remains true at the time of the next event by Lemma 2. Suppose then, that the event at t_1 does select x_1 from $V_{i,j}$. Until the time of the next event, bits of x_1 are sent to the crossbar at rate S, which means that output j must also be receiving bits from the crossbar at rate S. This means that $slack(x_1)$ experiences a *net increase* of at least $L(x_1)$ during the time between the two events, if $S \geq 2$. So, at the time the last bit of x_1 is being sent to the crossbar, $slack(x_1) \geq 0$. Consequently,

$$slack(x_2,t_2) = slack(x_1,t_2) - L(x_2) \geq -L(x_2) = -R(x_2,t_2)$$

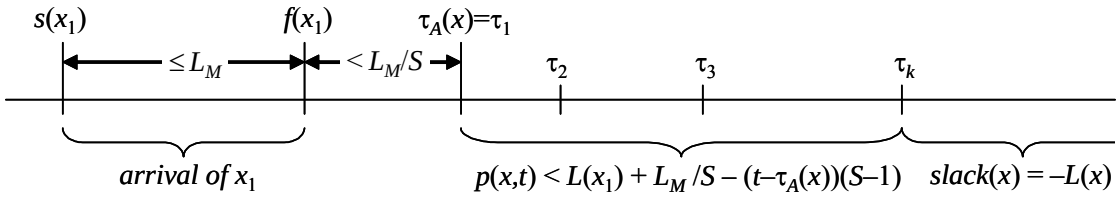

Figure 2. Diagram for Lemma 4.

It follows by induction that the result holds if t_2 is the time of any event later than t_1.

To complete the proof, we need to show that the condition on *slack* is maintained throughout the time interval between two consecutive events. If x is the packet at the head of the queue at the time t of some event, then x remains at the head of the queue between time t and the time of the next event. If x is not selected at time t, then the result follows by Lemma 2. If x is selected at time t, then during the time interval immediately following the event, $slack(x)$ increases at a rate $\geq 2S-2 \geq S$. Since the bits of x are also being transferred to the crossbar at rate S, the increase in slack is sufficient to keep up with the rate at which bits are leaving the input. Hence, $slack(x) \geq -R(x)$ holds throughout the period up to the next event. ■

Lemma 4. Let x be the first packet in $V_{i,j}$ at time t. For any PGV scheduler with speedup $S \geq 2$ and $B \geq 2L_M$, either $slack(x,t) \geq -R(x,t)$ or

$$p(x,t) < L(x_1) + L_M/S - (t-\tau_A(x))(S-1).$$

proof. Let x_1, x_2,… be the packets that arrive during the active period for $V_{i,j}$ that includes time t. Let $\tau_1=\tau_A(x)$ and let the time of subsequent events by labeled, τ_2, τ_3,… Note that if $slack(x,\tau_k) \geq -L(x)$ (where x is the first packet in $V_{i,j}$ at τ_k), then the condition on *slack* in the statement of the lemma holds at all later times by Lemma 3. Hence, it suffices to show that the condition on p in the statement of the lemma holds up until the first event where the condition on *slack* holds (see Figure 2). We show this, by induction on k. For $k=1$, note that x_1 is the first packet in $V_{i,j}$ at τ_1. If $slack(x_1,\tau_1) \geq -L(x_1)$, then we're done. Assume therefore, that $slack(x_1,\tau_1) < -L(x_1)$. Because $\tau_1 < f(x_1)+L_M/S$, and because $V_{i,j}$ became the first VOQ in the ordering at input i at time $s(x_1)$, $p(x_1,\tau_1) < L(x_1)+L_M/S$.

For $k>1$, let x' be the first packet in $V_{i,j}$ at time τ_{k-1}. If $slack(x',\tau_{k-1}) \geq -L(x')$, then we're done. Assume therefore, that $slack(x',\tau_{k-1}) < -L(x')$. By the induction hypothesis,

$$p(x',\tau_{k-1}) < L(x_1) + L_M/S - (\tau_{k-1}-\tau_1)(S-1)$$

Consequently,

$$\begin{aligned} q(x',\tau_{k-1}) &= slack(x',\tau_{k-1}) + p(x',\tau_{k-1}) \\ &< -L(x') + L(x_1) + L_M/S - (\tau_{k-1}-\tau_1)(S-1) \end{aligned}$$

If $x'=x_1$, then

$$q(x',\tau_{k-1}) < L_M/S \text{ and } B_{ij}(\tau_{k-1}) < L_M/(S-1) \leq L_M$$

where the second inequality follows from Lemma 1. If $x' \neq x_1$, then $(\tau_{k-1}-\tau_1) \geq L(x_1)/S$ and

$$\begin{aligned} q(x',\tau_{k-1}) &< -L(x') + L(x_1) + L_M/S - ((S-1)/S)L(x_1) \\ &= -L(x') + L(x_1)/S + L_M/S \\ &\leq 2L_M/S - L(x') \end{aligned}$$

and by Lemma 1,

$$\begin{aligned} B_{ij}(\tau_{k-1}) &< (1+1/(S-1))(2L_M/S - L(x')) \\ &< 2L_M - L(x') \end{aligned}$$

In both cases there is room for x' in $B_{i,j}$. Consequently, either x' is selected at τ_{k-1}, or some packet w that precedes x' is selected. Thus, at every event at which the condition on *slack* fails to hold, the packet selected at that event is selected either from $V_{i,j}$ or a VOQ that precedes $V_{i,j}$. This means that all bits sent to the crossbar between τ_1 and τ_k precede x. Thus,

$$p(x,\tau_k) < L(x_1) + L_M/S - (\tau_k-\tau_1)(S-1)$$

By the same reasoning, at any time t in the interval between τ_{k-1} and τ_k

$$p(x',t) < L(x_1) + L_M/S - (t-\tau_1)(S-1)$$

■

Lemma 5. Let x be the first packet in $V_{i,j}$ at time t and let $s(x) \leq t-T$ where $T=(1+1/(S-1))L_M$. If x has not yet been selected for transmission to the crossbar, then,

for any PGV scheduler with speedup $S \geq 2$ and $B \geq 2L_M$, $slack(x,t) \geq -L(x)$.

proof. By Lemma 4, either $slack(x,t) \geq -L(x)$ or

$$p(x,t) < L(x_1) + L_M/S - (t-\tau_A(x))(S-1)$$

In the first case, we're done, so assume $slack(x,t) < -L(x)$ and let x_1 be the first packet to arrive in the current active period. If $x=x_1$, then since

$$\begin{aligned} t &\geq s(x) + T \\ &> \tau_A(x) - (L(x_1) + L_M / S) + T \\ &\geq \tau_A(x) + L_M / S(S-1) \end{aligned}$$

it follows that $p(x,t) < L(x_1)$. That is, the number of bits that precede x at time t is less than the length of x, which contradicts the fact that x has not been selected yet.

Assume therefore, that $x \neq x_1$. In this case $s(x) \geq f(x_1)$, which means that $s(x) > \tau_A(x) - L_M/S$. Consequently,

$$\begin{aligned} t &> s(x) + T \\ &> \tau_A(x) - L_M / S + T \\ &\geq \tau_A(x) + L_M + L_M / S(S-1) \end{aligned}$$

Hence,

$$p(x,t) < L(x_1) + L_M / S - (L_M + L_M / S(S-1))(S-1) \leq 0$$

which is not possible. This yields a contradiction to the assumption that $slack(x,t) < -L(x)$. ■

It is straightforward to show that the bound on T in Lemma 5, cannot be improved significantly. It's also worth noting that a similar result can be shown with respect to $f(x)$ instead of $s(x)$. In this case, the time bound becomes $L_M/(S-1)$. We're now in a position to prove our first work-conservation result.

Theorem 1. Any PGV scheduler with $S \geq 2$ and $B \geq 2L_M$ is T-work-conserving, for any $T \geq (1+1/(S-1))L_M$. In particular, it is $(2L_M)$-work-conserving.

proof. Suppose some output j is idle at time t and no input is currently sending it a packet, but some input i has a packet x for output j with $s(x)+T<t$. Assume, without loss of generality, that x is the first packet in $V_{i,j}$ at time t. Then by Lemma 5, $slack(x,t) \geq -L(x)$. Since $q(x,t)=0$, this implies $p(x,t) \geq -L(x)$. In other words, the only bits that precede x are the bits in x itself. This means that input i can forward x to the crossbar at time t. Since the scheduler is prompt, this means that output j must be receiving bits from some input at time t contradicting the assumption that output j is idle at time t. ■

3.3 Packet LOOFA.

The *Least Occupied Output First Algorithm* (LOOFA) is a cell crossbar scheduling algorithm described in [5]. We define a packet crossbar scheduling algorithm based on LOOFA, called *Packet LOOFA* (PLF). Like PGV, PLF imposes a total order on the VOQs at each input, which is extended to an order on all the packets at the input. In PLF, one VOQ precedes another if its output contains fewer bits. At each scheduling event, the PLF scheduler selects some VOQ for which the crosspoint buffer has enough space to accommodate the first packet in the VOQ. If multiple VOQs are eligible under this criterion, it selects the VOQ that comes first in the ordering. The work-conservation results we prove below do not depend on the specific policy used by the output to select a crosspoint buffer.

The work-conservation result for PLF is comparable to that for PGV, but the required analysis is technically more difficult because the relative orders of VOQs can change. PLF is more responsive to changes in output queue lengths than PGV. While this has no effect on work-conservation when $S \geq 2$, it can yield better performance for smaller speedups.

To simplify the analysis to follow, we can view each time period $[t_1,t_2]$ between two consecutive scheduling events at an input, as consisting of three sub-intervals $[t_1,t_{1+}]$, $[t_{1+},t_{2-}]$ and $[t_{2-},t_2]$. We view the arrival of bits on inputs as occurring during the first sub-interval, the transfer of bits from inputs to outputs as occurring during the second sub-interval and the forwarding of bits on the outputs as occurring during the third sub-interval. Note then, that for any packet x at input i at both t_1 and t_2,

$$slack(x,t_{1+}) \geq slack(x,t_1) - (t_2 - t_1)$$

Also, if $slack(x,t_{2-}) \geq (t_2-t_1) - L(x)$, it follows that $q(x,t_{2-}) \geq (t_2-t_1)$ and consequently $slack(x,t_2) \geq -L(x)$. Our first lemma focuses on how the slack changes during the "middle" sub-interval. For an input i, let $minSlack(i,t)$ denote the minimum slack at time t among all the packets present at input i.

Lemma 6. Let times t_1 and t_2 be the times of consecutive scheduling events at input i. For any PLF scheduler with speedup S and $B \geq 2L_M$,

$$minSlack(i,t_{2-}) \geq minSlack(i,t_{1+}) + S(t_2 - t_1)$$

That is, *minSlack* increases at rate at least S.

proof. Let x be any ij-packet at input i at time t_{1+}, let $\Delta=(t_2-t_1)$ and let $slack(x,t_{1+})= minSlack(i,t_{1+})+ \sigma$. We will show that $slack(x)$ increases by at least $S\Delta- \sigma$ during the sub-interval $[t_{1+},t_{2-}]$. The lemma then follows directly. (Note that it is not sufficient to prove that the slack of a packet x that has minimum slack at the start of the sub-interval increases by $S\Delta$, since x may not have minimum slack at the end of the sub-interval.)

First, consider any packet y at input i that is not selected at t_1 and whose destination output receives fewer than $S\Delta$ bits during $[t_{1+},t_{2-}]$. Since $\Delta \le L_M/S$, the crosspoint buffers for this output must have contained fewer than L_M bits at t_1 meaning that the crossbar buffer for y must have contained fewer than L_M bits at t_1. Consequently, y was eligible for selection at t_1 and since it was not selected, the packet that was selected must have preceded y at t_1. So, for any packet y that remains at input i, either the output receives $S\Delta$ bits during the interval, or the input sends $S\Delta$ bits to the crossbar from a packet that precedes y at the start of the interval.

We say that a packet y at input i *passes* x, if at time t_{1+}, x precedes y and at time t_{2-}, y precedes x. If no packets pass x, the argument in the last paragraph implies that either $q(x)$ increases by $S\Delta$ or $p(x)$ decreases by $S\Delta$. Either way, $slack(c)$ increases by at least $S\Delta - \sigma$.

Assume then, that there are $r>0$ bits in packets that pass x and let y be the packet in the set of packets that pass x that comes latest in the packet ordering at t_{1+}. Let m be the number of bits received by output j during $[t_{1+},t_{2-}]$. Then,

$$q(x,t_{1+}) + m = q(x,t_{2-}) \ge q(y,t_{2-}) \ge q(y,t_{1+})$$

Let $k=p(y,t_{1+})-p(x,t_{1+})$. Now,

$$\begin{aligned} q(x,t_{1+}) - p(x,t_{1+}) &= minSlack(i,t_{1+}) + \sigma \\ &\le q(y,t_{1+}) - p(y,t_{1+}) + \sigma \\ &\le (q(x,t_{1+}) + m) - (p(x,t_{1+}) + k) + \sigma \end{aligned}$$

So $(m-k) \ge -\sigma$. Since y passes x, its output must receive fewer than m bits, so $S\Delta$ bits that precede it at t_{1+} must be forwarded. Of these at least $S\Delta-(k-r)$ must also precede x at t_{1+}. So,

$$p(x,t_{2-}) \le p(x,t_{1+}) + r - (S\Delta - (k-r)) \le p(x,t_{1+}) - S\Delta + k$$

Combining this, with $q(x,t_{2-})= q(x,t_{1+})+ m$ gives,

$$\begin{aligned} slack(x,t_{2-}) &= q(x,t_{2-}) - p(x,t_{2-}) \\ &\ge (q(x,t_{1+}) + m) - (p(x,t_{1+}) - S\Delta + k) \\ &\ge slack(x,t_{1+}) + S\Delta + (m-k) \\ &\ge slack(x,t_{1+}) + S\Delta - \sigma \end{aligned}$$

That is, $slack(x)$ increases by at least $S\Delta - \sigma$. ■

Lemma 6 has an important consequence that is captured in the following Corollary.

Corollary 1. Let x be a packet at input i which is not selected at the time t_1 of some scheduling event. For any PLF scheduler with speedup $S\ge 2$ and $B\ge 2L_M$, if the next scheduling event at input i occurs at t_2 and $slack(x,t_1)\ge -L(x)$, then $slack(x,t_2)\ge -L(x)$.

proof. By the discussion preceding Lemma 6 and Lemma 6 itself,

$$slack(x,t_{1+}) \ge slack(x,t_1) - (t_2 - t_1)$$

and

$$slack(x,t_{2-}) \ge slack(x,t_{1+}) + 2(t_2 - t_1)$$

Combining these two inequalities with $slack(x,t_1)\ge -L(x)$ gives $slack(x,t_{2-})\ge (t_2-t_1)-L(x)$. By the discussion preceding Lemma 6, this implies $slack(x,t_2)\ge -L(x)$. ■

Also note in the real system, rather than the artificial version we have adopted for the purposes of analysis, if $slack(x)\ge -L(x)$ at time t_1 then this remains true throughout the interval between t_1 and t_2.

Lemma 7. Let x_1 be the first packet in $V_{i,j}$ at the time t_1, of some event at input i, and let x_2 be the first packet in $V_{i,j}$ at some time $t_2>t_1$ in the same active period for $V_{i,j}$. For any PLF scheduler with speedup $S\ge 2$, if $slack(x_1,t_1)\ge -L(x_1)$, then $slack(x_2,t_2)\ge -R(x_2)$.

proof. Suppose that t_2 is the time of the next scheduling event at input i after t_1. If the event at t_1 does not select a packet from $V_{i,j}$, the inequality remains true at the time of the next event by Corollary 1. Suppose then, that the event at t_1 does select x_1 from $V_{i,j}$. Until the time of the next event, bits of x_1 are sent from the crossbar at rate S, which means that output j must also be receiving bits from the crossbar at rate S. This means that $slack(x_1)$ experiences a *net increase* of at least $L(x_1)$ during the time between the two events, if $S\ge 2$. So, at the time the last bit of x_1 is being sent to the crossbar, $slack(x_1)\ge 0$. Consequently,

$$slack(x_2,t_2) = slack(x_1,t_2) - L(x_2) \ge -L(x_2) = -R(,x_2)$$

It follows by induction that the result holds if t_2 is the time of any event later than t_1.

To complete the proof, we need to show that the condition on *slack* is maintained throughout the time interval between two consecutive events. If x is the packet at the head of the queue at the time t of some event, then x remains at the head of the queue between time t and the time of the next event. If x is not selected at time t, then the result follows the discussion following Corollary 1. If x is selected at time t, then during the time interval immediately following the event, $slack(x)$ increases at a rate $\geq 2S-2 \geq S$. Since the bits of x are also being transferred to the crossbar at rate S, the increase in slack is sufficient to keep up with the rate at which bits are leaving the input. Hence, $slack(x) \geq -R(x)$ holds throughout the period up to the next event. ■

Lemma 8. Let x be the first packet in $V_{i,j}$ at time t. For any PLF scheduler with speedup $S \geq 2$ and $B \geq 2L_M$, either

$$slack(x,t) \geq -R(x,t)$$

or

$$p(x,t) < L(x_1) + L_M/S - (t.-\tau_A(x))(S-1).$$

The proof of Lemma 8 is essentially the same as the proof of Lemma 4, with references to Lemma 3 replaced with reference to Lemma 7.

Lemma 9. Let x be the first packet in $V_{i,j}$ at time t and let $s(x) \leq t-T$ where $T=(1+1/(S-1))L_M$. If x has not yet been selected for transmission to the crossbar, then, for any PLF scheduler with speedup $S \geq 2$ and $B \geq 2L_M$, $slack(x,t) \geq -L(x)$.

The proof of Lemma 9 is essentially the same as the proof of Lemma 5, with references to Lemma 4 replaced with reference to Lemma 8. This leads directly to the work-conservation result for PLF.

Theorem 2. Any PLF scheduler with $S \geq 2$ and $B \geq 2L_M$ is T-work-conserving, for any $T \geq (1+1/S+1/S(S-1))L_M$. In particular, it is $(2L_M)$-work-conserving.

The proof is essentially the same as the proof of Theorem 1.

4. ORDER PRESERVING SCHEDULERS

The analysis of the previous section can be modified to show that variants of PGV and PLF are order-preserving, not just work-conserving. Corresponding results for unbuffered cell-based crossbars are proved in [2] and [10]. To convert the work-conserving schedulers to order-preserving schedulers, we assign to each packet x, a *timestamp* equal to $s(x)$. The outputs use the timestamps to select packets from crosspoint buffers, always selecting the oldest packet first. (We note that similar results can be proved for timestamps equal to $f(x)$.) The output line cards forward packets in the order of their timestamps and delay packets until their "age" exceeds a threshold equal to $(1+1/(S-1)+1/S)L_M$. We call the resulting versions of PGV and PLF, *PGV with Timestamps* (PGVT) and *PLF with Timestamps* (PLFT).

We define a packet crossbar scheduler to be *T-order-preserving* for a given speedup S and crosspoint buffer size B, if it is T-work-conserving and all packets are forwarded in the order of their timestamps. We show that PGV and PLF are T-work-conserving for $T = (1+1/(S-1)+1/S)L_M$.

To facilitate the analysis, we define a specific, implementation of the algorithms and phrase our analysis in terms of that implementation. This implementation is an artificial construct created entirely for the purpose of simplifying the analysis. It is not intended for practical use. In this implementation, we view each output line card as consisting of two stages. The first stage contains a time-ordered queue that forwards packets to the second stage in the order of their timestamps, once their age exceeds a threshold of $(1+1/(S-1))L_M$. Bits are forwarded from the first stage to the second stage at the external link rate. The second stage can be viewed as an *interruptible delay buffer* with a delay of L_M/S. In most situations, bits flow through the delay buffer in the order they enter, and flow out to the external link after a delay of L_M/S. However, we allow the flow of data through the delay buffer to be interrupted. In particular, if the first stage starts receiving bits from a packet that has a smaller timestamp than any packets currently passing through the delay buffer, the bits of the new packet will be inserted into the delay buffer ahead of all packets with smaller timestamps. Those bits in the delay buffer that are behind the insertion point are "stalled" while the insertion takes place. The insertion operation does not affect the bits of a packet that has already started flowing out on the external link.

We re-define $q(x)$ to be the number of bits in the first stage of the output that packet x is going to that belong to the same packet as x or that belong to packets with timestamps smaller than the timestamp for x

(for simplicity, we assume that all packets have unique timestamps). With this alteration, we define *slack* as before, relative to the re-defined q. We start with the analysis for PGVT, which uses a series of lemmas that parallel those of section 3.1

Lemma 10. Let x be an ij-packet at input i at times t_1 and $t_2>t_1$, where $t_1 \geq f(x)$ and $t_1 \geq \tau_A(x)$. For any PGVT scheduler with speedup S and $B \geq 2L_M$,

$$slack(x,t_2) \geq slack(x,t_1) + (S-2)(t_2-t_1)$$

That is, slack increases at rate at least S–2. Hence, for $S \geq 2$, *slack* does not decrease for any packet in a VOQ after the first event of the current active period.

proof. Consider what happens between any two consecutive events at input i. Let w be the first packet in the VOQ containing x. If, at the time of the first event, $B_{i,j}$ does not have room for w, then it contains more than L_M bits. Since packets are transferred to and from the crossbar at rate S, $B_{i,j}$ must have contained a packet at the time of the most recent scheduling event at output j. Since inputs forward packets from each VOQ in the order they are received, the first packet in $B_{i,j}$ at that time must have been "older" than x (had a smaller timestamp). Since outputs select packets from crosspoint buffers according to their timestamps, this means that the packet selected by output j at its last scheduling event was older than x. It also means that until the next event at input i, output j will be receiving bits from packets that are older than x at rate S, contributing to an increase in $q(x)$. On the other hand, if $B_{i,j}$ does have room for w at the time of the first event at input i, then either w must be selected for transmission to the crossbar or some packet that precedes w must be selected. In either case, this removes bits preceding x at rate S throughout this time period, contributing to a decrease in p. Consequently, whether $B_{i,j}$ has room for w or not, the transfer of bits to/from the crossbar contributes to an increase in $slack(x)$ at rate S, giving a net rate of increase of at least S–2. ■

Lemma 11. Let x_1 be the first packet in $V_{i,j}$ at the time t_1, of some event at input i, and let x_2 be the first packet in $V_{i,j}$ at some time $t_2>t_1$ in the same active period for $V_{i,j}$. For a PGVT scheduler with speedup $S \geq 2$, if $slack(x_1,t_1) \geq -L(x_1)$, then $slack(x_2,t_2) \geq -R(x_2)$.

The proof of Lemma 11 is essentially the same as that of Lemma 3. It simply must be re-interpreted in terms of the redefined q and using Lemma 10, in place of Lemma 2.

Lemma 12. Let x be the first packet in $V_{i,j}$ at time t. For a PGVT scheduler with speedup $S \geq 2$ and $B \geq 2L_M$, either $slack(x,t) \geq -R(x,t)$ or

$$p(x,t) < L(x_1) + L_M/S - (t-\tau_A(x))(S-1).$$

The proof of Lemma 12 is essentially the same as that of Lemma 4.

Lemma 13. Let x be the first packet in $V_{i,j}$ at time t and let $s(x) \leq t-T$ where $T=(1+1/(S-1))L_M$. If x has not yet been selected for transmission to the crossbar, then, for any PGVT scheduler with speedup $S \geq 2$ and $B \geq 2L_M$, $slack(x,t) \geq -L(x)$.

The proof of Lemma 13 is essentially the same as that of Lemma 5.

Theorem 3. A PGVT scheduler with $S \geq 2$ and $B \geq 2L_M$ is T-order-preserving for $T \geq (1+1/(S-1)+1/S)L_M$. In particular, it is $((5/2)L_M)$-order-preserving.

proof. Consider the portion of the system that includes only the first stage of each of the output line cards. If the first stage is modified so that it does not hold packets until their age exceeds the threshold of $(1+1/(S-1))L_M$, then this can be viewed as a complete system using the original PGV scheduling algorithm. Hence, it is $((1+1/(S-1))L_M)$-work-conserving by Theorem 1. Now note that this remains true, if the first stage holds packets back until their age exceeds the threshold of $(1+1/(S-1))L_M$. Since the output process of the second stage is just a time-shifted version of its input process, it follows that the entire system is T-work-conserving for $T=(1+1/(S-1)+1/S)L_M$.

Now suppose that at some time t, some output j is about to forward a packet z on the external link and the system contains a packet x for output j with a smaller timestamp than z. Assume, without loss of generality, that x is the packet satisfying these conditions with the smallest timestamp. Note that if any bits from x had reached the output line card by time t, then x would be forwarded instead of z. Therefore, we can assume that no bit of x has reached the output line card.

Since z is about to be forwarded on the external link at time t, either z entered the second stage at some time $t' \leq t-L_M/S$ or it entered later than $t-L_M/S$ and was inserted in front of some other packet with a larger timestamp. It's easy to show by induction that either z or some packet with a timestamp larger than z was entering the second stage at some time $t' \leq t-L_M/S$.

Such a packet could only have entered the second stage at a time later than $s(x)+(1+1/(S-1))L_M$. Hence, at some time t' in $[s(x)+(1+1/(S-1))L_M, t-L_M/S]$, there was a packet entering the second stage with a larger timestamp than x.

This implies that $q(x,t')=0$ and by Lemma 13, $slack(x,t') = -L(x)$. This implies that $p(x,t') = -L(x)$. Consequently, there are no bits at input i that precede x at time t', except the bits in x itself, meaning that x is eligible for transmission to the crossbar at t'. Once x enters the crossbar, it can be delayed only for the time it takes the packet currently entering the output line card to complete. This means that by t, x must be entering the line card, yielding a contradiction that establishes the theorem. ■

The corresponding result for PLFT can be proved using a similar analysis, which we omit. The analysis rests on a similar series of lemmas. Only the first of these is significantly different, and combines elements of Lemmas 6 and 10.

Theorem 4. A PLFT scheduler with $S\geq 2$ and $B\geq 2L_M$ is T-order-preserving for $T\geq(1+1/(S-1)+1/S)L_M$. In particular, it is $((5/2)L_M)$-order-preserving.

5. EFFECT OF LIMITED SPEEDUP

The analytical results of the last two sections show that the two crossbar scheduling algorithms studied perform well when the speedup is ≥ 2. However, they say nothing about how they can be expected to perform when the speedup is limited to less than 2. Since the cost of a crossbar scales up in proportion to its speedup, a designer might well choose to engineer a system with a smaller speedup, even though it means foregoing strong performance guarantees.

We adopt the simulation methodology used in references [8, 9], where a so-called stress test was used to probe the performance of a switch scheduler under demanding traffic conditions. We use a different traffic pattern than [8, 9]. In particular, we adapt a pattern used by Chuang et. al. in [2] as a worst-case example for their *Critical Cells First* scheduler. The traffic pattern involves n inputs and n phases. Input i receives packets during the first i phases, and no new traffic after that. In phase j, all of the arriving traffic is addressed to output j. Note that an ideal output-queued switch can complete transmission of all the packets within one packet time of the last packet's arrival. In fact, a crossbar operating with a speedup of 1 can also forward all the packets with no delay if the traffic pattern is known in advance. However, it is a difficult traffic pattern for a scheduler that must react to the arriving traffic with no fore-knowledge of what is to come.

Figure 3 shows an example of how the PLF algorithm performs on the test pattern. In this example, $n=6$ and the phase duration is equal to the time it takes for 50 packets to arrive on the input links. The packet length is fixed at 1000 bytes, the crosspoint buffer at 2000 bytes and the speedup is 1.2. The output line cards select the crosspoint buffer for which the corresponding input has the longest backlog of waiting packets.

The first chart shows the time history of the aggregated backlogs at the input line cards for each of the outputs (that is $V(+,j)$ denotes $V_{1,j} + \cdots + V_{6,j}$). The vertical line at time 300 is the time when an ideal output queued switch would be finished sending the last packet. In this case, the simulated switch completed the transmission of the last packet at about time 350, an *overshoot* of about 16%.

The second chart shows the time history of the aggregated backlogs at the crossbar buffers and the outputs. Most of the outputs have substantial backlogs until about time 285, when all the output-side backlogs are gone, even though the inputs still have data to be transferred.

The third chart shows the time history of the *miss fraction* of the six outputs, plus an average miss fraction. We say that an output is incurring a miss, if the output is idle, but there is a packet in the system that has been fully received. The miss fraction for a given time interval is the fraction of that interval during which misses are taking place. Note that we "charge" for a miss even if the packets that are present for the target output just arrived. Thus, a system may be T-work-conserving and still incur misses. For the specific test case illustrated, we observe short bursts of misses at the start of the fifth and sixth phases, then a longer period of misses from about time 285 to 330.

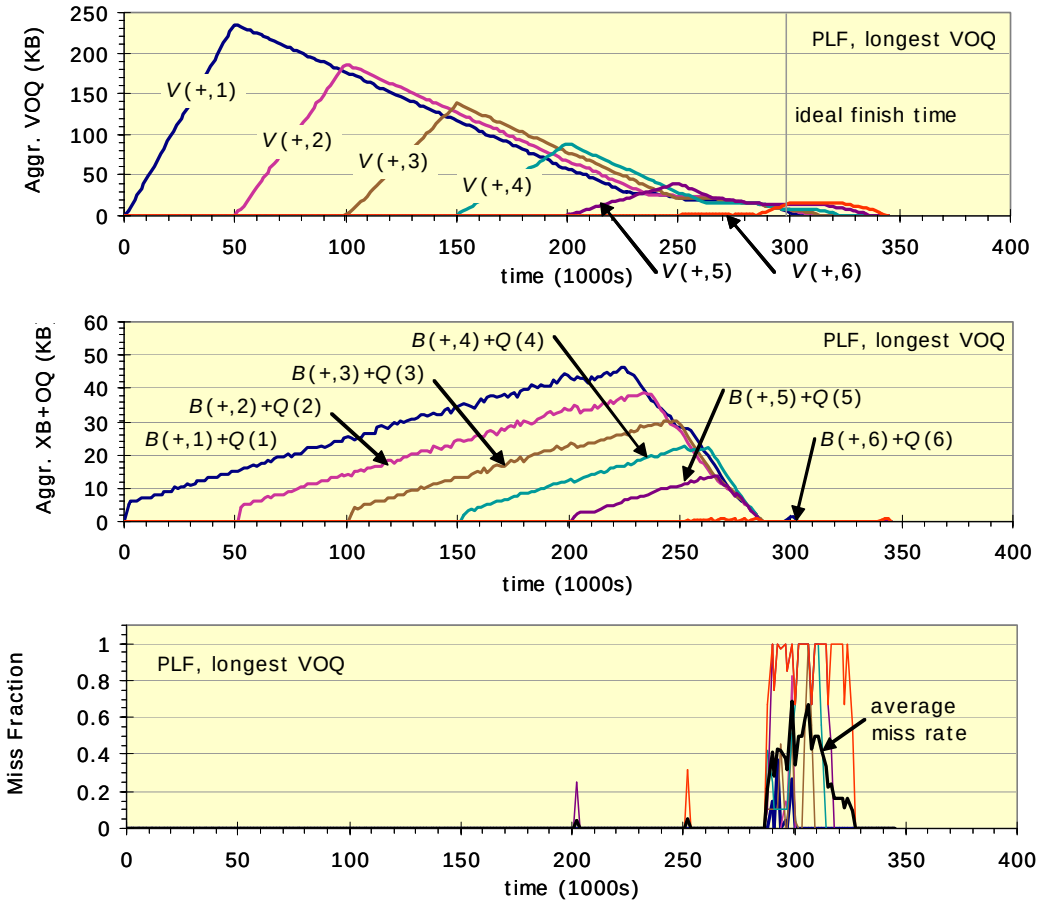

Figure 3. Sample Test Pattern Results

Figure 4 shows the results from a large number of similar tests. These cases involved larger switches (n=64) and phases long enough for 50 packets to arrive. The chart shows the overshoot for the test runs as a percentage, where the overshoot is the amount of extra time taken to forward the packets relative to an ideal, output-queued switch. For these results, the packet lengths have a bimodal distribution with peaks at the minimum and maximum packet lengths of 50 and 1000 bytes and an average of 200. Half of the total probability is in the two peaks, while the remainder is distributed uniformly across the intermediate packet lengths. The crossbar buffer size is 2000 bytes, twice the maximum packet length. The speedup is varied on the x-axis and the different curves show how the system performs for different output packet selection methods. In addition to the longest VOQ selection method mentioned earlier, we show results for random selection and selection based on timestamp value. We observe that the longest VOQ and timestamp selection methods provide virtually identical performance with respect to the overshoot metric, while the random selection method is clearly inferior.

Figure 5 shows a similar set of results in which the amount of buffer space in the crossbar buffers is varied. For these results, the longest VOQ selection method is used and the packet lengths have the same bimodal distribution as for the previous results. Reducing the crossbar buffer size from 2000 to 1000 bytes makes very little difference in the results, and while increasing the buffer size to 4000 yields some improvement, the difference is fairly small.

The results suggest that a designer might reasonably choose to implement a system with a speedup as low as 1.5.

6. CONCLUDING REMARKS

While our results have been derived for asynchronous crossbars that operate directly on variable length packets, they can also be applied to systems that seg-

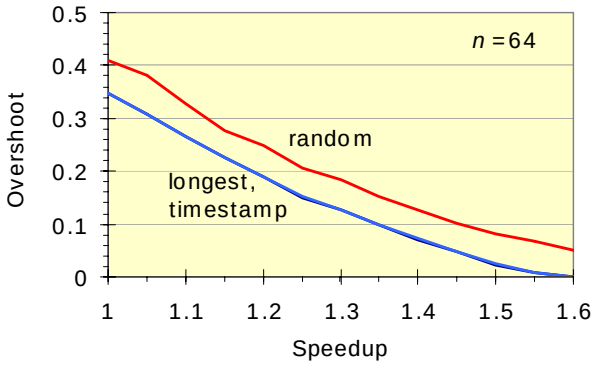

Figure 4. Results for different output packet selection criteria

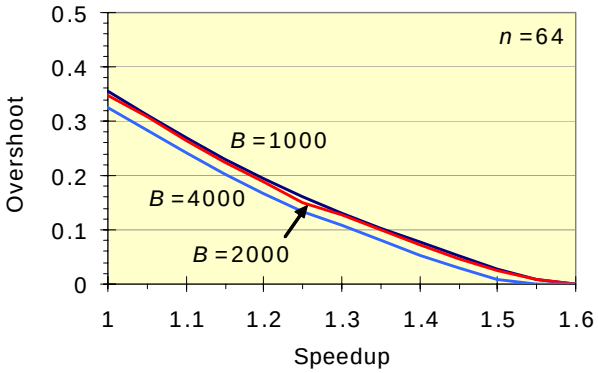

Figure 5. Results for different crossbar buffer sizes.

ment packets into cells, if the schedulers are modified to incorporate *hysteresis*. In such a scheduler, once a VOQ has been selected, the input must continue to supply cells from that VOQ until all cells in the packet have been transferred to the crossbar. Similarly, once an output has selected a particular crossbar buffer, it must continue to select that buffer until all cells of the packet have been transferred.

The results can also be extended to systems that place different constraints on where and when packets are buffered. In particular, most routers buffer packets at both inputs and output line cards, not just at the inputs. Buffering packets at the inputs allows error checks to be performed on the packets before forwarding them to the switch. Buffering them at the outputs allows similar checks to be performed, but is arguably less essential, since packet errors are less likely to occur within a router than on the external links. Having said that, other considerations may dictate that packets be buffered at outputs, as well as inputs and this raises the question of how the T-work-conservation and T-order-preservation properties are affected. It turns out that the effect is fairly minor, requiring only that the value of T be increased by L_M/S, to accommodate the added delay for a maximum length packet to be fully buffered at the outputs.

With an asynchronous crossbar, it is possible to build a system in which packets pass from inputs to outputs without ever being fully buffered. This is known as *cut-through switching* [4] and can provide superior delay performance. While cut-through switching is not typically used in routers, it can be useful in system contexts where it is important to minimize latency. While our results cannot be directly applied to such systems, it seems likely that similar results could be shown. The key requirement needed to obtain work-conservation is that once a packet has been selected to advance from an input line card to the crossbar or from the crossbar to an output line card, the flow of bits in that packet must not be interrupted until the end of the packet is reached. Inputs (outputs) must also be able to forward multiple packets to (from) the crossbar concurrently in certain cases. Consider for example, an input that is forwarding bits of a packet x to the crossbar as they come in. Since the bits are arriving at the link rate, the transfer of the bits of x to the crossbar uses only half the crossbar bandwidth (assuming S=2). If another packet y at the input becomes eligible for forwarding while x is still coming in (because its crossbar buffer has drained sufficiently to accommodate it), the input must be able to forward y to the crossbar concurrently with x in order to fully exploit the crossbar bandwidth. Without the ability to transfer packets concurrently to and from the crossbar, it will not be possible to achieve work-conservation.

One possible objection to the use of crosspoint buffers that are large enough to hold packets is that they are likely to be too expensive, even for modern integrated circuits. A 32 port crossbar equipped with buffers large enough to hold two 1500 byte packets would require a total of more than 3 MB of SRAM. While this is a substantial amount for on-chip memory, it falls within the range of what is currently feasible. Moreover, high performance crossbars are generally implemented using multiple components operating in parallel. The buffer space required by each such component is thus reduced in proportion to the number of components. For a system designed to support 40 Gb/s external links, a typical design might use 16

to 32 chips operating in parallel. This reduces the memory requirement per chip to about 100 to 200 KB. This is a fairly modest requirement and opens up the possibility of handling larger packets.

There are several ways the work described here can be extended. First work-conservation results can be developed for other scheduling algorithms, using the analysis techniques for asynchronous crossbars developed here. Another natural direction is to extend the analysis techniques to enable the establishment of stronger performance guarantees, such as delay bounds. Reference [3] shows how cell switches using buffered crossbars can provide such guarantees and it would be useful to develop similar guarantees for packet switches using buffered crossbars. Another interesting direction would be to establish work-conservation and order-preservation results for systems that use cut-through switching.

While we have argued that buffered crossbars seem to be needed to achieve work-conservation, it's conceivable that they are not. It would certainly be interesting to see if comparable results could be obtained without buffers in the crossbar. Alternatively, it may be possible to obtain similar results with buffers that are smaller than $2L_M$.

References

[1] Anderson, T., S. Owicki., J. Saxe and C. Thacker. "High speed switch scheduling for local area networks," *ACM Trans. on Computer Systems*, 11/93.

[2] Chuang, S.-T. A. Goel, N. McKeown, B. Prabhakar "Matching output queueing with a combined input output queued switch," *IEEE Journal on Selected Areas in Communications*, 12/99.

[3] Chuang, Shang-Tse, Sundar Iyer, Nick McKeown. "Practical Algorithms for Performance Guarantees in Buffered Crossbars," *Proceedings of IEEE INFOCOM*, 3/05.

[4] Kermani, Parviz and Leonard Kleinrock. "Virtual Cut-Through: A New Computer Communication Switching Technique." *Computer Networks* 3: 267-286, 1979.

[5] Krishna, P., N. Patel, A. Charny and R. Simcoe. "On the speedup required for work-conserving crossbar switches," *IEEE J. Selected Areas of Communications*, 6/99.

[6] McKeown , Nick. "iSLIP: a scheduling algorithm for input-queued switches," *IEEE Transactions on Networking*, 4/99.

[7] Nojima, S., E. Tsutsui, H. Fukuda, M.Hashimoto. "Integrated Services Packet Network Using Bus Matrix Switch", *IEEE Journal on Selected Areas of Communications*, 10/87.

[8] Pappu, P., J. Turner and K. Wong. "Stress-Resistant Scheduling Algorithms for CIOQ Switches, ," *Proceedings of ICNP*, 11/03.

[9] Pappu, Prashanth, Jonathan Turner and Ken Wong. "Work-Conserving Distributed Schedulers for Terabit Routers," *Proceedings of SIGCOMM*, 9/04.

[10] Rodeheffer, Thomas L. and James B. Saxe. "An Efficient Matching Algorithm for a High-Throughput, Low-Latency Data Switch ." Compaq Systems Research Center, Research Report 162, 11/5/98