Practical Algorithms for Performance Guarantees in Buffered Crossbars

Shang-Tse Chuang, Sundar Iyer, Nick McKeown
Computer Systems Laboratory,
Stanford University,
Stanford, CA 94305-9030.
{stchuang, sundaes, nickm}@stanford.edu

***Abstract*— Network operators would like high capacity routers that give guaranteed throughput, rate and delay guarantees. Because they want high capacity, the trend has been towards input queued or combined input and output queued (CIOQ) routers using crossbar switching fabrics. But these routers require impractically complex scheduling algorithms to provide the desired guarantees. In this paper, we explore how a buffered crossbar — a crossbar switch with a packet buffer at each crosspoint — can provide guaranteed performance (throughput, rate, and delay), with less complex, practical scheduling algorithms. We describe scheduling algorithms that operate in parallel on each input and output port, and hence are scalable. With these algorithms, buffered crossbars with a speedup of two can provide 100% throughput, rate, and delay guarantees.**

***Index Terms*— system design, combinatorics, packet switching, buffered crossbar, scheduling algorithm, performance guarantees, throughput, mimic, quality of service.**

I. Background

Network operators would like high capacity routers that give guaranteed performance. First, they prefer routers that guarantee throughput so they can maximize the utilization of their expensive long-haul links. Second, they want routers that can allocate to each flow a guaranteed rate. Third, they want the capability to control the delay for packets of individual flows for real-time applications. Because they want high capacity, the trend has been towards input queued or combined input and output queued (CIOQ) routers. Most of these routers use a crossbar switching fabric with a centralized scheduler. While it is theoretically possible to build crossbar schedulers that give 100% throughput [1] or rate and delay guarantees [2][3] they are considered too complex to be practical. No commercial backbone router today can make hard guarantees on throughput, rate or delay.

In practice, commercial systems use heuristics such as *i*SLIP [4] or a maximal matching algorithm such as WFA [5] with insufficient speedup to give guarantees. Perhaps the most promising way of obtaining guaranteed performance has been to use maximal matching with a speedup of two in the switch fabric [6]. But this only gives guaranteed throughput with no guarantees on rates and delay. And it still requires a centralized scheduler which doesn't scale with an increase in the number of ports due to the communication complexity. In this paper we'll see that the scheduler for a buffered crossbar is much simpler, and therefore more scalable, than for a traditional unbuffered crossbar.

A. The Buffered Crossbar

Figure 1 shows a 3×3 buffered crossbar, with line-rate R. To prevent head-of-line blocking, the inputs maintain virtual output queues (VOQs). Fixed length packets[1] wait in the VOQs to be transferred across the switch. Each crosspoint contains a buffer that can hold one cell. The buffer between input i and output j is $B(i,j)$; when the buffer holds a cell, $B(i,j)=1$, else $B(i,j)=0$.

Because the packets are all the same length, time is slotted, with a time slot equal to the time it takes for a cell to arrive on the external line. Internally, the switch runs faster than the external line, and the ratio between the two is the *speedup*. If the switch can remove S cells from each input and transfer S cells to each output in a time slot, then it has a speedup of S. Throughout this paper we will assume that $S=2$ or 3, and so the switch has output queues.

B. Why use Buffered Crossbars?

Buffered crossbars are interesting because they have simpler scheduling algorithms than an unbuffered cross-

[1]We will assume that variable length packets are processed internally as fixed length cells. This is common practice in high performance LAN switches and routers; variable length packets are segmented into cells as they arrive, carried across the switch as cells, and reassembled back into packets again before they depart. The size of the cell doesn't matter here, and is not necessarily equal in length to a 53-byte ATM cell.

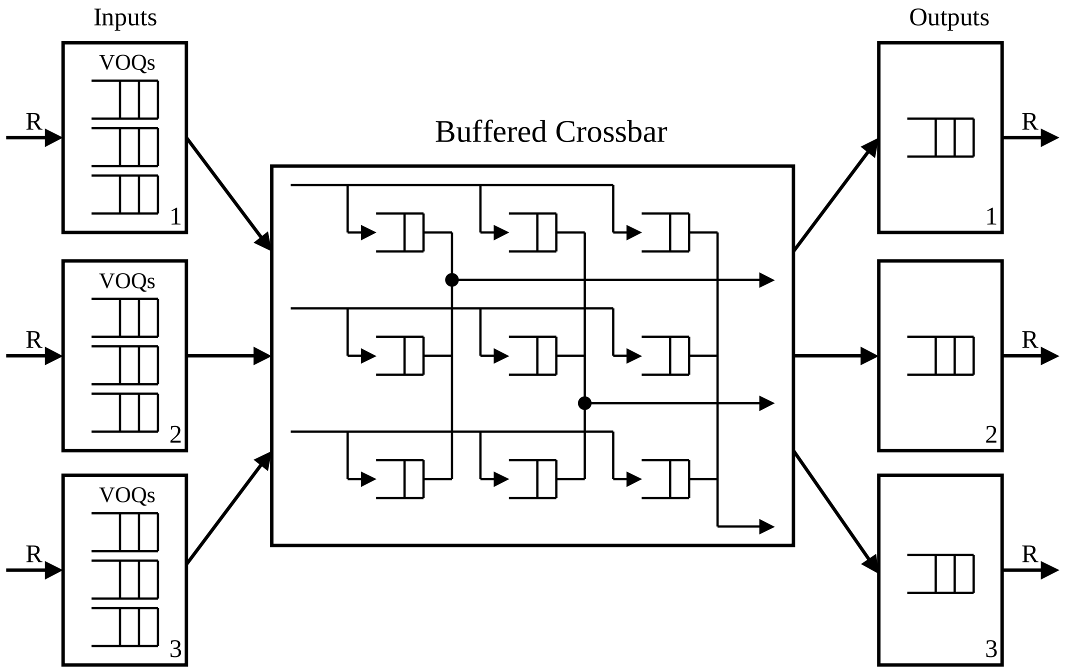

Fig. 1. The architecture of the buffered crossbar with three ports.

bar. In an unbuffered crossbar, the scheduler must find a matching between inputs and outputs that doesn't oversubscribe either. Overcoming both constraints at the same time leads to complex scheduling algorithms, such as maximum size [7] and maximum weight bipartite matchings [1], or iterative schedulers that are hard to pipeline [8, 4].

On the other hand, the scheduler for a buffered crossbar switch overcomes the two constraints in two independent stages. First, each input (independently and in parallel) picks a cell to place into a crosspoint buffer. And then in the second stage each output (independently and in parallel) picks a crosspoint buffer to take a cell from. The processing can be distributed to run on each input and output, and so no longer requires a single centralized scheduler. It can be pipelined to run at high speed, making buffered crossbars appealing for high performance switches and routers.

Researchers first noticed via simulation that buffered crossbars provide good throughput for admissible uniform traffic with simple algorithms [9, 10, 11]. Simulations also indicated that, with modest speedup, a buffered crossbar can closely approximate fair queueing [12]. Until recently, there were no analytical results to explain or confirm the observations made by simulations.

In [13] Javidi *et al.* proved that, with uniform traffic, a buffered crossbar can achieve 100% throughput. And then more recently, Magill *et al.* [14, 15] proved that a buffered crossbar with a speedup of two can mimic[2] a first come first serve output queued (FCFS-OQ) switch with any arrival traffic pattern. In this paper we extend Magill's work.

We describe a set of algorithms and prove that these algorithms can give 100% throughput, work-conservation and FCFS emulation (both of which can give rate guarantees when the input traffic is policed), and delay guarantees. The main benefit of these algorithms is that each input and output makes simple scheduling decisions *independently* and in *parallel* eliminating the need for a centralized scheduler. Our results show buffered crossbars can greatly simplify the scheduling process.

Of course, simplifying the scheduler comes at the expense of a more complicated crossbar; it now has to hold and maintain N^2 packet buffers. In the past this would have been prohibitively complex: The number of ports and capacity of a crossbar switch used to be limited by the N^2 crosspoints that dominated the chip area (hence the development of multi-stage switch fabrics, such as Clos, Banyan and Omega switches based on smaller crossbar elements). But nowadays, crossbar switches are limited by the number of pins required to get data on and off the chip [17]. Improvements in process technology, and reductions in geometries, means that the logic required for N^2 crosspoints is small

[2]Two different switches are said to mimic [16, 2] each other, if under identical inputs, identical packets depart from each switch at the same time.

compared to the size of chip needed for N inputs and outputs. The chips are pad-limited, with an underutilized die. A buffered crossbar can use the unused die for packet buffers. We believe that in current technology, the 128×128 unbuffered crossbar switch reported in [17] could hold 128^2 cell buffers.

C. Organization of this paper

The rest of the paper is organized as follows. In Sections III and V we show that a buffered crossbar can give 100% throughput and rate guarantees when $S = 2$. In Section VI, we show that the buffered crossbar can give delay guarantees when $S = 3$. In Section VI-B, we introduce a novel mechanism called crosspoint scheduling which supports delay guarantees when $S = 2$.

II. Preliminaries

Figure 2 shows the scheduling phases in a buffered crossbar with a speedup of two. The two scheduling phases each consists of two parts: input scheduling, and output scheduling.

We will adopt the notation and definitions introduced in [6]. The switch has N ports, and VOQ$_{ij}$ holds cells at input i destined for output j. Z_{ij} is the occupancy of VOQ$_{ij}$,[3] and $\tilde{Z_{ij}} = Z_{ij} + B_{ij}$ is the sum of the number of cells in the VOQ and the corresponding crosspoint buffer. $A_{ij}(n)$ is the cumulative number of arrivals to VOQ$_{ij}$ up until time n, and we'll assume that the arrival process obeys a strong law of large numbers:

$$\lim_{n\to\infty} \frac{A_{ij}(n)}{n} \quad = \quad \lambda_{ij}, a.s., \ \forall i, j. \tag{1}$$

where λ_{ij} is the arrival rate to VOQ$_{ij}$.

Similarly, $D_{ij}(n)$ is the cumulative number of departures from Z_{ij} up until time n. We'll say that the switch is *rate stable* if

$$w.p.1 \lim_{n\to\infty} \frac{D_{ij}(n)}{n} \quad = \quad \lambda_{ij}, a.s., \ \forall i, j. \tag{2}$$

A matching algorithm is said to give 100% throughput, (in [6] this is called efficient) if (2) holds for any arrival process satisfying (1) and the arrival traffic is admissible, i.e.

$$\sum_i \lambda_{ij} < 1, \sum_j \lambda_{ij} < 1. \tag{3}$$

[3]We'll see later that other queueing structures are useful and it is not necessary to place cells in VOQs.

III. Achieving 100% throughput with an arbitrary scheduling algorithm

In what follows, we will show that the buffered crossbar can give 100% throughput. The result is quite strong in the sense that it holds for any work-conserving scheduling algorithm with a speedup of two. In other words, each input i can choose to serve any non-empty VOQ for which $B(i, j) = 0$, and each output j can choose to serve any crosspoint buffer for which $B(i, j) = 1$.

Theorem 1: (Sufficiency) A buffered crossbar can achieve 100% throughput with speedup two for any admissible traffic.

Proof: In what follows, we describe an intuition of the proof. The main proof uses fluid models [6], and appears in Appendix A.

For each VOQ$_{ij}$, let C_{ij} denote the sum of the cells waiting at input i, and the cells destined to output j (including cells in the crosspoint buffers for output j),

$$C_{ij} = \sum_k Z_{ik} + \sum_k (Z_{kj} + B_{kj}). \tag{4}$$

If C_{ij} is stable for all i and j, then the switch is stable. It is easy to see that when VOQ$_{ij}$ is non-empty (i.e. $Z_{ij} > 0$), then C_{ij} decreases in every scheduling phase. There are two cases:

- **Case 1:** $B_{ij} = 1$. Output j will receive one cell from the buffers destined to it and $\sum_k (Z_{kj} + B_{kj})$, will decrease by one.
- **Case 2:** $B_{ij} = 0$. Input i will send one cell from its VOQs to a crosspoint buffer, and $\sum_k Z_{ik}$ will decrease by one.

With $S = 2$, C_{ij} will decrease by two per time slot. When the inputs and outputs are not oversubscribed, the expected increase in C_{ij} is strictly less than two per time slot, and so C_{ij} has bounded expectation. This means that Z_{ij} has bounded expectation, $D_{ij} \geq \lambda_{ij}$ and the buffered crossbar has 100% throughput. ■

IV. Making sure cells depart on time

In the next few sections we'll describe how a buffered crossbar can make rate guarantees and delay guarantees. We'll repeatedly use the idea that a cell must depart by a specific time. This means that a cell must be transfered to its output in order to leave on time. If the cell is prevented from reaching its output in time, it will miss its deadline, and we'll fail to meet the guarantee.

A cell can be prevented from reaching its output on time by other cells at its input. The more cells that are ahead of it, the longer it will take to be transferred to the output. The closer we get to its departure time, the more

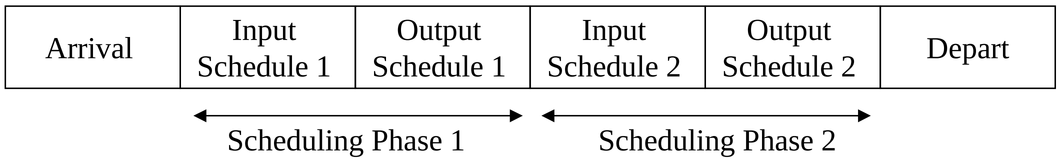

Fig. 2. The scheduling phases for the buffered crossbar. The exact order of the phases doesn't matter; but we will use this order to simplify proofs.

urgently it needs to be transferred. We'll capture the notion of urgency with the term "margin". The margin is a measure of how soon the cell needs to be transferred to its output. The more cells there are at the output, the larger the margin, and the less urgent the transfer. The cells at the output provide a cushion that will keep the output busy until we are able to transfer the cell. Similarly, the margin is smaller if there are more cells at the input that prevent the cell from being transferred. This leads to the following definitions:

Definition 1: **Input Margin —**Each input maintains a priority list of cells waiting to be transferred. Unless the crosspoint buffer prevents them from leaving, cells will depart according to their priority order. Consider a cell c queued at the input. Cell c's input margin, $IM(c)$, is the number of cells at its input with higher priority.

Definition 2: **Output Margin —**Cell c's output margin, $OM(c)$, is the number of cells at output j that will leave before cell c, plus the number of cells destined to output j that are queued ahead of cell c at the same input.

Definition 3: **Total Margin —**$TM(c) = OM(c) - IM(c)$.

The total margin reflects the urgency with which we must transfer the cell to its output. Our approach will be to find scheduling algorithms for which the total margin is always non-negative.

Although not strictly necessary, this will ensure that when a cell is transferred to the output its output margin is greater than zero, (or reaches zero in the time slot it is transferred). The idea is that when a cell's output margin reaches zero, the cell's input margin must also equal zero. This means either: (1) that the cell is already at its output, and will depart the output on time, or (2) that the cell is at the head of its priority list (because its input margin is zero), and will be transferred to the output immediately, which ensures that the cell will depart the output on time.

V. Achieving rate guarantees with a Buffered Crossbar

A router can give rate guarantees if: (1) Input traffic is policed so that mis-behaving flows can not adversely affect other flows, and (2) The outputs are *work-conserving*, i.e. an output is busy whenever there is a cell in the system for it.[4] In this section, we will assume that inputs to the router are policed, and show that a buffered crossbar scheduler can be work-conserving with $S = 2$, and hence can provide rate guarantees.

We will assume that the inputs use the following insertion policy [5] to decide which cells to send into the crosspoint buffers:

1) When a cell arrives to an empty VOQ, the VOQ is moved to the head of the VOQ priority list.
2) Each input picks a cell for which its VOQ is non-empty, and its crosspoint buffer is empty. The input gives preference to cells based on the VOQ priority list.

The output scheduling policy is the same as before, i.e. each output picks any cell (independently and in parallel) from among the non-empty crosspoint buffers.

A. Work conservation

The following lemmas lead to a proof that the buffered crossbar is work conserving with a speedup of two.

Lemma 1: If c is on the input side, $TM(c)$ is non-decreasing from time slot to time slot.

Proof: Based on the above insertion policy, the input margin for any cell c, is the sum of the cells that belong to VOQs with higher priority than VOQ$_{ij}$, plus those queued ahead of cell c in its VOQ. Also, since the order of departures of cells from an output does not matter for work conservation, the output margin for any cell c is the number of cells waiting at output queue j, plus the number of cells queued ahead of cell c in its VOQ. Assume $TM(c) \geq 0$ at the beginning of a time slot. During the arrival phase, $IM(c)$ can increase by at most one because an arriving cell might be inserted ahead of c. During the departure phase, $OM(c)$ will decrease by one, so $TM(c)$ can decrease by at most two in a single time slot.

[4]If a packet switch is work-conserving it gives 100% throughput, and the average delay faced by the cells is minimized.

[5]This insertion policy was previously described in [2], and was called "group by virtual output queues" (GBVOQ).

Now consider what happens during the two scheduling phases. Consider two cases.

- **Case 1:** If $B_{ij} = 0$, then we know that in the input scheduling phase, a cell will be transferred from input i to one of the buffers B_{i*}. If cell c is transferred to $B(i,j)$, then we no longer need to consider it. If a different cell is transferred to its crosspoint buffer, the cell would belong to c's input margin, and $IM(c)$ will decrease by one.
- **Case 2:** If $B_{ij} = 1$, then a cell will be transferred from one of the crosspoint buffers B_{*j} to output j by the output scheduler. So $OM(c)$ will increase by one.

Therefore, $TM(c)$ increases by at least one per scheduling phase. With two scheduling phases per time slot, $TM(c)$ increases by at least two. Taking into account arrivals, departures and both scheduling phases, $TM(c)$ can not decrease from time slot to time slot. ■

Lemma 2: The total margin of a newly arriving cell is non-negative.

Proof: From Lemma 1, we know that if $TM(c) < 0$ then its total margin must have been negative when it arrived. Let t be the first time that an arriving cell has negative total margin. Consider two cases.

- **Case 1**: If the cell was inserted at the head of the priority list, its input margin is zero. Since the output margin is always a non-negative value, the total margin of the cell is non-negative, which contradicts our assumption.
- **Case 2**: If the cell was not inserted at the head of the priority list, it must be inserted immediately behind another cell, c', destined to the same output as cell c. Since c' was inserted before t, it must have been inserted with non-negative total margin. By Lemma 1, its total margin is still non-negative at time t. But since $IM(c) = IM(c') + 1$, and $OM(c) \geq OM(c') + 1$, then $TM(c) \geq TM(c')$. So the total margin of cell c must also be non-negative, which again contradicts our assumption.

■

Theorem 2: (Sufficiency) A buffered crossbar can be work conserving with speedup two, regardless of the incoming traffic pattern.

Proof: Assume the buffered crossbar is not work conserving at some time t. This means that there is an output j whose output queues are empty, all the buffers B_{*j} are empty after the input scheduling phases, and there is a cell c destined to output j queued at the head of the VOQ of some input i. Since cell c was not transferred to B_{ij} at time t, $IM(c) > 0$. Since there are no cells in the output queues of output j, and cell c is the HOL cell, $OM(c) = 0$. So the total margin of cell c is negative at time t. However, Lemma 1 and Lemma 2 imply that total margin can never be negative, thus contradicting our assumption. Therefore the buffered crossbar is work conserving. ■

B. Mimicking a FCFS-OQ Switch

In a FCFS-OQ switch,[6] packets depart in the same order they arrived. This is a tighter requirement than work-conservation: a work-conserving router can send packets in any order so long as outputs are never unnecessarily idle. The departure rates are the same, and the outputs are busy at exactly the same time in both switches, but in any time slot, different cells can depart.

Maintaining true FCFS departure order is more onerous for the switch as it no longer has the option to send any cell; it must send a particular cell (or at least one of up to N cells that could have arrived at the same time).

Theorem 3: (Sufficiency) A buffered crossbar can mimic a FCFS-OQ switch with speedup two, regardless of the incoming traffic pattern.

Proof: This result was first proved in [14].

In a FCFS-OQ switch, the departure time of a cell is known when it arrives; it is simply the current time plus number of cells in the switch destined to the same output that were there when it arrived. We'll assume that every cell has a timestamp indicating the time it arrived. So that our buffered crossbar can mimic a FCFS-OQ switch, make $OM(c)$ be the number of cells currently waiting at output queue j that arrived earlier than cell c, plus the number of cells queued ahead of cell c in VOQ$_{ij}$. The input selects a VOQ to serve in the same manner as before. The output chooses a cell from the crosspoint buffer with the lowest timestamp (ties can be broken any way). ■

VI. Delay Guarantees in a Buffered Crossbar

In an output queued (OQ) router, delay guarantees are provided by a weighted fair queueing (WFQ) scheduler [18, 19]. Flows are shaped by, for example, a leaky-bucket source. Arriving packets are segmented into cells and placed into a per-flow queue; the queue is served by the WFQ scheduler so that each packet departs within a pre-agreed delay.

In order to abstract the fair queueing scheduler, we use a Push-In First-Out (PIFO) queue. We introduced it in [2] to help prove properties of crossbar switches. We'll use it again here. PIFO is a generalization of a

[6]Here output-queued switch includes switches based on centralized shared memory.

WFQ scheduler, and includes a class of work-conserving schedulers that includes WFQ, and strict priorities. If an unbuffered crossbar switch can emulate a PIFO output queued switch, then it can emulate a WFQ output queued switch too.

A PIFO queue is defined as follows. There is a single queue of all cells waiting to depart from an output. When a new cell arrives for the output, it is "pushed-in" to an arbitrary location in the queue. Once in the queue, the cell's relative ordering with cells already in the queue doesn't change; cells can't switch places. Of course, new cells can arrive later and push-in ahead of (or behind) the cell. Cells can only depart from the head of line.

Essentially, when a cell arrives, the WFQ scheduler picks its departure order, relative to other cells already in the switch. It could, for example, be scheduled to depart immediately, and ahead of all currently queued cells. This is going to cause problems for the buffered crossbar switch. Imagine the situation in which a crosspoint buffer is non-empty, and a new cell arrives that needs to leave *before* the cell in the crosspoint buffer. In the basic architecture there is no way for the new cell to overtake the cell in the crosspoint buffer; and so we say there is *"crosspoint blocking"*.

We could overcome crosspoint blocking by allowing the more urgent cell to pass-by the cell in the crosspoint buffer. But this creates a scheduling problem: It means we no longer split the scheduling into independent input and output stages, and so the scheduler will become just as complex as the scheduler in an unbuffered crossbar. We would be back to where we started.

Instead, we will allow the input to replace the cell in the crosspoint buffer with a more urgent cell, swapping out the old one, and exchanging it for the new one. Logically, the cell that was previously in the crosspoint buffer is recalled to the input where it is treated like a newly arriving cell. The new cell is put into position in the crosspoint buffer, ready to be removed by the output scheduler in time to leave the switch before its deadline. We'll need extra speedup to give time for the new cell to be sent to the crosspoint buffer. We'll modify the arrival phase so that we can swap the cell in the crosspoint buffer with a newly arriving cell, and we'll say the switch has a speedup of three. We don't need extra speedup to retrieve the old cell; in practice, the input would keep a copy of cells currently in the crosspoint buffers, and would simply overwrite the old one.

A. Delay guarantees with speedup three

First, we'll now describe how the input selects which cell to send to the crosspoint buffer. Any insertion policy needs to meet two requirements. (1) To prevent crosspoint blocking, the cells from an input destined to a given output must be inserted based on their departure order. (2) Upon insertion, the total margin of a cell must be non-negative. This is required so that a cell may reach the output in time.

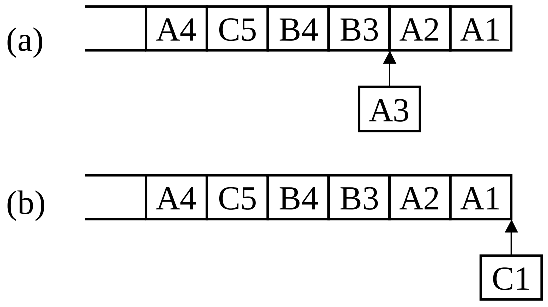

Fig. 3. The insertion policy for achieving delay guarantees. The figure shows the priority list for a given input. The letter denotes the output destination, and the number denotes the cell's departure order for a given output. (a) Arriving cell $A3$ is inserted immediately after cell $A2$. (b) Arriving cell $C1$ is inserted at the head of the priority list.

The Insertion Policy: As a consequence of the first requirement, an arriving cell c destined to output port j is inserted behind all cells destined to output j with a departure time less than cell c. To further satisfy the second requirement, cell c is inserted *immediately* behind the cell that departs before it (if it exists) destined to the same output. If no such cell exists, cell c is inserted at the head of the priority list. Similar to Lemma 2, this ensures that the total margin of the cell c is non-negative.

The priority list defined by this insertion policy, has the property that cells from input i to output j, are ordered based on their PIFO departure order. An example is shown in Figure 3.[7]

We are now ready to describe the input and output scheduling algorithms to emulate a PIFO output queued switch.

- **Input Scheduling.** An input picks the cell closest to the head of line which has an empty crosspoint buffer.
- **Output Scheduling.** An output picks the cell with the earliest departure time from a non-empty crosspoint buffer.

Theorem 4: (Sufficiency) A buffered crossbar can mimic a PIFO-OQ router (and hence give delay guarantees) with speedup three, regardless of the incoming

[7]This insertion policy should be contrasted with the CCF insertion policy in [2], which does not maintain cells from input i to output j in the correct departure order. CCF does not need to maintain this ordering because the stable marriage algorithm described in [2] considers all cells queued at the inputs while scheduling.

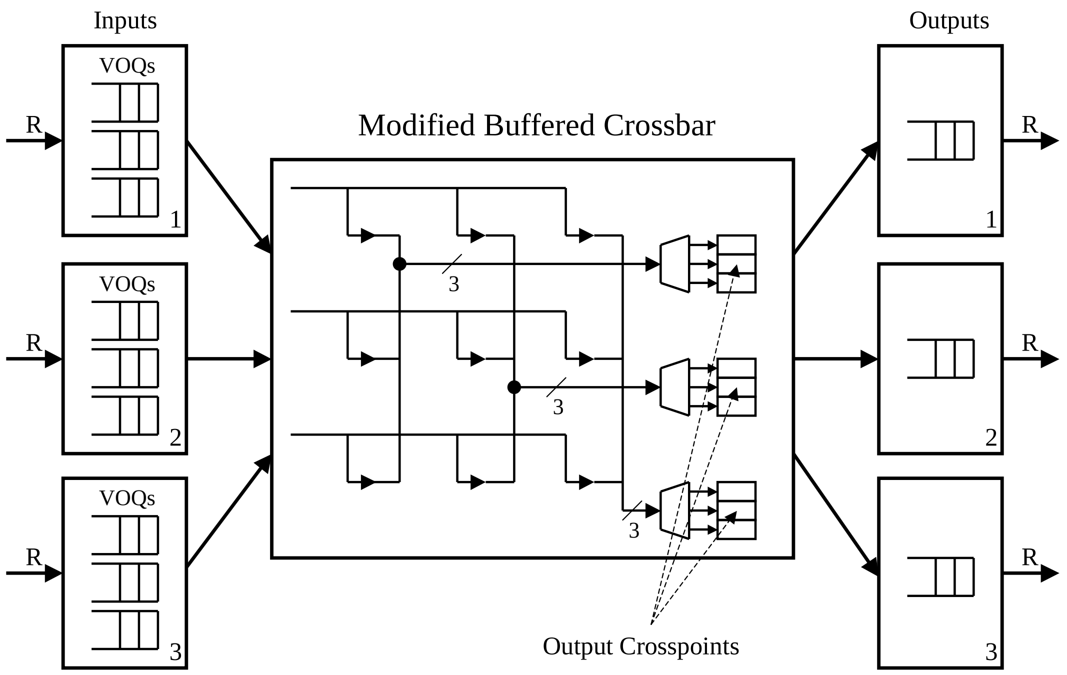

Fig. 4. The architecture of a modified buffered crossbar with three ports.

traffic pattern.

Proof: See Appendix B. ■

B. Delay guarantees with speedup two

It would be preferable to find a way to give delay guarantees with speedup two rather than three. In this section we'll use a scheme we call *crosspoint scheduling* to try and do so. The basic idea is this: Before, we needed extra speedup to replace an old cell in the crosspoint buffer with a newly arrived, more urgent cell. But if the cell in the crosspoint buffer was not put there in the first place, we wouldn't need to swap it. If instead of storing the cell in the crosspoint buffer we hold a "cell identifier" that contains the departure time, the output can still pick a cell according to the time it needs to leave. The cell is transferred later, once the output has made its decision. This means we only send cells through the switch when they really need to be transferred, and we don't need the extra speedup to overwrite cells in the crosspoints and recall cells to the input.

The process of scheduling and transferring cells would now take place in two distinct phases, just like in an unbuffered crossbar. First, scheduling would be done by inputs and outputs. The outputs would pick, or grant to, a cell identifier in the crosspoint. In the second phase, cells would be transferred according to the grants made by the outputs.

This doesn't quite solve the problem; as we'll see shortly, we also need to modify the way buffers are managed in the crossbar. The problem is that an input could receive N grants (one from each output) in a single scheduling phase. Over p consecutive phases the number of grants received by an input is bounded by $p+N-1$. This is because an input can place in the crosspoints at most one cell identifier per scheduling phase, and there are at most N outstanding cell identifiers. Each output chooses at most one cell identifier per scheduling phase, so there are at most p grants for an output over any p consecutive phases.

Because an input can only send two cells per time slot, an input granted in scheduling phase p might not be transferred until phase $p+N-1$. Worse still, this cell prevents the input from sending another cell to the same output for another N scheduling phases. We therefore need to modify the crossbar further to solve the problem and emulate a PIFO-OQ switch.

Instead of one buffer per crosspoint, there will now be N buffers per output as shown in Figure 4. There are still N^2 buffers in the crossbar, but buffers are dedicated to outputs, rather than input/output pairs.

Theorem 5: (Sufficiency) With speedup two and N cells of buffering per output, a buffered crossbar can mimic a PIFO-OQ switch with a fixed delay of $N/2$ time slots.

Proof: An input can receive at most $p + N - 1$ grants over any p consecutive scheduling phases. If the input adds new grants to the tail of a FIFO, and reads one grant from the head of the FIFO each scheduling phase, then the FIFO will never contain more than $N-1$ grants.

Each time the input takes a grant from the FIFO, it sends the corresponding cell to the set of N crosspoint buffers for its output. Because the grant FIFO is served once per phase, a cell that is granted at scheduling phase p will reach the output crosspoint buffer by phase $p + N - 1$.

We need to verify that the per-output buffers in the crossbar never overflow. If the crosspoint scheduler issues a grant at phase p, then the corresponding cell will reach the output crosspoint buffer between phases p and $p + N - 1$. Therefore, during scheduling phase p, the only cells which can be in the output crosspoint buffer are cells which were granted between phases $p-N$ to $p - 1$. With N buffers per output, the buffers will never overflow, and each cell faces a delay of at most N scheduling phases, or $N/2$ time slots (because $S = 2$). ■

In summary, a buffered crossbar that uses crosspoint scheduling and N buffers per output doesn't perfectly mimic a PIFO-OQ switch, but it still gives delay guarantees. Cells can be arranged to depart from the buffered crossbar precisely $N/2$ time slots later than they would from a PIFO-OQ switch. In both switches delay can be bounded and guaranteed.

The guarantee comes at the expense of a more complicated buffering scheme in the crossbar. Since each output crosspoint buffer can receive up to N cells per scheduling phase, there is additional implementation complexity. Alternatively, we could modify the original buffered crossbar so it has N cells for each $B(i, j)$, for a total of N^3 cells. While requiring much more storage, it will also mimic a PIFO-OQ switch with a fixed delay of $N/2$ time slots with speedup two directly using crosspoint scheduling. This might make sense for small values of N.

VII. Conclusions

Buffered crossbars seem to provide an easy evolutionary path for switches and routers that currently use unbuffered crossbars and a centralized scheduler. Whereas centralized schedulers get very complicated, a buffered crossbar can use a scheduler that makes decision independently and in parallel at each output. With a speedup of two, and simple scheduling algorithms, a buffered crossbar can easily provide 100% throughput and give rate guarantees. For many applications, this would be sufficient, and would be quite an improvement over what is currently available.

If delay guarantees are needed, we need to increase the speedup to three, or modify the buffering inside the crossbar. Although the crossbar is more complex than before, the speedup is still two, the bandwidth and pin count is the same as before, and no memory needs run faster than twice the line-rate. We do not need to sacrifice the simplicity of the scheduler; the scheduling algorithm still has the nice property of two separate, independent phases to schedule inputs and outputs, allowing high speed, pipelined designs.

References

[1] N. McKeown, A. Mekkittikul, V. Anantharam and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *IEEE Transactions on Communications*, Vol.47, No.8, Aug. 1999.

[2] S-T. Chuang, A. Goel, N. McKeown and B. Prabhakar: " Matching Output Queueing with a Combined Input Output Queued Switch", *IEEE J. Select. Areas Commun.*, **17**, no. 6, pp 1030-1039. A short version appears in *The Proceedings of Infocom '99*.

[3] S. Iyer, R. Zhang and N. McKeown , "Routers with a Single Stage of Buffering", *ACM SIGCOMM '02*, Pittsburgh, USA, Sep. 2002.

[4] N. McKeown, "iSLIP: A Scheduling Algorithm for Input-Queued Switches" *IEEE Transactions on Networking*, Vol 7, No.2, April 1999.

[5] Y. Tamir and H.C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 1, pp. 13-27, 1993.

[6] Dai J., Prabhakar B., "The throughput of data switches with and without speedup", *IEEE INFOCOM 2000*, vol. 2, Tel Aviv, Mar. 2000, pp. 556-564

[7] J. E. Hopcroft, R. M. Karp, "An $O(N^{5/2})$ Algorithm for Maximum Matching in Bipartite Graphs," *Society for Industrial and Applied Mathematics J. Computers*, vol. 2, pp. 225-231.

[8] T.E. Anderson, S.S. Owicki, J.B. Saxe, and C.P. Thacker, "High speed switch scheduling for local area networks," *ACM Transactions on Computer Systems*, Vol. 11, No. 4, pp. 319-352, Nov. 1993.

[9] Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao, "CIXB-1: Combined Input-One-cell-Crosspoint Buffered Switch," *IEEE Workshop on High Performance Switching and Routing*, Dallas, TX, July 2001.

[10] Rojas-Cessa, E. Oki, and H. J. Chao, "CIXOB-k: Combined Input-Crosspoint-Output Buffered Packet Switch," in *IEEE Globecom*, San Antonio, Texas, Nov. 2001.

[11] Lotfi Mhamdi, Mounir Hamdi, "MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches", IEEE Communications Letters, 2003.

[12] D. Stephens, H. Zhang, "Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture", in *Proc. INFOCOM* 1998.

[13] T. Javidi, R Magill, and T. Hrabik, "A High Throughput Scheduling Algorithm for a Buffered Crossbar Switch Fabric", in *Proc. IEEE International Conference on Communications*, vol. 5, pp. 1586-1591

[14] B. Magill, C. Rohrs, R. Stevenson, "Output Queued Switch Emulation by a Buffered Crossbar Fabric", in *Allerton conference on communication, computing and control*, UIUC, Urbana, Oct. 2002.

[15] B. Magill, C. Rohrs, R. Stevenson, "Output-Queued Switch Emulation by Fabrics With Limited Memory", in *IEEE Journal on Selected Areas in Communications*, pp.606-615, May. 2003.

[16] S. Iyer, N. McKeown , "Analysis of the Parallel Packet Switch Architecture", in *IEEE/ACM Transactions on Networking*, Vol. 11-2, pp. 314-324, April 2003.

[17] N. McKeown, C. Calamvokis, S. Chuang, "A 2.5Tb/s LCS Switch Core", Hot Chips, Stanford University, August 2001.

[18] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm", *ACM Computer Communication Review (SIGCOMM'89)*, pp. 3-12, 1989.

[19] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to fbw control in integrated services networks: The single node case", *IEEE/ACM Transaction on Networking*, Vol. 1, No. 3, pp. 344-357, June 1993.

[20] D. Shah, "Stable algorithms for Input Queued switches," *Proc. Allerton conference on communication, computing and control*, UIUC, Urbana, Oct. 2001.

APPENDIX A
ACHIEVING 100% THROUGHPUT FOR AN ARBITRARY SCHEDULING ALGORITHM

We first introduce the fluid models for switches described in [6]. We will use the fluid model for that the buffered crossbar can give 100% throughput.

A. Fluid Models

We first consider the dynamics of the discrete time switch. Let scheduling algorithm be $\mathcal{S}$. Recall that in the buffered crossbar, because the cells can be buffered in buffers $B(.,.)$, it is possible that in each scheduling slot, all the cells destined to one output could have arrived from all the inputs. We denote this as a $(1, N)$ semi-matching, i.e. an input can send no more than one cell, but an output may receive as many as N cells. Let $\prod$ be the collection of all possible $(1, N)$ semi-matchings of the buffered crossbar. In each scheduling slot, an input i, can get service to an output j, if $B(i, j) = 0$. Note that it is possible that an input i is not connected to any output if all the buffers $B(i, *)$ for that input are full. We will call this a null service. Thus each input can get one of $N + 1$ possible services. Note that, for an $N \times N$ buffered crossbar, $|\prod| = N^{N+1}$. For any $\pi \in \prod$, let $T^{\mathcal{S}}_{\pi}(n)$ be cumulative amount of time that the semi-matching is scheduled between time $[0, n]$ under scheduling algorithm $\mathcal{S}$. Again, $T^{\mathcal{S}}_{\pi}(0) = 0$ for all $\pi \in \prod$. The following equations hold for the switch: for $n \geq 0$ and $i, j = 1, \ldots, N$,

$$Z_{ij}(n) = Z_{ij}(0) + A_{ij}(n) - D_{ij}(n) \tag{5}$$

$$D_{ij}(n) = \sum_{\pi \in \prod} \sum_{\ell=1}^{n} \pi_{ij} 1_{Z_{ij}(\ell)>0}(T^{\mathcal{S}}_{\pi}(\ell) - T^{\mathcal{S}}_{\pi}(\ell - 1)) \tag{6}$$

$$T^{\mathcal{S}}_{\pi}(\cdot) \text{ is non-decreasing, } \sum_{\pi \in \prod} T^{\mathcal{S}}_{\pi}(n) = n \tag{7}$$

The first equation describes evolution of Z_{ij}, which is well known. The second equation keeps the counts of the total number of departures from Z_{ij}. And the third equation expresses the fact that each input i is connected (via a semi-matching) to some output j.

Similar to [6], we can write that under the condition of (1) the following is the deterministic, continuous fluid model of a switch. This fluid model obeys the following set of equations, $t \geq 0$, and $i, j = 1, \ldots, N$

$$Z_{ij}(t) = Z_{ij}(0) + \lambda_{ij} t - D_{ij}(t) \geq 0 \tag{8}$$

$$\dot{D}_{ij}(t) = \sum_{\pi \in \prod} \pi_{ij} \dot{T}^{\mathcal{S}}_{\pi}(t), \text{ if } Z_{ij}(t) > 0 \tag{9}$$

$$T^{\mathcal{S}}_{\pi}(\cdot) \text{ is non-decreasing, } \sum_{\pi \in \prod} T^{\mathcal{S}}_{\pi}(t) = t \tag{10}$$

where, for a function f, $\dot{f}(t)$ denotes the derivative of f at t. When $\dot{f}(t)$ is used, it is assumed that f is differentiable at t. Any solution (D, T, Z) to the equations (8)-(10) is called a fluid model solution. Consider the following definition from [6]:

Definition 4: The fluid model of a switch operating under a matching algorithm is said to be *weakly stable* if for every fluid model solution (D, T, Z) with $Z(0) = 0$, $Z(t) = 0$ for $t \geq 0$.

Lemma 3: A switch operating under a matching algorithm is rate stable if the corresponding fluid model is weakly stable.

Proof: Refer to [6]. ■

From theorem 3, to show that the scheduling algorithm is rate stable, one requires to prove that every fluid model solution (D, T, Z) of switch has the property

$$Z(t) = 0, t \geq 0, \text{ if } Z(0) = 0.$$

We will use the following definition from [20] and a lemma from [6].

Definition 5: A function f is said to be *proper Lyapunov function* (which takes the sizes of the VOQs, x, as arguments), if it has the following two properties:(a) F is monotonically non-decreasing function with $F(0) = 0, F(x) > 0$ for $x > 0$. (b) $\dot{F}(x)$ exists for all $x \geq 0$.

Lemma 4: Suppose there exists a proper Lyapunov function, $f : [0, \infty) \rightarrow [0, \infty)$, which is an absolutely continuous with $f(0) = 0$ for the fluid model of the switch. Also suppose that $\dot{f}(t) \leq 0$ for almost every t (wrt Lebesgue measure) for which $f(t) > 0$ and f is differentiable at t. Then $f(t) = 0$ for almost every $t \geq 0$, and the fluid model of the switch is weakly stable.

Proof: Refer to [6]. ■

B. *Proof for Theorem 1*

We will use the above lemma in proving the main theorem of this section.

Theorem 1: Under an arbitrary scheduling algorithm, the buffered crossbar gives 100% throughput.

Proof: Consider the fluid model of a buffered crossbar having a speedup of two, operating under an arbitrary algorithm. Let (D, T, Z) be a fluid model solution with $Z(0) = 0$. Let $L_i(t) = \sum_k Z_{ik}(t)$ denote the total amount of fluid queued at input i at time t. Similarly, let $M_j(t) = \sum_k \tilde{Z}_{kj}(t)$ be the total amount of fluid destined for output j (including the buffers for output j) and queued at some input at time t.

Let Q be the $N \times N$ matrix with each entry being 1. One can check that

$$L(t) = QZ(t), M(t) = \tilde{Z(t)}Q \quad t \geq 0. \tag{11}$$

Define,

$$\begin{aligned} f_1(t) &= \langle Z(t), L(t) \rangle. && (12) \\ f_2(t) &= \langle \tilde{Z(t)}, M(t) \rangle. && (13) \\ f(t) &= f_1(t) + f_2(t) && (14) \end{aligned}$$

where for two matrices A and B of the same size, $\langle A, B \rangle = \sum_{ij} A_{ij} B_{ij}$. It follows that $f(t) \geq 0$ for $t \geq 0$ and $f(0) = 0$. It is also clear that $f(t) = 0$ implies that $Z(t) = 0$. We would like to show that $f(t) > 0$ implies $\dot{f}(t) \leq 0$, from which and Lemma 4 the weak stability of the fluid model follows. We claim (and will shortly prove) that

$$\dot{f}(t) = 2\langle Z(t), \dot{L}(t) \rangle + 2\langle \tilde{Z(t)}, \dot{M}(t) \rangle. \tag{15}$$

To establish (15), one first observes that from (12)

$$\begin{aligned} f_1(t) &= \sum_{i,j} Z_{ij}(t) L_i(t) \\ &= \sum_{i,j} Z_{ij}(t) \Big(\sum_k Z_{ik}(t) \Big) \\ &= \sum_{i,j,k} Z_{ij}(t) Z_{ik}(t) \end{aligned}$$

Therefore,

$$\begin{aligned} \dot{f}_1(t) &= \sum_{i,j,k} \dot{Z}_{ij}(t) Z_{ik}(t) + \sum_{i,j,k} Z_{ij}(t) \dot{Z}_{ik}(t) \\ &= 2 \sum_{i,j,k} Z_{ij}(t) \dot{Z}_{ik}(t) \end{aligned}$$

Similarly from (13)

$$\begin{aligned} f_2(t) &= \sum_{i,j} \tilde{Z}_{ij}(t) M_j(t) \\ &= \sum_{i,j} \tilde{Z}_{ij}(t) \Big(\sum_k \tilde{Z}_{kj}(t) \Big) \\ &= \sum_{i,j,k} \tilde{Z}_{ij}(t) \tilde{Z}_{kj}(t) \end{aligned}$$

Therefore,

$$\begin{aligned} \dot{f}_2(t) &= \sum_{i,j,k} \dot{\tilde{Z}}_{ij}(t) \tilde{Z}_{kj}(t) + \sum_{i,j,k} \tilde{Z}_{ij}(t) \dot{\tilde{Z}}_{kj}(t) \\ &= 2 \sum_{i,j,k} \tilde{Z}_{ij}(t) \dot{\tilde{Z}}_{kj}(t) \end{aligned}$$

So from (14)-(16),

$$\begin{aligned} \dot{f}(t) &= 2 \sum_{i,j,k} \Big(Z_{ij}(t) \dot{Z}_{ik}(t) + \tilde{Z}_{ij}(t) \dot{\tilde{Z}}_{kj}(t) \Big) \\ &= 2 \sum_{i,j} \Big(Z_{ij}(t) \sum_k \dot{Z}_{ik}(t) \Big) + \Big(\tilde{Z}_{ij}(t) \sum_k \dot{\tilde{Z}}_{kj}(t) \Big) \end{aligned}$$

Re-substituting, $\tilde{Z}_{ij} = Z_{ij} + B_{ij}$, and using (4), we get

$$\dot{f}(t) = 2 \sum_{i,j} \Big(\Big(Z_{ij}(t) \sum_k \dot{C}_{ik}(t) \Big) + \Big(B_{ij}(t) \sum_k \dot{\tilde{Z}}_{kj}(t) \Big) \Big)$$

Define $\dot{P}_{ij} = Z_{ij}(t) \sum_k \dot{C}_{ik}(t)$. Also define $\dot{Q}_{ij} = B_{ij}(t) \sum_k \dot{\tilde{Z}}_{kj}(t)$. We will now show that if $f(t) > 0$, then $\dot{f}(t) < 0$. Now we know from (4) that $\sum_k \dot{C}_{ik}(t)$ is strictly negative when $Z_{ij}(t) > 0$ and the traffic conforms to (3) i.e. it is admissible. So the product term, $\dot{P}_{ij} \leq 0$. Similarly, if $B_{ij}(t) = 1$, then $\sum_k \dot{\tilde{Z}}_{kj}(t) < 0$. So the product $\dot{Q}_{ij} \leq 0$. In both cases for a given (ij) pair, $\dot{P}_{ij}$ and $\dot{Q}_{ij}$ are equal to zero only if $Z_{ij} = 0$ and $B_{ij} = 0$ respectively. It follows that if $f(t) > 0$, then $\dot{f}(t) < 0$. So the fluid model of the switch is weakly stable. From Lemma 3, the scheduling algorithm is rate stable and since the traffic is admissible, the buffered crossbar gives 100% throughput. ■

APPENDIX B
PROOF FOR THEOREM 4

Before we prove the theorem, we will need the following Lemmas.

Lemma 5: After the modified arrival phase, all cells in the crosspoints $B(i, j)$ will have a departure time lower than any cell queued at input i for output j.

Proof: (By induction) Assume that the above property holds until time slot t. At time $t + 1$, there can be at most one newly arriving cell c to an input which have a lower departure time than the cell in the corresponding crosspoint. By definition, the modified arrival phase allows cell c, to swap with the cell in the corresponding crosspoint. Hence, the above property continues to hold at time $t + 1$. ■

Lemma 6: The total margin, $TM(c)$, of a cell c waiting on the input side is non-decreasing from time slot to time slot.

Proof: Given the newly defined input and output scheduling policies and Lemma 5 the only other difference as compared to Lemma 1 is in the modified arrival phase. Irrespective of whether a swap occurred or not, there is only one newly arriving cell to deal with i.e. if a swap does not occur, then it is a cell which just arrived at the input, else if a swap occurs, then the newly arriving cell is the cell from the swapped crosspoint. The rest of the proof is similar to Lemma 1. ■

Lemma 7: The total margin of a newly arriving cell is non-negative.

Proof: As described in Lemma 6, we only need to be concerned about inserting one newly arriving cell to the priority list at the input irrespective of whether a swap occurred or not. The rest of the proof is similar to Lemma 2. ■

Theorem 4: A buffered crossbar can mimic a PIFO-OQ router (and hence give delay guarantees) with speedup three, regardless of the incoming traffic pattern.

Proof: Lemma 6 and Lemma 7 imply that the total margin of a cell can never be non negative. So every cell reaches its output port before its scheduled departure time. ■