On the speedup required for combined input and output queued switching*

Balaji Prabhakar   *Nick McKeown*†
Departments of Electrical Engineering and Computer Science
Stanford University.

Abstract

Architectures based on a non-blocking fabric, such as a crosspoint switch, are attractive for use in high-speed LAN switches, IP routers, and ATM switches. When operating at the highest speed, memory bandwidth limitations dictate that queues be placed at the input of the switch. But it is well-known that input-queueing can lead to low throughput, and does not allow the control of latency through the switch. This is in contrast to output-queueing which maximizes throughput and permits the accurate control of packet latency through scheduling. We ask the question: Can a switch with combined input and output queueing be designed to *behave identically* to an output-queued switch? In this paper, we prove that if the switch uses virtual output queueing and has an internal speedup of just four, it is possible for it to behave identically to an output queued switch, regardless of the nature of the arriving traffic. Our proof is based on a novel scheduling algorithm, called *Most Urgent Cell First.* We find that with a speedup of four the most urgent cell first algorithm (or MUCFA) enables perfect emulation of a FIFO output-queued switch, i.e. one in which packets depart in the same order that they arrived. We extend this result to show that with a small modification, the MUCFA algorithm enables perfect emulation of a variety of output scheduling policies, including strict priorities and weighted fair-queueing. This result makes possible switches that perform as if they were output-queued, yet use memories that run more slowly.

*To appear in the Special Issue on Control Methods for Communication Networks, (issue **35**, 12, Dec 1999) of *Automatica.*

†This research was initiated when Nick McKeown visited BRIMS, Hewlett-Packard Labs, Bristol in September '96.

1 Introduction

Many commercial switches and routers today employ output-queueing.[1] When a packet arrives at an output-queued (OQ) switch, it is immediately placed in a queue that is dedicated to its outgoing line, where it will wait until departing from the switch. This approach is known to maximize the throughput of the switch: so long as no input or output is oversubscribed, the switch is able to support the traffic and the occupancies of queues remain bounded.

The use of a separate queue for each output means that flows of packets for different outputs are kept separate, and cannot interfere with each other. By carefully scheduling the time that a packet is placed onto the outgoing line, a switch or router can control the packet's latency, and hence provide quality-of-service (QoS) guarantees. But output queueing is impractical for switches with high line rates, or with a large number of ports: the fabric and memory of an $N \times N$ switch must run N times as fast as the line rate (or the bandwidth of the input/output lines). Unfortunately, at the highest line rates, memories with sufficient bandwidth are simply not available. For example, consider a 32×32 OQ switch operating at a line rate of 10Gbit/s. If we use a 512-bit memory datapath (that is, 512 is the number of bits that can be simultaneously written into or read out of the memory), we require memory devices that can perform both a write *and* a read operation every 1.6ns.

On the other hand, the fabric and the memory of an input queued (IQ) switch need only run as fast as the line rate. This makes input queueing very appealing for switches with fast line rates, or with a large number of ports. That is, for a given speed of memory it is possible to build a faster switch; or for a given speed switch it is possible to use slower, lower-cost memory devices. For example, consider again the 32×32 switch operating at a line rate of 10Gbit/s. If the switch uses input-queueing instead of output-queueing, we can use memory devices that perform a write and a read operation every 51.2ns. This is readily achievable with commercially available memories.

But the main problem of IQ switching is head-of-line (HOL) blocking, which can have a severe effect on throughput. It is well-known that if each input maintains a single FIFO, then HOL blocking can limit the throughput to just 58.6% [9].

One method that has been proposed to reduce HOL blocking is to increase the "speedup" of a switch. A switch with a speedup of S can remove up to S packets from each input and deliver up to S packets to each output within a time slot, where a time slot is the time between packet arrivals at input ports. Hence, an OQ switch has a speedup of N while an IQ switch has a speedup of 1. For values of S between 1 and N packets need to be buffered at the inputs before switching as well as at the outputs after switching. We call this architecture a combined input and output queued (CIOQ) switch.

[1] When we refer to output-queueing in this paper, we include designs that employ centralized shared memory.

Both analytical and simulation studies of a CIOQ switch which maintains a single FIFO at each input have been conducted for various values of speedup [4, 8, 7, 14, 5]. A common conclusion of these studies is that with $S = 4$ or 5 one can achieve about 99% throughput when arrivals are independent and identically distributed at each input, and the distribution of packet destinations is uniform across the outputs.

But it has been shown that a throughput of 100% can be achieved with a speedup of just one, if we arrange the input queues differently. That is, HOL blocking can be eliminated entirely using a scheme known as *virtual output queueing* in which each input maintains a separate queue for each output. It has been shown that for independent arrivals, the throughput of an IQ switch can be increased to 100% [12]. We may draw the conclusion: *Speedup is not necessary to eliminate the effect of HOL blocking.*

In practice, we are not only interested in the throughput of a switch, but also in the latency of individual packets. This is particularly important if a switch or router is to offer QoS guarantees. Packets in an IQ switch not only contend for an output, they also contend for entry into the switch fabric with packets that are destined for other outputs. We call this phenomenon *input contention*. Each input can deliver only one packet into the fabric at a time; if it has packets for several free outputs, it must choose just one packet to deliver, holding other packets back. This places a packet at the mercy of other packets destined for other outputs. This is in stark contrast with output-queueing, where a packet is unaffected by packets destined for other outputs. We may draw the conclusion: *To control delay, we need a mechanism to eliminate input contention.*

Previous studies of CIOQ switches make no guarantees about the delay of an individual packet; instead they consider only average delay and throughput. We are interested in controlling the delay of individual packets. Hence our result subsumes previous work, and our approach is quite different. Rather than find values of speedup that work well on average, or with simplistic and unrealistic traffic models, we find the minimum speedup such that a CIOQ switch behaves *identically* to an OQ switch for *all* types of traffic. Here, "behave identically" means that, when the *same* inputs are applied to both the OQ switch and to the CIOQ switch, the corresponding output processes from the two switches are completely indistinguishable. Two processes are indistinguishable if and only if their packet sequences are identical – both in terms of packet-occurance times and packet identities. Further, we place no restrictions on arrivals: our results apply for any type of traffic, even if the arrivals saturate the switch.

The need for a switch that can deliver a certain grade of service, *irrespective of the applied traffic* is particularly important given the number of recent studies that show how little we understand about network traffic processes [11, 16]. Indeed, a sobering conclusion of these studies is that it is not yet possible to accurately model or simulate a trace of actual network traffic. Furthermore, new applications, protocols or data-coding mechanisms may bring new traffic types in future years. A well-designed network switch should perform predictably in the face of all types of arrival process.

In this respect the formulation presented here is both novel and powerful: We seek algorithms that enable a CIOQ switch to perform exactly the same as an OQ switch, using memory devices operating more slowly, for arbitrary switch sizes, and for arbitrary input traffic patterns. Specifically, we exhibit a packet scheduling algorithm that enables a CIOQ switch with a speedup of four to mimic an OQ switch employing a variety of scheduling policies including, FIFO, Strict Priority and Weighted Fair Queueing (WFQ). We also discuss the complexity of implementing the algorithms we propose.

Since this formulation of the problem became known, some interesting contributions including novel algorithms and implementations have been made, most notably in Charny [1], Charny et al [2], Chuang et al [3], Krishna et al [10].

The paper is organized as follows. Sections 2 and 3 introduce the notation and terminology that will be used in the sequel. Sections 4 and 5 prove that it is possible to exactly emulate a FIFO OQ switch by a CIOQ switch so long as the fabric speedup is at least 4. Section 6 extends this result to OQ switches that employ a variety of "monotone" output scheduling policies, including Strict Priority and Weighted Fair Queueing. Section 7 discusses the complexity of implementing the algorithms we introduce. Section 8 concludes.

2 Exact mimicking of FIFO output-queueing

Consider the single stage, $N \times N$ switch shown in Figure 1. Throughout the paper we assume that packets begin to arrive at the switch from time $t = 1$, the switch having been empty before that time. Although packets arriving to the switch or router may have variable length, we will assume that they are treated 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 [15, 17]. We take the arrival time between cells as the basic time unit. The switch is said to have a *speedup of* S, for $S \in \{1, 2, \cdots, N\}$ if it can remove up to S cells from each input and transfer at most S cells to each output in a time slot. A speedup of S requires the fabric of the switch to run S times as fast as the input or output line rate. As mentioned in the introduction, the extreme values of $S = 1$ and $S = N$ give a purely input-queued (IQ) and a purely output-queued (OQ) switch respectively. For $1 < S < N$ buffering is required both at the inputs and at the outputs, and leads to a combined input and output queued (CIOQ) architecture. The following is the problem we wish to solve.

The speedup problem: Determine the smallest value of S, say S_{min}, and an appropriate cell scheduling algorithm π that

1. allows a CIOQ switch to exactly mimic the performance of an output-queued switch (in a sense that will be made precise),

2. achieves this for any arbitrary input traffic pattern,

3. is independent of switch size.

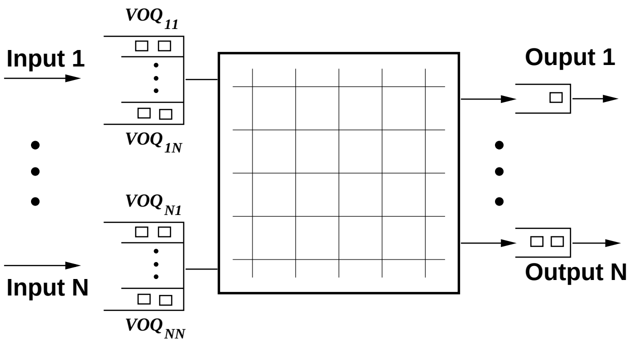

Figure 1: *A schematic of a CIOQ switch*

In an OQ switch, arriving cells are immediately forwarded to their corresponding outputs. This (a) ensures that outputs never idle so long as there is a cell destined for them in the system, and (b) allows the departure of cells to be scheduled to meet latency constraints. Because of these features an OQ switch has the highest possible throughput and allows a tight control of cell latency which is important for supporting multiple qualities-of-service (QoS). We will require that any solution of the speedup problem possess these two desirable features; that is, a CIOQ switch must *exactly mimic* the performance of an OQ switch in the following sense.

Identical Behavior: Consider an OQ switch whose output buffers are first-in-first-out (FIFO). A CIOQ switch is said to *behave identically* to an OQ switch if, under identical inputs, the departure time of every cell from both switches is identical.

To complete the description of the model, we refer to Figure 1 again. All input and output buffers are assumed to have infinite capacity. Each input maintains a separate FIFO queue for cells destined for each output. Hence, there are N FIFO queues at each input. Call these queues *Virtual Output Queues (VOQs)* – with the understanding that VOQ_{ij} buffers cells at input i destined for output j. Finally, we wish to make explicit the assumption that the output buffers of the CIOQ switch are not necessarily FIFO, although the OQ switch whose performance it is mimicking has FIFO output buffers.

A scheduling algorithm selects a matching between inputs and outputs in such a way that each non-empty input is matched with at most one output and, conversely, each output is matched with at most one input. The matching is used to configure the switch before cells are transferred from the input side to the output side. A CIOQ switch with a speedup of S is able to make S such transfers each time slot.

3 MUCFA: A scheduling algorithm that achieves identical behavior

In this section we present a novel scheduling algorithm that allows a CIOQ switch with a small speedup to behave identically to an OQ switch for any input traffic. The algorithm is called the *Most Urgent Cell First Algorithm* (MUCFA).

We begin by introducing the notion of a "phase".

Definition 1 *For a switch with speedup S, a time slot is said to be divided into S equal phases. During each phase ϕ_i, $1 \leq i \leq S$, the switch can remove at most one cell from each input and can transfer at most one cell to each output.*

It is assumed that cells arriving at the switch will do so at the beginning of phase ϕ_1, while departures from the switch take place at the end of phase ϕ_S.

A crucial aspect of MUCFA is the concept of the "urgency of a cell". Recall that the definition of "identical behavior" requires a CIOQ switch to *identically* match cell departures with an OQ switch when they are both subjected to identical inputs. Therefore, our definition of identical behavior requires a CIOQ switch and a reference OQ switch. This is illustrated in Figure 2.

The urgency of a cell is first explained with respect to the reference OQ switch. Every arriving cell to this switch is stamped with a number, which is its "urgency value" at that time. This number indicates the time from the present that it will depart from the switch. At each successive time slot, the urgency value is decremented by one. When the value reaches zero, the cell will depart. Alternatively, since the buffers of the OQ switch are FIFO, the urgency of a cell at any time equals the number of cells ahead of it in the output buffer at that time.

More precisely, if a cell c arrives at input i at time T and departs from output j at time $D \geq T$, its urgency at any time R, $T \leq R \leq D$, equals $D - R$. Suppose there are two cells, a and b, in the buffer at output j at some time, with urgencies u_a and u_b, respectively. Cell a is said to be "more urgent" than b if $u_a < u_b$. Given that the output buffer is FIFO, it is clear that if b arrived at the switch after a then necessarily $u_a < u_b$. If a and b arrive at the same time, then $u_a < u_b$ iff the number of the input port at which a arrives is less than the number of the input port at which b arrives. That is, the OQ switch is assumed to transfer cells from inputs to outputs in a round robin fashion starting with the smallest numbered input first.

Now consider the CIOQ switch. By assumption, the same input is applied to it and to the OQ switch. (Thus for every cell c in the CIOQ switch, there is an exact copy in the OQ switch. We shall refer to this exact copy as the "clone of cell c".) Therefore, cell c arrives at input i at time T and is destined for output j. Since the speedup may now be

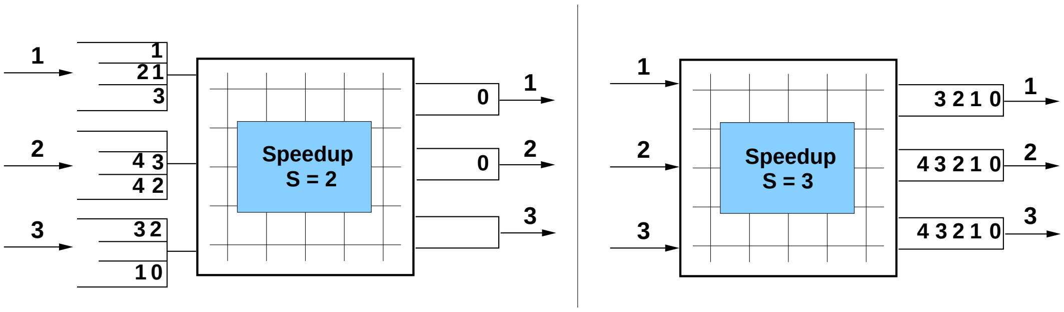

Figure 2: *A CIOQ switch (left) and its reference OQ switch (right).*

less than N, c may not be forwarded to the buffer at j during time slot T. Note that c *may not be required* at output j for some time, because its clone in the OQ switch is some distance from the HOL. Therefore, the urgency is an indication of how much time there is before c is needed at its output if the CIOQ switch is to mimic the behavior of the OQ switch. This motivates the following definition.

Definition 2 *The urgency of a cell in a CIOQ switch at any time is the distance its clone is from the head of the output buffer in the corresponding reference OQ switch.*

The cells in any output buffer of the CIOQ switch are arranged in increasing order of urgencies, with the most urgent cell at the head. Once cell c is forwarded to its output in the CIOQ switch, its position is determined by its urgency.

We are now ready to describe the Most Urgent Cell First Algorithm (MUCFA).

Phase-by-phase description of MUCFA

1. At the beginning of each phase outputs try to obtain their most urgent cells from the inputs.

2. If more than one output requests an input, then the input will grant to that output whose cell has the smallest urgency number. If there is a tie between two or more outputs, then the output with the smallest port number wins.

3. Outputs that lose contention at an input will try to obtain their next most urgent cell from another (unmatched) input.

4. When no more matching of inputs and outputs is possible, cells are transferred and MUCFA goes to the next phase (Step 1).

The operation of MUCFA over one time slot is illustrated by means of an example in Figure 3. Note that at the beginning of phase 1, both outputs 1 and 2 request input 1 to obtain their most urgent cells. Since there is a tie in the urgency of their requests, by our assumption input 1 grants to output 1. Output 2 proceeds to obtain its next most urgent cell which happens to be at input 2 and has an urgency of 3.

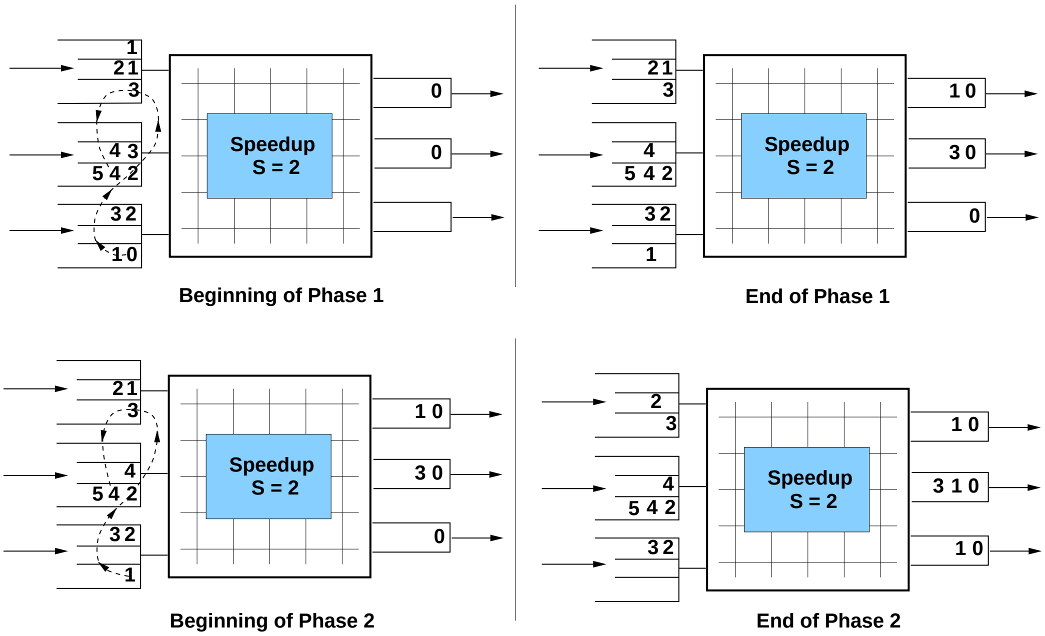

Figure 3: *The operation of a 3×3 CIOQ switch with* $S = 2$ *over one time slot under MUCFA. The dashed lines indicate the "output thread" of the cell with urgency 5 in* VOQ_{23}.

We can make the following key observation about the working of MUCFA: During any phase, there are only two reasons that a cell will not be transferred from its input to its output

Input contention: The output is ready to receive the cell, but the input wants to send a more urgent cell. (In the example of Figure 3, output 2 can't receive its most urgent cell in phase 1 because input 1 wants to send to output 1.)

Output contention: The input wants to send the cell, but the output wants to receive a more urgent cell. (In phase 2 of the example of Figure 3, input 2 can't send its most urgent cell because output 3 wants to receive from input 3.)

3.1 MUCFA and the Stable Marriage Problem

The way in which MUCFA matches inputs and outputs is a variation of the *stable marriage problem*, which was first introduced by Gale and Shapley in 1962 [6]. Solutions to the stable marriage problem find a "stable" and complete matching between inputs and outputs. A match is *unstable* if there is an input and output who are not matched to each other, yet both prefer the other to their partner in the current matching. A *stable* matching is any matching that is not unstable. There exists a well-known algorithm (the Gale-Shapley algorithm, or GSA) that will always find a stable matching in N^2 iterations.

MUCFA can be implemented using the GSA with preference lists as follows. Output j first assigns a preference value to each input i, equal to the urgency of the cell at head-of-line of VOQ_{ij}. If VOQ_{ij} is empty then the preference value of input i for output j is set to $+\infty$. The preference list of the output is the ordered set of its preference values for each input. Likewise, each input assigns a preference value for each output, and creates the preference list accordingly. A matching of inputs and outputs can then be obtained using GSA. The relationship between the stable marriage problem and cell scheduling is explored in more detail in [13].

4 The main result

Theorem 1 *An $N \times N$ CIOQ switch operating under MUCFA can behave identically to an OQ switch, regardless of input traffic patterns and for arbitrary values of N, if its speedup $S \geq 4$.*

Theorem 2, which is a strengthening of Theorem 1 will be proved in the next section. For now, we will explore some of the implications of Theorem 1, assuming that it is true. This will allow us to come to certain conclusions which help in the statement and proof of Theorem 2. In order to proceed, we will need to introduce the concept of "output threads" and "input threads".

Definition 3 *At any time, the <u>output thread of a cell c</u> which is queued in VOQ_{ij} is the ordered set of all cells c' which are queued in $VOQ_{i'j}$, $1 \leq i' \leq N$, and are more urgent than c. The <u>thread of output j</u> is the output thread of its least urgent cell.*

For example, the output thread of the cell with urgency five in VOQ_{23} at the beginning of phase 1 (see Figure 3) has cells with urgencies $\{0, 1, 2, 3, 4\}$. The output thread of the same cell at the beginning of phase 2 has cells with urgencies $\{1, 2, 3, 4\}$. The dashed lines in Figure 3 indicate the output thread of this cell at the beginning of phases 1 and 2.

Definition 4 *The <u>input thread of a cell c</u> queued in VOQ_{ij} is the ordered set of all cells c' which are in $VOQ_{ij'}$, $1 \leq j' \leq N$, and are more urgent than c. If cells p and q have the same urgency then p is placed before q in an input thread if p's output has a smaller number than q's output. The <u>thread of input i</u> is the input thread of its least urgent cell.*

For example, the input thread of the cell with urgency three in VOQ_{13} at the beginning of phase 1 (see Figure 3) has cells with urgencies $\{1, 1, 2\}$. The input thread of the same cell at the beginning of phase 2 has cells with urgencies $\{1, 2\}$.

With these definitions, one may draw some inferences about MUCFA. (The following discussion is intended to motivate the statement and proof of Theorem 2 and is therefore presented in an informal manner.) Consider a CIOQ switch with speedup S operating under MUCFA from time 1, having been empty before that time. It will fail to behave identically to an OQ switch at time T if an input thread has $S+1$ or more cells with urgency 0. If this should happen, then clearly there are not enough phases to transfer all the most urgent cells to their outputs, and MUCFA fails. Therefore, if MUCFA causes a CIOQ switch with speedup S to behave identically to an OQ switch, it must be the case that *every input thread has S or fewer cells with urgency 0 at the beginning of every time slot*. Conversely, if there are always S or fewer cells with urgency 0 at each input, then MUCFA never fails. We record this in the following lemma.

Lemma 1 *A CIOQ switch with speedup S operating under MUCFA behaves identically to an OQ switch if, and only if, there are S or fewer cells with urgency 0 in each input at all times.*

Since cells in an input thread are ordered according to urgency, this is the same as saying that a cell with urgency 0 cannot appear in the $(S+1)^{th}$ position in any input thread. Similarly, it is also clear that a cell with urgency 1 cannot appear in the $(2S+1)^{th}$ position at any time (assuming that every 0 occupies a position less than or equal to S), as this would lead to a failure of MUCFA in the next cell time. In general, Lemma 1 is equivalent to the statement: A CIOQ switch with speedup S operating under MUCFA behaves identically to an OQ switch if, and only if, a cell with urgency l cannot occupy position $(l+1)S+1$ in an input thread at any time.

If we assume that MUCFA behaves identically to an OQ switch at all times when the speedup equals S, it is clear that it will also behave identically at every speedup $S' > S$. Indeed, more ought to be true: Under identical inputs if a tagged cell c is forwarded to its output F phases after its arrival when the speedup is S, then it must be forwarded to its output within $F' \leq F$ phases when the speedup is S'. In particular, if c belongs to the thread of input i at time T when the speedup is S', then it also belongs to the thread of input i at time T when the speedup is S. This implies the following crucial point.

Key observation *If MUCFA behaves identically to an OQ switch at speedup S, then at any speedup $S' \geq S$ a cell with urgency l cannot appear at position $S(l+1)+1$ in an input thread.*

In Theorem 2 we prove the following stronger statement for $S \geq 4$: At the beginning of each time slot T, a cell with urgency l does not occupy position $l+1$ in an input thread; excluding any cell that might have just arrived. If this property were true for all input threads at all times then clearly MUCFA never fails to behave identically to an OQ switch, and Theorem 1 is verified.

5 A speedup of 4 suffices

In this section we shall prove Theorem 2 from which Theorem 1 follows as a corollary. But first, we need to develop the following lemma.

Lemma 2 *Consider a tagged cell c which, at the beginning of time slot T, is at an input of a CIOQ switch with speedup S operating under MUCFA. If c remains in its input at the end of time slot T and is not forwarded to its output, then a totality of S cells either from c's input thread or from its output thread must be delivered to their outputs during time slot T.*

Proof This is a consequence of input and output contention. That is, c is not forwarded to its output during a phase either because a cell in its input (output) thread has kept its input (output) busy. And there are S such phases in each time slot. ■

Theorem 2 *Consider an $N \times N$ CIOQ switch operating under MUCFA with a speedup of S. Suppose that the switch has been operating from time slot 1, having been empty before that time. Let $S^i(t)$ be the thread at input i just at the beginning of time slot t, before any new cells have arrived. Then for each i and for each t, it is <u>never</u> the case that a cell with urgency l occupies position $l+1$ in $S^i(t)$ so long as $S \geq 4$.*

Discussion: Observe that if Theorem 2 is true, there can never be more than two cells at an input with urgency 0 at any time. Indeed, if there are two cells with urgency 0 at an input at time t and Theorem 2 is true, then one of them *must* have arrived at the input at the beginning of time t. Thus, every cell with urgency 0 will be at its respective output at the end of the second phase of every cell time (either because it was already there or because it was transferred in phase 1 or 2). This ensures that the CIOQ switch can mimic the OQ switch at all times.

We now briefly describe the idea of the proof of Theorem 2, which is proved by contradiction. Thus, we suppose that at some time T the conclusion of the theorem is false. In other words, there is a cell, say p, which appears farther in an input thread than its urgency value. We argue that the only way this could have happened is for MUCFA to have transferred cells more urgent than p and therefore left p behind at its input. But, these more urgent cells must belong either to p's input thread or to p's output thread, as these are the only cells that could have taken precedence over p. If we could bound the total number of these more urgent cells (taking into account cells which were already present when p arrived and also new arrivals), then since Lemma 2 guarantees a minimum rate at which the more urgent cells are drained, we can obtain the desired contradiction. The details are as follows.

Proof of Theorem 2 Suppose T is the first time the thread, $S^I(T)$, of input an I has a cell with urgency l occupying position $l+1$. (To be more precise, we must suppose that at time T a cell with urgency k occupies position $k+1$ *or greater*. However, should a cell with urgency k occupy position $k+m$ for $m > 1$, then there must be a cell in the input thread with urgency $l \leq k$ occupying position $l+1$. This is simply a consequence of cells in an input thread being ordered according to nondecreasing urgency values, as a moment's reflection shows.) Consider the thread $S^I_{l+1}(T) \subset S^I(T)$ consisting of the first $l+1$ cells of $S^I(T)$. Note that the least urgent cell of $S^I_{l+1}(T)$ has an urgency of l.

(1) Let c be the cell belonging to $S^I_{l+1}(T)$ that arrived earliest, and let u be its urgency at time T. It follows that $u \leq l$. It also follows that c arrived at least $l+1$ cell times ago.

(2) Suppose c actually arrived at time $T - A$. By (1) $A \geq l+1$, and c's urgency upon arrival equals $u + A$ precisely.

(3) By Lemma 2, every time slot that c is in the system on the input side, a totality of S cells belonging to the input and/or output threads of c must be sacrificed in order to prevent c from going to its output.

(4) Since c arrives at time $T - A$ and remains in its input until time $T - 1$, the number of "sacrifice cells" required during this time period equals the number of phases in $[T - A,\ T - 1]$ which equals $S \times A$.

(5) By assumption of T being the first time at which things go wrong, the maximum number of cells in c's input thread at time $T - A$ is less than or equal to $u + A$. These are possible "sacrifice cells".

(6) By definition of urgency, the maximum number of cells in c's output thread at time $T - A$ is less than or equal to $u + A$. These are also possible "sacrifice cells".

(7) Putting (5) and (6) together, when c arrives, the maximum number of sacrifice cells in its input and output threads is no more than $2(u + A)$.

(8) Between $T - A + 1$ and $T - 1$, the maximum number of cells that can arrive at input I is less than or equal to $A - 1$. Of these arrivals l will belong to $S^I_{l+1}(T)$ and hence cannot be "sacrifice cells". This implies that the maximum number of sacrifice cells that can arrive at input I after c is no more than $A - 1 - l$.

(9) A grand total on the maximum possible "sacrifice cells" is (putting (7) and (8) together): $2(u + A) + A - 1 - l = 3A + u + (u - l) - 1$. But,

$$
\begin{aligned}
3A + u + (u - l) - 1 &\leq 3A + u \qquad (\text{since } u \leq l) \\
&\leq 4A - 1 \qquad (\text{since } u \leq l \leq A - 1).
\end{aligned}
$$

(10) The number in (9) falls short of the requirement in (4) if $S \geq 4$. This contradiction proves the theorem. ■

6 Extension to arbitrary output scheduling policies

So far we have only considered emulating an OQ switch that employs the FIFO scheduling policy using a CIOQ switch. We have done this by providing an algorithm, MUCFA, that uses "urgencies" to aid its scheduling. These urgencies were inferred from the reference OQ switch.

We will now show that MUCFA can be used to mimic an OQ switch employing a wider range of output scheduling policies than just FIFO. Essentially, the extension to these non-FIFO scheduling policies involves very little change to the basic structure of MUCFA and to the theorems of the previous section. We shall first need to develop some notation and specify the class of output scheduling policies that will be considered.

Definition 5 *An output scheduling policy is a rule operating independently at each output port and determines the order in which cells depart from that output.*

Thus, if an output employs the FIFO policy, the departure order of cells is determined by their arrival times (and input port numbers, if multiple cells arrive at different inputs for that output in a single cell time).

Definition 6 *An output scheduling policy is said to be monotone if, once it has been determined, the relative departure order of any two cells p and q does not change over time.*

A simple way of visualizing this class of policies is to imagine a single "push-in" queue at the output, where an arriving cell may be pushed into any location but cells may depart only from the front. Note that newly arriving cells may increase the absolute departure time of an existing cell, but cannot change the position of the existing cell relative to another.

The importance of the class of monotone policies is that it includes several policies that are commonly used to provide quality-of-service (QoS) guarantees. For example, it includes the Strict Priority Policy (wherein certain flows have a strictly higher priority than other flows), and Weighted Fair Queueing (WFQ) (in which the bandwidth at the output port is shared among outgoing flows in proportion to their declared weights).

Definition 7 *The expected departure time, $EDT_c(t)$, of a cell c at any time t is the time from the present that it would depart from the switch if no new cells arrived to the switch after time t.*

Thus $EDT_c(t)$ is the same as the urgency of c when the output scheduling policy is FIFO. For FIFO output scheduling policies $EDT(t)$ decreases exactly by one every time slot. This need not be the case in general since new cells may arrive for an output that take precedence over an existing cell, causing its EDT to increase.

6.1 MUCFA for QoS

We now show how a variant of the algorithm MUCFA may be used in a CIOQ switch with a speedup of four to enable it to behave identically to an OQ switch employing any monotone scheduling policy.

Let MUCFA-E be the algorithm that during any phase of time slot t schedules the transfer of cells from inputs to outputs in exactly the same manner as MUCFA, except for basing its scheduling decisions on a cell's $EDT(t)$ instead of its "urgency".

Corresponding to Definitions 3 and 4, we have the following definitions.

Definition 8 *At any time t, the output thread, $OT_c(t)$, of a cell c which is queued in VOQ_{ij} is the ordered set of all cells c' queued in $VOQ_{i'j}$, $1 \leq i' \leq N$ whose EDT is smaller than the EDT of c.*

Definition 9 *At any time t, the input thread, $IT_c(t)$, of a cell c queued in VOQ_{ij} is the ordered set of all cells c' which are in $VOQ_{ij'}$, $1 \leq j' \leq N$ whose EDT is smaller than the EDT of c. If two cells p and q at input i have the same EDT, then p is placed before q in an input thread if p's output has a smaller number than q's output.*

In light of the above definitions, we paraphrase Theorem 2 in a form that is more suitable for our purpose. Thus Theorem 2 is equivalent to the following statement:

Theorem 3 (Restatement of Theorem 2) *Consider an $N \times N$ CIOQ switch operating under MUCFA with a speedup of $S \geq 4$. Suppose that the CIOQ switch is mimicking a FIFO OQ switch. Suppose also that the CIOQ switch has been operating from time 1, having been empty before then. At any time $t \geq 1$ and for every cell c, $IT_c(t) \leq EDT_c(t)$, excluding any new arrivals to the switch at time t.*

Proof Observe that $EDT_c(t)$ is the same as the urgency of c at time t (Definition 2), since the OQ switch is employing the FIFO policy. By the conclusion of Theorem 2, $IT_c(t) \leq u_c(t)$. Thus Theorem 3 is verified. ∎

We are now ready to extend Theorem 2 to monotone output scheduling policies.

Theorem 4 *Consider an $N \times N$ CIOQ switch operating under MUCFA-E with a speedup of $S \geq 4$. Suppose that the CIOQ switch is aiming to mimic an OQ switch employing* *a monotone output scheduling policy. If the CIOQ switch has been operating from time 1, having been empty before then, then at any time $t \geq 1$ and for every cell c, $IT_c(t) \leq EDT_c(t)$, excluding any new arrivals to the switch at time t. If this true at all times, the CIOQ switch will mimic the OQ switch.*

Remark: The crucial difference in the case of monotone output scheduling policies is that new cells arriving to the switch after a cell c might increase *both* its input and output threads. Since only one cell can arrive at an input per time slot, the input thread of c can only increase by one. But this increase in input thread by one occurs even in the setting of Theorem 2 and has been shown to cause no trouble. However, the output thread of c can increase by upto n per time slot, where $1 \leq n \leq N$. This happens when cells arrive at n different inputs destined for c's output and have a higher priority than c.

It is therefore tempting to think that these new arrivals to c's output thread will become "sacrifice cells" and prevent c from being forwarded to its output on time. Although the new higher priority arrivals to c's output are potential "sacrifice cells", the fortunate situation is that they will *simultaneously increase* c's EDT, making c less urgent for its output! Hence they also will not cause any problems. The details follow.

Proof The proof involves a minor modification to the proof of Theorem 2. Aiming for a contradiction, let us suppose that T is the first time the thread, $S^I(T)$, of input I has a cell with $EDT(T)$ equal to l occupying position $l+1$. (As a gentle reminder, we recall that $EDT(T)$ is calculated by excluding all cells that arrive at the switch at the beginning of time T.) Consider the thread $S^I_{l+1}(T) \subset S^I(T)$ consisting of the first $l+1$ cells of $S^I(T)$. Note that the cell with the largest expected departure time in $S^I_{l+1}(T)$ has an $EDT(T)$ equal to l.

(1) Let c be the cell belonging to $S^I_{l+1}(T)$ that arrived earliest, and let $EDT_c(T)$ be its expected departure time at the beginning of time T. It follows that $EDT_c(T) \leq l$. It also follows that c arrived at least $l+1$ cell times ago (since it was the earliest arrival in $S^I_{l+1}(T)$).

(2) Suppose c actually arrived at time $T - A$. By (1) $A \geq l+1$, and let c's EDT upon arrival equal $EDT_c(T - A)$.

(3) By Lemma 2, every time slot that c is in the system on the input side, a totality of S cells belonging to the input and/or output threads of c must be sacrificed in order to prevent c from going to its output.

(4) Since c arrives at time $T - A$ and remains in its input until time $T - 1$, the number of "sacrifice cells" required during this time period equals the number of phases in $[T - A,\ T - 1]$ which equals $S \times A$.

(5) By assumption of T being the first time at which things go wrong, the maximum number of cells in c's input thread at time $T - A$ is less than or equal to $EDT_c(T - A)$. These are possible "sacrifice cells".

(6) By definition of EDT, the maximum number of cells in c's output thread at time $T-A$ is less than or equal to $EDT_c(T-A)$. These are also possible "sacrifice cells".

(7) Putting (5) and (6) together, when c arrives, the maximum number of sacrifice cells in its input and output threads is no more than $2EDT_c(T-A)$.

(8) Between $T-A+1$ and $T-1$, the maximum number of cells that can arrive at input I is less than or equal to $A-1$. Of these arrivals l will belong to $S^I_{l+1}(T)$ and hence cannot be "sacrifice cells". This implies that the maximum number of sacrifice cells that can arrive at input I after c is no more than $A-1-l$.

(9) For $T-A+1 \leq t \leq T-1$, let $X(t)$ the number of cells that arrive for c's output *and have a higher priority than* c. These cells will be a part of the output thread of c and are potential "sacrifice cells". Their total number is equal to $\sum_{t=T-A+1}^{T-1} X(t)$.

(10) The EDT's of the cell c at times $T-A$ and $T-1$ are related by the following equation:

$$EDT_c(T-1) = EDT_c(T-A) + \sum_{t=T-A+1}^{T-1} X(t) - (A-1),$$

which merely states that c's EDT can increase by new higher priority arrivals and decreases exactly by one due to a departure from its output during each time slot.

Since at time T we disregard new arrivals to the switch, $EDT_c(T) = EDT_c(T-1)-1$. Therefore,

$$EDT_c(T) = EDT_c(T-A) + \sum_{t=T-A+1}^{T-1} X(t) - A.$$

(11) A grand total on the maximum possible "sacrifice cells" is (putting (7), (8) and (9) together):

$$GT = 2EDT_c(T-A) + A - 1 - l + \sum_{t=T-A}^{T-1} X(t).$$

Now,

$$\begin{aligned}
GT &= EDT_c(T-A) + 2A - 1 - l + EDT_c(T) \qquad \text{(using (10))} \\
&\leq 3A - l - 1 + 2EDT_c(T) \qquad \text{(again, using (10))} \\
&\leq 3A - l - 1 + 2l \qquad (\text{since } EDT_c(T) \leq l \text{ from (1)}) \\
&= 3A + l - 1 < 4A - 1 \qquad (\text{since } l < A \text{ from (2)})
\end{aligned}$$

(12) The number in (11) falls short of the requirement in (4) if $S \geq 4$. This contradiction shows that $IT_c(t) \leq EDT_c(t)$ at all times t, excluding any new arrivals at time t.

To conclude the proof, note if we include new arrivals to the switch at time t, then there cannot be more than two cells at any input with $EDT(t) = 0$. In fact if there are two cells, then one of them must necessarily be a new arrival at time t. As a result all cells with $EDT = 0$ will be at their respective outputs by the end of phase 2, and the CIOQ switch will not fail to behave identically to the OQ switch. ■

7 Implementational Complexity

The previous sections show that it is possible to exactly emulate a wide range of output scheduling policies with a CIOQ switch that employs a speedup of four. It appears that the algorithms we propose for achieving this exact emulation trade one problem for another: They overcome the memory bandwidth (or speedup) requirement of N but introduce an extra scheduling and processing overhead. However, note that memory bandwidths are increasing very slowly over time while processing speeds double roughly every 18 months (according to the so-called Moore's Law). Thus the trade-off is in the correct direction.

Nevertheless, it is still important to understand the exact nature of the trade-off and ask the question: Exactly how hard are MUCFA and MUCFA-E to implement? We do not have a complete answer to this question yet. But the following discussion shows some of the issues that face the implementor of MUCFA and MUCFA-E.

A comparison of the operation of a CIOQ switch employing MUCFA-E to the operation of an OQ switch shows that there are two important additional functions for the CIOQ switch to perform. These are (i) Determining the urgency/expected departure time of an arriving cell, and (ii) The process of matching inputs and outputs for transferring cells. We consider each of these two additional functions in turn. An important third function is to constantly update EDT values of each cell given new, higher priority arrivals. We comment on this aspect in conjunction with (i).

7.1 Determining urgency/EDT

We approach this issue by examining the difficulty of determining the EDT corresponding to a fixed output scheduling policy. We shall find, not surprisingly, that the difficulty of inferring the EDT differs with output scheduling policies. For the sake of concreteness and to get a reasonable sense of the difficulty involved, we consider the following output scheduling policies: FIFO, Strict Priority and WFQ.

The FIFO output scheduling policy: To perform the matching process in MUCFA, the CIOQ switch needs to know the "urgency" of each cell at each time slot. It suffices to determine the urgency of an arriving cell; since under the FIFO output scheduling policy, the urgency of a cell decreases exactly by one each time slot. By simply decrementing the urgency value by one in each time slot, the CIOQ switch knows the urgency of any cell at any time. (Of course, having to decrement the urgency of *every* cell in *every* time slot is a nuisance which can easily be avoided by marking the arrival time T_A and the urgency upon arrival U_A for each cell. Then the urgency at a future time $T > T_A$ is given by $U_T = U_A - (T - T_A)$.)

So, how can the CIOQ switch infer the urgency of an arriving cell? Suppose each output in the CIOQ switch maintains a pointer that indicates the value of the urgency to be assigned to the next arriving cell destined for that output. Say that at the beginning of time t, a cell arrives at input i destined for output j. If this is the only arriving cell destined for output j, then input i assigns the urgency value maintained by the pointer at output j to the cell and output j increases the value of the pointer by 1. If n cells arrive at different inputs destined for output j, then they each receive the *same* urgency value from the pointer at output j and the pointer is increased by n. Ties in urgency values among cells at different inputs destined for the same output are broken by using input port numbers. At the end of each time slot the pointer is decremented by one so long as it is positive. Of course, the pointer is not decremented if its value is zero at the end of a time slot.

The above scheme points out that it is not necessary for the CIOQ switch to refer to the OQ switch to determine the urgency of cells. Indeed, it also points out that there is no need for an elaborate communication mechanism between various inputs and outputs to determine urgencies – a single query from an input to the pointer of an output at the beginning of the time slot is sufficient.

The Strict Priority output scheduling policy: Recall that an output is said to employ the Strict Priority policy if the traffic leaving that output is divided into C classes, say, and cells belonging to Class m will not be allowed to leave the switch so long as there is a cell in the switch belonging to Class n for $n < m$. Within each class cells are ordered FIFO for departure. As may be readily verified, the Strict Priority policy is a monotone output scheduling policy.

Under this scheme each output pointer is a vector $(EDT_1, ..., EDT_C)$ denoting the EDT values to be assigned to cells that arrive for that output in each of the C classes. If at the beginning of time t a cell belonging to Class n has an EDT value E_n, then its actual departure time if no new cells arrived to the switch during times t or greater is exactly equal to $t + E_n + \sum_{i \leq n-1} EDT_i$.

To determine the EDT of an arriving Class n cell, the corresponding input queries the appropriate output pointer for Class n's current EDT value. The pointer is then incremented by 1. (In case of multiple Class n arrivals we proceed as in the FIFO scheme and assign them all the same EDT value and suitably update the pointer.) Once the EDT values are assigned, cells in an input/output thread are ordered according to classes and within each class they are ordered according to EDT values.

Again notice that there is no need for a reference OQ switch or for any communication between various inputs and outputs. Thus, so far as determining EDTs, the Strict Priority is just a little more complex than the FIFO policy.

The WFQ output scheduling policy: We believe this is a much more complex case. In the previous case of Strict Priority we have exploited the FIFO nature of ordering cells within each class *and* the strict ordering between cells of different classes. If under WFQ we let the number of classes for each output equal the number of flows leaving the switch through that output, it is still true that the cells belonging to each class can be ordered

FIFO for departure. However, in general, there will not be strict ordering between each of the classes. Thus there is a need to know the precise EDT of each cell in relation to all the cells destined for the same output at each time. Perhaps there is a way of simplifying this as well but at the moment we do not expect a substantial simplification.

7.2 Complexity of the matching process

Since MUCFA and MUCFA-E can be implemented as variants of the stable marriage problem using EDTs to draw up the preference lists for inputs and outputs, by the well-known Gale-Shapley Algorithm the worst-case number of iterations required to obtain a match is N^2. However, due to the manner in which preference lists are drawn using EDTs it turns out that worst-case number of iterations required in the switching context is only N. We establish this fact below.

Let $\mathcal{I}$ and $\mathcal{O}$ be two nonempty subsets of the set of all inputs and outputs, respectively. For an input $i \in \mathcal{I}$ let $IT^{i,\mathcal{O}}$ be the thread at input i consisting only of cells destined for outputs $o \in \mathcal{O}$. That is, $IT^{i,\mathcal{O}}$ is obtained from the thread at input i by removing all cells that are not destined for one of the outputs in $\mathcal{O}$. Similarly, define $OT^{o,\mathcal{I}}$ to be the thread for output o consisting of cells at some input $i \in \mathcal{I}$.

Lemma 3 *Consider the set $\mathcal{C}$ of all cells at some input $i \in \mathcal{I}$ destined for some output $o \in \mathcal{O}$. Let $IT^{i,\mathcal{O}}$ and $OT^{o,\mathcal{I}}$ be as described above. There is a cell $c \in \mathcal{C}$ at an input $i \in \mathcal{I}$ destined for an output $o \in \mathcal{O}$ which is at the head of $IT^{i,\mathcal{O}}$ and $OT^{o,\mathcal{I}}$.*

Proof Choose cell $c_1 \in \mathcal{C}$ at some input $i_1 \in \mathcal{I}$ such that it is at the head of $IT^{i_1,\mathcal{O}}$. Suppose that c_1 is destined for output o_1. If c_1 is at the head of $OT^{o_1,\mathcal{I}}$, then we are done. If not, then let c_2 be the cell at the head of $OT^{o_1,\mathcal{I}}$. Observe that due to strict ordering of EDTs on an output thread, $EDT_{c_1} > EDT_{c_2}$. Let c_2 be at input i_2. If c_2 is at the head of $IT^{i_2,\mathcal{O}}$, then we are done. If not, then let c_3 be at the head of $IT^{i_2,\mathcal{O}}$. Since EDTs are in non-decreasing order on input threads, it follows that $EDT_{c_2} \geq EDT_{c_3}$. Proceeding thus, we obtain a series of cells c_n such that

$$EDT_{c_1} > EDT_{c_2} \geq EDT_{c_3} > EDT_{c_4} \geq EDT_{c_5} \cdots \tag{1}$$

Since the cardinality of $\mathcal{I}$ and $\mathcal{O}$ is at most N and each of the c_n's is either at the head of an input thread or an output thread, the total number of c_n's is at most $2N$. Given this and the non-negativity of EDTs, Equation (1) implies that there must be a cell c which is both at the head of its input thread and its output thread. This proves the lemma. ∎

Theorem 5 *For an $N \times N$ CIOQ switch employing either MUCFA or MUCFA-E and an internal speedup of at least 4, the number of iterations required to match inputs and outputs in each phase is never more than N.*

Proof Let $\mathcal{I}_i$ and $\mathcal{O}_i$ be the sets of inputs and outputs, respectively, involved in the i^{th} iteration of the matching process in some phase. Consider $\mathcal{I}_1$ and $\mathcal{O}_1$. As a consequence of Lemma 3, there is a cell d_1, say, at input $k_1 \in \mathcal{I}_1$ destined for output $l_1 \in \mathcal{O}_1$ which is at the head of both $IT^{k_1,\mathcal{O}_1}$ and $OT^{l_1,\mathcal{I}_1}$. This causes output l_1 to request input k_1 in the first iteration and for input k_1 to accept output l_1. Thus input l_1 and output k_1 will be matched and not participate in iterations 2 or higher, and the cardinality of $\mathcal{I}_2$ will be less than the cardinality of $\mathcal{I}_1$ by at least 1. Similarly, the cardinality of $\mathcal{O}_2$ will be less than the cardinality of $\mathcal{O}_1$ by at least 1.

Repeated application of Lemma 3 guarantees that the cardinality of $\mathcal{I}_n$ decreases by at least one in each iteration. Since the cardinality of $\mathcal{I}_1$ is at most N, we get the desired result. ∎

8 Conclusion

With the continued demand for faster and faster switches, it is increasingly difficult to implement switches that use output queueing or centralized shared memory. Before long, it may become impractical to build the highest performance switches and routers using these techniques.

It has been argued for some time that most of the advantages of output-queuing (OQ) can be achieved using combined input and output queueing (CIOQ). While this has been argued for very specific, benign traffic patterns there has always been a suspicion that the advantages would diminish in a more realistic operating environment.

Our result proves that a CIOQ switch can *behave identically* to an OQ switch that employs a variety of "monotone" scheduling policies, including Weighted Fair Queueing (WFQ). Perhaps more importantly, we show this is true for any sized switch, or for any traffic arrival pattern. The three sufficient conditions for this result to hold are: (i) virtual output queues are maintained at each input, (ii) at the end of each cell time, a novel scheduling algorithm, which we call *Most Urgent Cell First* be used to configure the non-blocking switch fabric, and (iii) the switch fabric and memory run four times as fast as the external line rate; i.e. at a speedup of four.

Acknowledgement: The authors thank Shang-Tse Chuang, Ashish Goel and Mingyan Zhu of Stanford University for various discussions of the speedup problem. Nick McKeown also thanks Jeremy Gunawardena, Scientific Director of Hewlett-Packard's Basic Research Institute in the Mathematical Sciences (BRIMS), for inviting him to BRIMS where much of this work was initiated. Balaji Prabhakar thanks Anna Charny for conversations regarding MUCFA and its implementational complexity.

References

[1] A. Charny: "Providing QoS Guarantees in Input Buffered Crossbar Switches with Speedup", PhD Thesis, MIT, 1998.

[2] A. Charny, P. Krishna, N. Patel and R. Simcoe: "Algorithms for Providing Bandwidth and Delay Guarantees in Input-buffered Crossbars with Speedup", *Presented at the 6th IEEE/IFIP IWQ0S '98*, May 1998.

[3] S-T. Chuang, A. Goel, N. McKeown and B. Prabhakar: " Matching Output Queueing with a Combined Input Output Queued Switch", to appear in *IEEE JSAC*. A short version appears in *The Proceedings of Infocom '99.*

[4] C-Y. Chang, A.J. Paulraj and T. Kailath: "A Broadband Packet Switch Architecture with Input and Output Queueing", *Proc. Globecom '94*, pp.448-452.

[5] J.S.-C. Chen and T.E. Stern, "Throughput analysis, optimal buffer allocation, and traffic imbalance study of a generic nonblocking packet switch," IEEE J. Select. Areas Commun., Apr. 1991, vol. 9, no. 3, pp. 439-49.

[6] D. Gale; L.S. Shapley; "College Admissions and the stability of marriage", *American Mathematical Monthly*, vol.69, pp.9-15, 1962.

[7] A.L. Gupta and N.D. Georganas, "Analysis of a packet switch with input and output buffers and speed constraints," *Proc. Infocom '91*, Bal Harbour, FL, April 1991, pp.694-700.

[8] I. Iliadis and W.E. Denzel: "Performance of packet switches with input and output queueing," *Proc. ICC '90*, Atlanta, GA, April 1990. pp.747-53.

[9] M. Karol, M. Hluchyj and S. Morgan: "Input Versus Output Queueing on a Space Division Switch", *IEEE Trans. Comm*, vol.35, no.12, pp.1347-1356.

[10] P. Krishna, N. Patel, A. Charny and R. Simcoe: "On the Speedup Required for Work-conserving Crossbar Switches" *Presented at the 6th IEEE/IFIP IWQ0S '98*, May 1998. To appear in *IEEE JSAC.*

[11] W.E. Leland; W. Willinger; M. Taqqu; D. Wilson; "On the self-similar nature of Ethernet traffic", *Proc. of Sigcomm*, San Francisco, pp.183-193. Sept 1993.

[12] N. McKeown, V. Anantharam, J. Walrand: "Achieving 100% Throughput in an Input-Queued Switch", *INFOCOM '96*, pp.296-302.

[13] McKeown, N.; "Scheduling Algorithms for Input-Queued Cell Switches", *PhD Thesis, University of California at Berkeley*, May 1995.

[14] Y. Oie; M. Murata, K. Kubota, and H. Miyahara, "Effect of speedup in nonblocking packet switch," *Proc. ICC '89*, Boston, MA, June 1989, pp. 410-14.

[15] C. Partridge, et al. "A fifty gigabit per second IP router," To appear in *IEEE/ACM Transactions on Networking*.

[16] V. Paxson; S. Floyd; "Wide area traffic: the failure of Poisson modeling." *IEEE/ACM Transactions on Networking* Vol.3, no.3, pp.226-44. June 1995.

[17] Cisco Systems GSR 12000 Technical Product Description, http://www.cisco.com/warp/public/733/12000/index.shtml.