Matching Output Queueing with a Combined Input Output Queued Switch[1]

Shang-Tse Chuang
Ashish Goel
Nick McKeown
Balaji Prabhakar
Stanford University

Abstract — *The Internet is facing two problems simultaneously: there is a need for a faster switching/routing infrastructure, and a need to introduce guaranteed qualities of service (QoS). Each problem can be solved independently: switches and routers can be made faster by using input-queued crossbars, instead of shared memory systems; and QoS can be provided using WFQ-based packet scheduling. However, until now, the two solutions have been mutually exclusive — all of the work on WFQ-based scheduling algorithms has required that switches/routers use output-queueing, or centralized shared memory. This paper demonstrates that a Combined Input Output Queueing (CIOQ) switch running twice as fast as an input-queued switch can provide precise emulation of a broad class of packet scheduling algorithms, including WFQ and strict priorities. More precisely, we show that for an* $N \times N$ *switch, a "speedup" of* $2 - 1/N$ *is necessary and a speedup of two is sufficient for this exact emulation. Perhaps most interestingly, this result holds for all traffic arrival patterns. On its own, the result is primarily a theoretical observation; it shows that it is possible to emulate purely OQ switches with CIOQ switches running at approximately twice the line-rate. To make the result more practical, we introduce several scheduling algorithms that, with a speedup of two, can emulate an OQ switch. We focus our attention on the simplest of these algorithms, Critical Cells First (CCF), and consider its running-time and implementation complexity. We conclude that additional techniques are required to make the scheduling algorithms implementable at high speed, and propose two specific strategies.*

1 Introduction

Many commercial switches and routers today employ output-queueing.[2] 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 waits 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. Furthermore, by carefully scheduling the time 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

1. This paper was presented at Infocom '99, New York, USA.

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

impractical for switches with high line rates and/or with a large number of ports, since the fabric and memory of an $N \times N$ switch must run N times as fast as the line rate. Unfortunately, at high line rates, memories with sufficient bandwidth are simply not available.

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. For this reason, the highest performance switches and routers use input-queued crossbar switches [3][4]. But IQ switches can suffer from 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% [5].

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 one. 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.

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 [6][7][8][9]. 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. Whereas these studies consider *average* delay (and simplistic input traffic patterns), they make no guarantees about the delay of individual packets. This is particularly important if a switch or router is to offer QoS guarantees.

We believe that a well-designed network switch should perform predictably in the face of all types of arrival process[1] and allow the delay of individual packets to be controlled. Hence 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.) This approach was first formulated in the recent work of Prabhakar and McKeown [12]. They show that a CIOQ switch with a speedup of four can behave identically to a FIFO OQ switch for arbitrary input traffic patterns and switch sizes. In this sense, this paper builds upon and extends the results in [12], as described in the next paragraph. A number of researchers have recently considered various aspects of the speedup problem, most notably [18] which obtains packet delay bounds and [19] which finds sufficient conditions for maximizing throughput through work conservation and mimicking of output queueing.[2]

In this paper, we show that a CIOQ switch with a speedup of two can behave identically to an OQ switch. The result holds for switches with an arbitrary number of ports, and for any traffic

1. 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 network traffic processes [11]. 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.

2. [20] aimed to extend the results of [12], but the algorithms and proofs presented there are incorrect and do not solve the speedup problem. See http://www.cs.cmu.edu/~istoica/IWQoS98-fix.html for a discussion of the errors.

arrival pattern. It is also found to be true for a broad class of widely used output link scheduling algorithms such as weighted fair queueing, strict priorities, and FIFO. We introduce some specific scheduling algorithms that achieve this result. We also show more generally that a speedup of $2 - \frac{1}{N}$ is both necessary and sufficient for a CIOQ switch to behave identically to a FIFO OQ switch.

It is worth briefly considering the implications of this result. It demonstrates that it is possible to emulate an $N \times N$ OQ switch using buffer memory operating at only twice the speed of the external line. Previously, an OQ switch could only be implemented with memories operating at N times the speed of the external line. However, the advantages do not come for free. In essence, the memory bandwidth is reduced at the expense of a fast cell scheduling algorithm that is required to configure the crossbar. As we shall see, the scheduling algorithms are complex, the best known-to-date having a running-time complexity of N. (We discuss the implementation complexity in some detail in Section 5). This means that it is not yet practicable to emulate fast OQ switches with a large number of ports. While we propose some strategies in this paper, this is a topic for further research.

1.1 Background

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 [4][3]. We take the arrival time between cells as the basic time unit and refer to it as a *time slot*. The switch is said to have a *speedup of* S, for $S \in \{1, 2, \ldots, N\}$ 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 switch fabric to run S times as fast as the input or output line rate. 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 and an appropriate cell scheduling algorithm π that

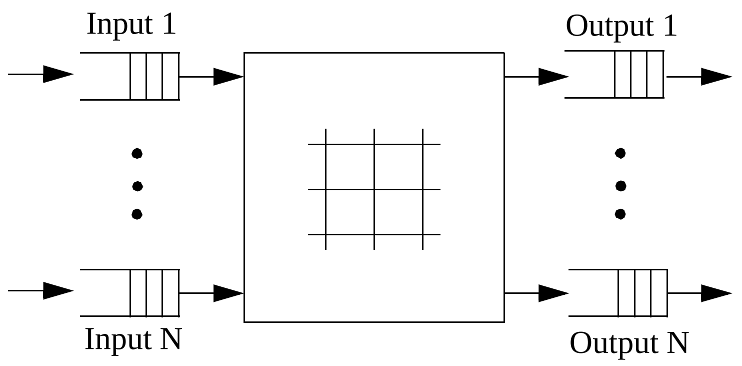
1. allows a CIOQ switch to exactly mimic the performance of an output-queued switch (in a sense that will be made precise),
2. achieves this for any *arbitrary* input traffic pattern,
3. is independent of switch size.

In an OQ switch, arriving cells are immediately forwarded to their corresponding outputs. This (a) ensures that the switch is *work-conserving*, i.e. an output never idles so long as there is a cell destined for it in the system, and (b) allows the departure of cells to be scheduled to meet latency constraints.[1] We will require that any solution of the speedup problem possess these two desirable features; that is, a CIOQ switch must *behave identically* to an OQ switch in the following sense:

1. For ease of exposition, we will at times assume that the output uses a FIFO queueing discipline, i.e. cells depart from the output in the same order that they arrived to the inputs of the switch. However, we are interested in a broader class of queueing disciplines: ones that allow cells to depart in time to meet particular bandwidth and delay guarantees.

Identical Behavior: 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.

Figure 1: A General Combined Input and Output Queued (CIOQ) switch.



As a benchmark with which to compare our CIOQ switch, we will assume there exists a shadow $N \times N$ OQ switch that is fed the same input traffic pattern as the CIOQ switch. Our goal is to arrange for each cell to depart from the CIOQ switch at exactly the same time as its counterpart cell departs from the OQ switch. In the CIOQ switch, the sequence in which cells are transferred from their input queues to the output queue is determined by a scheduling algorithm. In each time slot, the scheduling algorithm matches each non-empty input with at most one output and, conversely, each output is matched with at most one input. The matching is used to configure the crossbar fabric 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 during each time slot.

Selecting the appropriate scheduling algorithm is the key to achieving identical behavior between the CIOQ switch and its shadow OQ switch.

1.2 Push-in Queues

Throughout this paper, we will make repeated use of what we will call a *push-in queue*. Similar to a discrete-event queue, a push-in queue is one in which an arriving cell is inserted at an arbitrary location in the queue based on some criterion. For example, each cell may carry with it a departure time, and is placed in the queue ahead of all cells with a later departure time, yet behind cells with an earlier departure time. The only property that defines a push-in queue is that once placed in the queue, cells may not switch places with other cells. In other words, their relative ordering remains unchanged. In general, we distinguish two types of push-in queues: (1) “Push-In First-Out” (PIFO) queues, in which arriving cells are placed at an arbitrary location, and the cell at the head of the queue is always the next to depart. PIFO queues are quite general — for example, a WFQ scheduling discipline operating at an output queued switch is a special case of a PIFO queue. (2) “Push-In Arbitrary-Out” (PIAO) queues, in which cells are removed from the queue in an arbitrary order. i.e. it is not necessarily the case that the next cell to depart is the one currently at the head of the queue.

It is assumed that each input of the CIOQ switch maintains a queue, which can be thought of as an ordered set of cells waiting at the input port. In general, the CIOQ switches that we consider, can all be described using PIAO input queues.[1] Many orderings of the cells are possible — each ordering leading to a different switch scheduling algorithm, as we shall see.

Each output maintains a queue for the cells waiting to depart from the switch. In addition, each output also maintains an *output priority list*: an ordered list of cells at the inputs waiting to be transferred to this particular output. The output priority list is drawn in the order in which the cells would depart from the OQ switch we wish to emulate (i.e. the shadow OQ switch). This priority list will depend on the queueing policy followed by the OQ switch (FIFO, WFQ, strict priorities etc.).

1.3 Definitions

The following definitions are crucial to the rest of the paper.

Definition 1: Time to Leave — *The "time to leave" for cell c, TL(***c***), is the time slot at which* **c** *will leave the shadow OQ switch. Note that it is possible for TL(c) to increase. This happens if new cells arrive to the switch, destined for c's output, and have a higher priority than c. (Of course, TL(***c***) is also the time slot in which c must leave from our CIOQ switch for the identical behavior to be achieved.)*

Definition 2: Output Cushion — *At any time, the "output cushion of a cell c", OC(***c***), is the number of cells waiting in the output buffer at cell* **c***'s output port with a smaller time to leave value than cell* **c***.*

Notice that if a cell is still on the input side and has a small (or zero) output cushion, the scheduling algorithm must urgently deliver the cell to its output so that it may depart on time. Since the switch is work-conserving, a cell's output cushion decreases by one during every time slot, and can only be increased by newly arriving cells that are destined to the same output and have a more urgent time to leave.

Definition 3: Input Thread — *At any time, the "input thread of cell c", IT(***c***), is the number of cells ahead of cell* **c** *in its input priority list.*

In other words, *IT(***c***)* represents the number of cells currently at the input that need to be transferred to their outputs more urgently than cell **c**. A cell's input thread is decremented only when a cell ahead of it is transferred from the input, and is possibly incremented by newly arriving cells. Notice that it would be undesirable for a cell to simultaneously have a large input thread and a small output cushion — the cells ahead of it at the input may prevent it from reaching its output before its time to leave. This motivates our definition of *slackness*.

Definition 4: Slackness — *At any time, the "slackness of cell c", L(***c***), equals the output cushion of cell* **c** *minus its input thread i.e.* $L(\mathbf{c}) = OC(\mathbf{c}) - IT(\mathbf{c})$.

Slackness is a measure of how large a cell's output cushion is with respect to its input thread. If a cell's slackness is small, then it urgently needs to be transferred to its output. Conversely, if a cell has a large slackness, then it may languish at the input without fear of missing its time to leave.[1]

Figure 2: A snapshot of a CIOQ switch

1. In practice, we need not necessarily use a PIAO queue to implement these techniques. But we will use the PIAO queue as a general way of describing the input queueing mechanism.

1. Note that a cell's input thread and slackness are only defined when the cell is waiting at the input side of the switch.

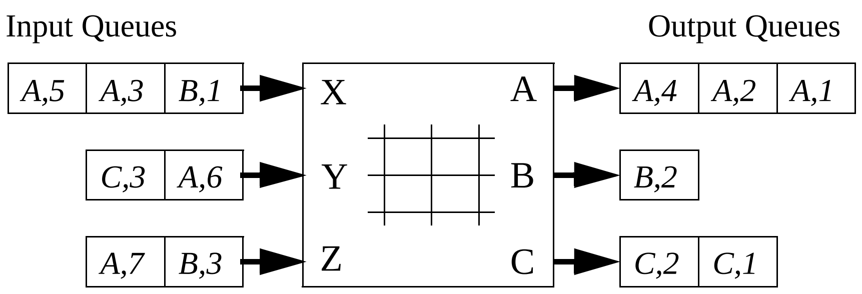


Figure 2 shows a snapshot of a CIOQ switch with a number of cells waiting at its inputs and outputs. For convenience we assume the time the snapshot was taken to be 1. Let (P, t) denote a cell that, in the shadow switch, will depart from output port P at time t. Consider, for example, the cell **c** denoted in the figure by $(A, 3)$. For the CIOQ switch to mimic the shadow OQ switch, the cell must depart from port *A* at time 3. Its input thread is $IT(\mathbf{c}) = 1$, since $(B, 1)$ is the only cell ahead of **c** in the input priority list. Its output cushion is $OC(\mathbf{c}) = 2$, since out of the three cells queued at *A*'s output buffer, only two cells $(A, 1)$ and $(A, 2)$ will depart before it. Further, the slackness of cell **c** is given by $L(\mathbf{c}) = OC(\mathbf{c}) - IT(\mathbf{c}) = 1$.

1.4 The general structure of our CIOQ scheduling algorithms:

For most of this paper we are going to concern ourselves with CIOQ switches that have a speedup of two. Hence, we will break each time slot into four phases:

1. **The Arrival Phase**
   All arrivals of new cells to the input ports take place during this phase.
2. **The First Scheduling Phase**
   The scheduling algorithm selects cells to transfer from inputs to outputs, and then transfers them across the crossbar.
3. **The Departure Phase**
   All departures of cells from the output ports take place during this phase.
4. **The Second Scheduling Phase**
   Again, the scheduling algorithm selects cells to transfer from inputs to outputs and transfers them across the crossbar.

The order in which the four phases occur is not crucial to our algorithms. However we shall stick to the above ordering as it makes our proofs simpler.

A *matching* of input ports to output ports is a (not necessarily maximal) set of cells waiting on the input side such that all these cells can be sent across the crossbar in a single transfer (i.e. are free of input and output contention). During each scheduling phase the scheduler finds a *stable* matching between the input ports and the output ports.

Definition 5: **Stable Matching** — *A matching of input ports to output ports is said to be stable if for each cell* **c** *waiting in an input queue, one of the following holds:*

1. Cell **c** is part of the matching, i.e. **c** will be transferred from the input side to the output side during this phase.

2. A cell that is ahead of **c** in its input priority list is part of the matching.

3. A cell that is ahead of **c** in its output priority list is part of the matching.

Notice that conditions 2 and 3 above may be simultaneously satisfied, but condition 1 excludes the other two. The conditions for a stable matching can be achieved using the so-called *stable marriage problem*. Solutions to the stable marriage problem are called stable matchings and were first studied by Gale and Shapely [13]— they gave an algorithm that finds a stable matching in at most M iterations, where M is the sum of the lengths of all the input priority lists.

Our specification of the scheduling algorithm for a CIOQ switch is almost complete: the only thing that remains is to specify how the input queues are maintained. Different ways of maintaining the input queues result in different scheduling algorithms. In fact, the various scheduling algorithms presented later differ *only* in the ordering of their input queues. For reasons that will become apparent, we will restrict ourselves to a particular class of orderings, which is defined as follows.

Definition 6: PIAO Input Queue Ordering — *When a cell arrives, it is given a priority number which dictates its position in the queue. i.e. a cell with priority number X is placed at location (X+1) from the head of the list. A cell is placed in an input priority list according to the following rules:*

1. Arriving cells are placed at (or, "push-in" to) an arbitrary location in the queue,

2. The relative ordering of cells in the queue does not change once cells are in the queue, i.e. cells in the queue cannot switch places, and

3. Cells may be selected to depart from the queue from any location.

Thus, to complete our description of the scheduling algorithms, we need only specify an insertion policy which determines where an arriving cell gets placed in its input queue.

On the output side, the CIOQ switch keeps track of the time to leave of each waiting cell. During each time slot the cell that departs from an output and is placed onto the outgoing line is the one with the smallest time to leave. For the CIOQ switch to successfully mimic the shadow OQ switch, we must ensure that each cell crosses over to the output side before it is time for the cell to leave.

Even before we finish defining the algorithm, we can already see that it must maintain a large amount of state. More importantly, the algorithm must keep track of a large amount of global state, taking into account information about the queues at all the inputs and all the outputs. We will discuss in Section 5 the information complexity of these algorithms, and the difficulty of implementing them at high speed.

2 Necessity and Sufficiency of a Speedup of 2-1/N

Having defined speedup, we now address the next natural question: what is the minimum required speedup, S, for a CIOQ switch to emulate an OQ switch. The following theorem answers this question.

Theorem 7: *(Necessity). An $N \times N$ CIOQ switch needs a speedup of at least $2 - \frac{1}{N}$ to exactly emulate an $N \times N$ FIFO OQ switch.*

Proof: The proof is by counter-example and is presented in Appendix A.

Remark: Since FIFO is a special case of a variety of output queueing disciplines (Weighted Fair Queueing, Strict Priorities etc.), the lower bound applies to these queueing disciplines as well.

Theorem 8: *(Sufficiency). An $N \times N$ CIOQ switch with a speedup of $2 - \frac{1}{N}$ can exactly emulate an $N \times N$ FIFO OQ switch.*

Proof: The proof is based on an insertion policy that we call Last In Highest Priority (LIHP) and is presented in Appendix B.

3 A Simple Input Queue Insertion Policy for a Speedup of 2

The proof of Theorem 8 is based on the LIHP input queue insertion policy and — unfortunately — the proof is complex and somewhat counterintuitive. Further, LIHP is complex to implement, making it of little practical value. So in an attempt to provide a more intuitive understanding of the speedup problem, we present a simple and slightly more practical insertion policy that, with a speedup of two, mimics an OQ switch with a FIFO queueing discipline. We call this insertion policy Critical Cells First (CCF).

Recall that to specify a scheduling algorithm for a CIOQ switch, we just need to give an insertion policy for the input queues. “Critical Cells First” (CCF) inserts an arriving cell as far from the head of its input queue as possible, such that the input thread of the cell is not larger than its output cushion. More formally:

The CCF Insertion Policy: Suppose cell **c** arrives at input port P. Let X be the output cushion of **c**. Insert cell **c** into the $(X+1)^{\text{th}}$ position from the front of the input queue at P. Hence, upon arrival cell **c** has a slackness of zero. If the size of this list is less than X cells, then place **c** at the end of the input priority list at P. Hence, in this case, **c** has a positive slackness.

A consequence of this is that a cell’s slackness is non-negative upon arrival. The intuition behind this insertion policy is that a a cell with a small output cushion needs to leave soon (i.e. it is “more critical”), and therefore needs to be delivered to its output sooner than a cell with a larger output cushion. In other words, a cell with a large output cushion can safely reside further from the head of its input queue.

We now prove that CCF, with a speedup of two, mimics an OQ switch. Informally, the proof proceeds as follows. We first show a property of the CCF algorithm: that a cell never has a negative slackness, i.e. a cell’s input thread never exceeds its output cushion. We then proceed to show how this ensures that a cell always reaches the output side in time to leave.

Lemma 1: *The slackness, L, of a cell* **c** *waiting on the input side is non-decreasing from time slot to time slot.*

Proof: Let the slackness of **c** be L at the beginning of a time slot. During the arrival phase, the input thread of **c** can increase by at most one because an arriving cell might be inserted ahead of **c** in its input priority list. During the departure phase, the output cushion of **c** decreases by one. If **c** is scheduled in any one of the scheduling phases, then it is delivered to its output and we need no longer concern ourselves with **c**. Otherwise, during each of the two scheduling phases, either the input thread of **c** decreases by one, or the output cushion of **c** increases by one (by the property of stable matchings — see Definition 5). Therefore the slackness of **c** increases by at least one during each scheduling phase. Counting the changes in each of the four phases (arrival, departure, and two scheduling phases), we conclude that the slackness of cell **c** can not decrease from time slot to time slot.

Remark: Because the slackness of an arriving cell is non-negative, it follows from Lemma 1 that the slackness of a cell is always non-negative.

Theorem 9: *Regardless of the incoming traffic pattern, a CIOQ switch that uses CCF with a speedup of 2 exactly mimics a FIFO OQ switch.*

Proof: Suppose that the CIOQ switch has successfully mimicked the OQ switch up until time slot $t-1$, and consider the beginning (first phase) of time slot t. We must show that any cell reaching its time to leave is either: (1) already at the output side of the switch, or (2) will be transferred to the output during time slot *t*. From Lemma 1, we know that a cell always has a non-negative slackness. Therefore, when a cell reaches its time to leave (i.e. its output cushion has reached zero), the cell's input thread must also equal zero. This means either: (1) that the cell is already at its output, and may depart on time, or (2) that the cell is simultaneously at the head of its input priority list (because its input thread is zero), and at the head of its output priority list (because it has reached its time to leave). In this case, the stable matching algorithm is guaranteed to transfer it to its output during the time slot, and therefore the cell departs on time.

4 Providing QoS guarantees

As pointed out in the introduction, the goal of our work is to control the delay of cells in a CIOQ switch in the same way that is possible in an OQ switch. But until now, we have considered only the emulation of an OQ switch in which cells depart in FIFO order. We now show that, with a speedup of two, CCF can be used to emulate an OQ switch that uses the broad class of PIFO (Push-In First-Out) queueing policies; a class that includes widely-used queueing policies such as WFQ and Strict Priority queueing. Notice that with an arbitrary PIFO policy, the TL of a cell never decreases, but may increase as a result of arrival of higher priority cells.

The description of CCF remains unchanged; however the output cushion and the output priority lists are calculated using the OQ switch that we are trying to emulate.

Theorem 10: *Regardless of the incoming traffic pattern, a CIOQ switch that uses CCF with a speedup of 2 exactly mimics an OQ switch that adheres to a PIFO queueing policy.*

The proof of Theorem 10 is almost identical to that of Theorem 9, and is omitted here for brevity.

5 Towards making CCF practical

CCF as presented above suffers from two main disadvantages. First, the stable matching that we need to find in each scheduling phase can take as many as N^2 iterations.[1] Second, the algorithm has a high information complexity — CCF needs to know both the output cushion and time to leave of each cell at the inputs; information that is not locally available at each input, but

1. It is not immediately obvious that N^2 iterations suffice. The reason for this is that if two cells at the same input port are destined to the same output port, the one with the lower TL occurs ahead of the other in the input priority list.

depends on the state of all the switch outputs. We address these disadvantages in this section. The Delay Till Critical (DTC) strategy reduces the number of iterations needed to compute a stable matching to N (from N^2); and the Group By Virtual Output Queue (GBVOQ) algorithm can emulate FIFO and WFQ OQ switches without using global information. Unfortunately, the two solutions do not compose i.e. we can reduce either the number of iterations to N or reduce the information complexity, but not both at the same time. We leave such a composition as an open problem.

5.1 The Delay Till Critical (DTC) strategy:

The "Delay Till Critical" strategy is as follows: *During each scheduling phase, mark as "active" all cells with a slackness of zero, and mark all other cells inactive.* The stable matching algorithm now considers only active cells. Intuitively, cells with zero slackness are the most critical and should be considered for immediate transfer across the fabric. Since the slackness of a cell can never become negative,[1] CCF combined with DTC strategy can emulate any OQ switch that follows a PIFO queueing policy.

It remains to show that this simple strategy reduces the number of iterations required to compute a stable matching from N^2 to N. Before we prove this fact, let us examine the problem that we are trying to remove. At any time instant, we define the dependency graph G to be a directed graph with a vertex corresponding to each active cell that is waiting on the input side of the CIOQ switch. Let A and B be two cells waiting at the input side. There is a directed edge from B to A if and only if cell A is ahead of B either in an input queue or in an output priority list. Clearly two cells have to share either the same input port or the same output port if there is to be an edge between them. If we use CCF as defined in Section 3, there may be cycles in this dependency graph. These cycles are the main cause of inefficiency in finding stable matchings, and the DTC strategy is designed to remove these cycles.

Lemma 2: *If DTC is used in conjunction with CCF then, during any scheduling phase, the dependency graph G is acyclic.*

We defer the proof of Lemma 2 to Appendix C and instead focus on its implications, and how a match can be constructed in N iterations. First, let's consider how the first cell in the match is found. Since there are no cycles in G, there has to be at least one "sink" (i.e. a cell with no outgoing edges). Let cell X be the sink. Since there are no active cells ahead of X in either its input queue or its output priority list, cell X has to be part of any stable matching of active cells. Hence X is guaranteed to be transferred to the output side, and therefore we can remove from the graph all cells which have the same input or output port as X; they clearly can not be part of the match. The resulting graph is again acyclic, and we can repeat the above procedure $N-1$ more times to obtain a stable matching. Notice that each iteration of the above N iteration algorithm is quite straightforward.

We now address the second disadvantage of CCF, i.e. that of high information complexity.

5.2 The Group By Virtual Output Queue (GBVOQ) algorithm:

With CCF, the stable matching algorithm needs to calculate both the time to leave and output cushion of each cell in the input queues. These quantities require centralized information about the

1. As soon as the slackness becomes zero, the cell would be marked active and the slackness would increase by one during the current scheduling phase (see Lemma 1).

state of all the queues in the system, making CCF (as described) unsuitable for a distributed implementation. However, for emulating a FIFO OQ switch, we can group incoming cells into Virtual Output Queues and obtain an upper bound of N on the number of cells that need to be considered. The algorithm, GBVOQ, which achieves this bound is described below.

At each input, GBVOQ maintains a single priority list as before, as well as a VOQ for each output port. All cells belong to a VOQ and the single input priority list. When a new cell arrives, it is always placed at the tail of the corresponding VOQ. If the VOQ was empty, the new cell is placed at the head of the input priority list. If, on the other hand, the VOQ is non-empty, the new cell is inserted in the input priority list just behind the last cell belonging to the same VOQ. i.e. all cells that are in the same VOQ occupy contiguous positions in the input priority list. Therefore, to make a scheduling decision, it is sufficient to just keep track of the relative priority ordering of VOQs. Since there are at most N VOQs at an input port in a FIFO switch, the size of the input priority list is bounded. Since GBVOQ assigns a non-negative slackness to an incoming cell, a CIOQ switch that uses GBVOQ with a speedup of two successfully emulates a FIFO OQ switch.

Apart from small priority lists, GBVOQ has other desirable properties. First, the decision of where an incoming cell needs to be inserted is much simpler for GBVOQ than CCF — each input port can maintain its local priority queue without any access to global information. Second, during the stable matching phase, to determine which of two cells has a higher output priority, we just need to compare the arrival timestamps of the two cells; the cell which arrived earlier will have a smaller TL (and hence a higher output priority) because of the First In First Out property.

Like CCF, GBVOQ can be used in conjunction with the DTC strategy to reduce the number of iterations needed to compute a stable matching. In fact, DTC is made much simpler when used in conjunction with GBVOQ because of the following property: if the cell at the head of a VOQ is marked inactive during a scheduling phase, the entire VOQ can be marked inactive, reducing the number of cells that need to be marked active/inactive. However, and unfortunately, to determine which VOQs need to be marked active, we again need access to global state, namely the output cushion of each cell at the head of a VOQ. Finding a solution which simultaneously has low information complexity and low number of iterations is an interesting open problem.

GBVOQ with per-flow VOQs can emulate WFQ with low information complexity. However, we have not been able to show a bound of even N^2 on the number of iterations for this case.

6 Extensions to Multicasting

We now briefly discuss the speedup required for the exact emulation of an OQ switch by a CIOQ switch when the incoming traffic is multicast. Suppose that the maximum “fanout” of an incoming cell (i.e., the maximum number of outputs it wishes to go to) is *F*. Then by replicating this cell when it arrives at the input we get up to *F* new arrivals in each time slot at that input. (This contrasts with the unicast case $F = 1$, in which at most 1 new cell arrives). Thus, to extend Theorem 9 to include multicast traffic, we must allow for up to *F* new arrivals to take place per time slot. During a departure phase, the slackness of a cell can go down by at most 1. During an arrival phase, the slackness of a cell already in the system can go down by at most *F* since a new cell with fanout *F* may get inserted ahead of it. Recall, from the proof of Lemma 1, that the slackness of a cell must go up by at least 1 during each scheduling phase. Therefore an equivalent of Lemma 1 (and hence Theorem 9) holds for multicast if the speedup is $F + 1$.

7 Conclusions

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 will 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.

We have seen that a CIOQ switch with a speedup of just two can behave identically to an OQ switch which employs a wide variety of packet scheduling algorithms, such as WFQ, strict priorities, etc. Perhaps more importantly, we show this to be true for any traffic arrival pattern and for arbitrary switch sizes.

However, while this result makes possible a significant reduction in memory bandwidth, it comes at the expense of a scheduling algorithm. The scheduling algorithm is required to configure the crossbar, operating at least twice as fast as cells can arrive. While the algorithms that we describe here are quite simple, they have a running time complexity of at least $O(N)$ making them unsuitable for fast switches with a large number or ports. But the result does not preclude algorithms that are more readily implemented at higher speed. We believe this to be an important area for future research.

8 Acknowledgements

Balaji Prabhakar thanks Anna Charny for discussions on the speedup problem, the subject of her PhD thesis [21], where it was pointed out that speedup could be used to provide delay guarantees for QoS-constrained flows.

9 References

[1] A. Demers, S. Keshav; S. Shenker, “Analysis and Simulation of a Fair Queueing Algorithm,” J. of Internetworking: Research and Experience, pp.3-26, 1990.

[2] L. Zhang, “Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks,” ACM Transactions on Computer Systems, vol.9 no.2, pp.101-124, 1990.

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

[4] McKeown, N.; Izzard, M.; Mekkittikul, A.; Ellersick, W.; and Horowitz, M.; “The Tiny Tera: A Packet Switch Core” *Hot Interconnects V, Stanford University,* August 1996.

[5] M. Karol; M. Hluchyj; S. Morgan, “Input versus output queueing on a space-division switch,” IEEE Transactions on Communications, vol. 35, pp. 1347-1356, Dec 1987.

[6] I. Iliadis and W.E. Denzel, “Performance of packet switches with input and output queueing,” in Proc. ICC ‘90, Atlanta, GA, Apr. 1990. p.747-53.

[7] A.L. Gupta and N.D. Georganas, “Analysis of a packet switch with input and output buffers and speed constraints,” in Proc. InfoCom ‘91, Bal Harbour, FL, Apr. 1991, p.694-700.

[8] Y. Oie, M. Murata, K. Kubota, and H. Miyahara, “Effect of speedup in nonblocking packet switch,” in Proc. ICC ‘89, Boston, MA, Jun. 1989, p. 410-14.

[9] 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, p. 439-49.

[10] N. McKeown; V. Anantharam; J. Walrand, “Achieving 100% Throughput in an input-queued switch,” Infocom ‘96.

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

[12] B. Prabhakar and N. McKeown, “On the Speedup Required for Combined Input and Output Queued Switching.” Stanford University Technical Report, STAN-CSL-TR-97-738. November 1997.

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

[14] M.Andrews, B.Awerbuch, A. Fernandez, J. Kleinberg, T. Leighton, and Z. Liu. “Universal stability results for greedy contention-resolution protocols.” *37th IEEE symposium on Foundations of Computer Science*, pp. 380-389 (1996).

[15] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. Williamson. “Adversarial queueing theory.” *28th ACM Symposium on Theory of Computing,* p. 376-385 (1996).

[16] A. Goel. “Stability of Networks and Protocols in the Adversarial Queueing Model for Packet Routing.” Stanford University Technical Note STAN-CS-97-59.

[17] T. Feder, N. Megiddo, and S. Plotkin. “A sublinear parallel algorithm for stable matching.” *Fifth ACM-SIAM Symposium on Discrete Algorithms*, p. 632-637 (1994).

[18] A. Charny, P. Krishna, N. Patel, and R. Simcoe. “Algorithms for Providing Bandwidth and Delay Guarantees in Input-Buffered Crossbars with Speed Up.” *Presented at 6th IEEE/IFIP IWQoS ‘98, Napa, California.* May 1998.

[19] P. Krishna, N. Patel, A. Charney , and R. Simcoe. “On the Speedup Required for Work-Conserving Crossbar Switches” *Presented at 6th IEEE/IFIP IWQoS ‘98, Napa, California.* May 1998.

[20] I. Stoica, and H. Zhang. “Exact Emulation of an Output Queueing Switch by a Combined Input Output Queueing Switch.” *Presented at 6th IEEE/IFIP IWQoS ‘98, Napa, California.* May 1998.

[21] A. Charny. “Providing QoS Guarantees in Input-Buffered Crossbar Switches with Speedup”, Ph.D. dissertation, August 1998, MIT.

Appendix A: The Necessity of a Speedup of 2-1/N

With a speedup of two, the above algorithms (CCF and GBVOQ) exactly mimic an arbitrary size OQ switch. The next natural question to ask is whether it is possible to emulate output queueing using a CIOQ switch with a speedup less than 2. In this section we show a lower bound of $2 - \frac{1}{N}$ on the speedup of any CIOQ switch that emulates OQ switching, even when the OQ switch uses FIFO. Hence the algorithms that we have presented in this paper are almost optimal. In fact, the difference of $\frac{1}{N}$ can be ignored for all practical purposes.

Since a speedup between 1 and 2 represents a non-integral distribution of phases, we first describe how scheduling phases are distributed. A speedup of $2 - \frac{1}{N}$ corresponds to having a *truncated* time slot out of every N time slots; the truncated time slot has just one scheduling phase, whereas the other $N - 1$ time slots have two scheduling phases each. In Figure 3, we show the difference between one-phased and two-phased time slots. For the purposes of our lower bound, we need to assume that the scheduling algorithm does not know in advance whether a time slot is truncated.

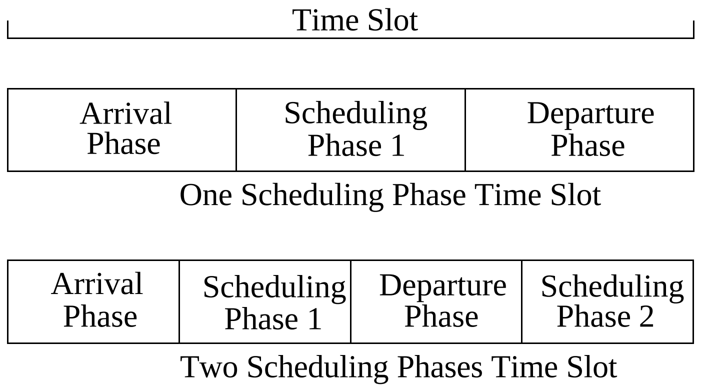


Figure 3: One scheduling phase and two scheduling phase time slots.

Recall from Section 3 that a cell is represented as P-TL, where P represents which output port the cell is destined to, and TL represents the time to leave for the cell. For example, the cell C-7 must be scheduled for port C before the end of time slot 7.

The input traffic pattern that provides the lower bound for an $N \times N$ CIOQ switch is given below. The traffic pattern spans N time slots, the last of which is truncated.

1. In the first time slot, all input ports receive cells destined for the same output port, P_1.
1. In the second time slot, the input port that had the lowest time to leave in the previous time slot does not receive any more cells. In addition, the rest of the input ports receive cells destined for the same output port, P_2.
1. In the i^{th} time slot, the input ports that had the lowest time to leave in each of the $i-1$ previous time slots do not receive any more cells. In addition, the rest of the input ports must receive cells destined for the same output port, P_i.

We can repeat the above traffic pattern as many time as required to create arbitrarily long traffic patterns. In Figure 4, we show the above sequence of cells for a 4×4 switch. The departure

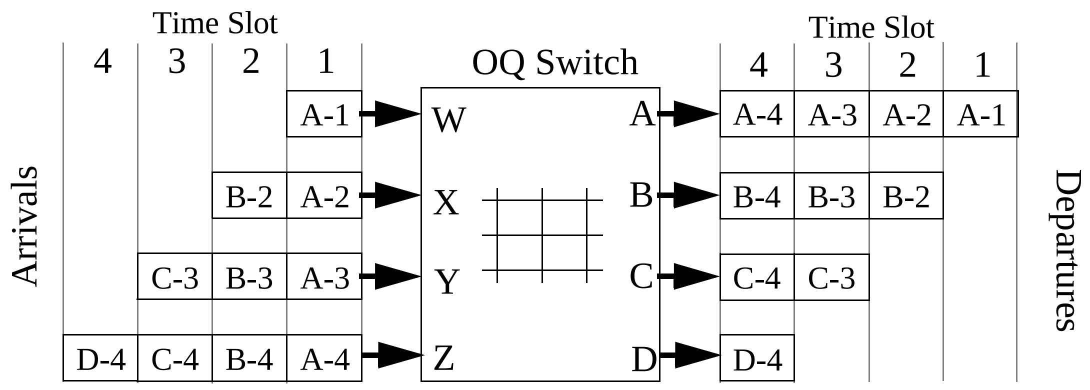


Figure 4: Lower bound Input Traffic Pattern for a 4x4 switch.

events from the OQ switch are depicted on the right, and the arrival events are on the left. For simplicity, we present the proof of our lower bound on this 4×4 switch instead of a general $N \times N$ switch.

Figure 5 shows the only possible schedule for transferring these cells across in seven phases.

Phase	PA	PB	PC	PD
1	**A-1**			
2				
3		**B-2**		
4				
5			**C-3**	
6				
7				**D-4**

(a)

Phase	PA	PB	PC	PD
1	A-1			
2	**A-2**			
3		B-2		
4		**B-3**		
5			C-3	
6			**C-4**	
7				D-4

(b)

Phase	PA	PB	PC	PD
1	A-1			
2	A-2			
3	**A-3**	B-2		
4		B-3		
5		**B-4**	C-3	
6			C-4	
7				D-4

(c)

Phase	PA	PB	PC	PD
1	A-1			
2	A-2			
3	A-3	B-2		
4	**A-4**	B-3		
5		B-4	C-3	
6			C-4	
7				D-4

(d)

Figure 5: Scheduling Order for the lower bound input traffic pattern in Figure 4.

Of the four time slots, the last one is truncated, giving a total of seven phases. Cell A-1 must leave the input side during the first phase, since the CIOQ switch does not know whether the first time slot is truncated. Similarly, cells B-2, C-3, and D-4 must leave during the third, fifth, and seventh phases, respectively (see Figure 5(a)). Cell A-2 must leave the input side by the end of the third phase. But it cannot leave during the first or the third phase because of contention. Therefore, it must depart during the second phase. Similarly, cells B-3 and C-4 must depart during the fourth and sixth phases, respectively (see Figure 5(b)). Continuing this elimination process (Figure 5(c), (d)), there is only one possible scheduling order. For this input traffic pattern, the switch needs all seven phases in four time slots which corresponds to a minimum speedup of $\frac{7}{4}$ (or $2 - \frac{1}{4}$).

Theorem 11: *A minimum speedup of* $2 - \frac{1}{N}$ *is necessary for an* $N \times N$ *CIOQ switch operating under* ***any*** *algorithm which is not allowed to consider the number of scheduling phases in a time slot.*

The proof of Theorem 11 is a straight-forward extension of the 4×4 CIOQ switch example.

Appendix B: The Sufficiency of a Speedup of 2-1/N to Mimic a FIFO Output Queued Switch

We now show that it is possible to emulate a FIFO OQ switch using a speedup of $2 - \frac{1}{N}$. Specifically, we show that this emulation can be achieved by a CIOQ switch which follows the general framework described in Section 2, using a scheme that we call "Last In Highest Priority" (LIHP) to determine input priorities for incoming cells. As the name suggests, LIHP places a newly arriving cell right at the *front* of the input priority list. The analysis in this section borrows heavily from ideas described in Section 3.

In this section we use a slightly different time slot structure. A "normal" time slot has an arrival phase followed by two scheduling phases and then a departure phase, whereas a "truncated" time slot has an arrival phase, a scheduling phase, and then a departure phase. Since the speedup is $2 - \frac{1}{N}$, we assume that there are at least $N - 1$ normal phases between two truncated phases. The CIOQ switch does not need to know which phases are truncated.

At any time instant, and for any cell X, let $NTS(X)$ denote the number of truncated time slots between now and the time when this cell leaves the OQ switch, inclusive. Recall from Section 2 that $L(X) = OC(X) - IT(X)$ is the slackness of cell X, where $OC(X)$ and $IT(X)$ refer to the output cushion and input thread of the cell, respectively.

The following lemma holds for *CIOQ switches that use* LIHP and operate at a speedup of $2 - \frac{1}{N}$.

Lemma 3: *If the OQ switch being emulated is FIFO, then* $L(X) \geq NTS(X)$ *after the first scheduling phase and just before the arrival phase, for all cells* X *waiting on the input side.*

Theorem 12 is a consequence of Lemma 3 — we defer the proof of the lemma itself to the end of this section.

Theorem 12: *A speedup of* $2 - \frac{1}{N}$ *suffices for a CIOQ switch that uses LIHP to emulate a FIFO OQ switch.*

Proof: Suppose it is time for cell X to leave the OQ switch, and suppose that the CIOQ switch has successfully mimicked a FIFO OQ switch so far. Clearly, $OC(X)$ must be zero. If X has already crossed over to the output side then we are done. So suppose X is still queued at its input port. If the current time slot were truncated then $L(X)$ would be at least one (Lemma 3). But then the input thread would be negative, which is not possible. Therefore, the current time slot has two scheduling phases. Invoking Lemma 3 again, $L(X)$ must be at least zero after the first scheduling phase. Since $OC(X)$ is zero, the input thread of X must be zero too. Cell X, therefore, is at the front of both its input and its output priority lists, and will cross the switch in the second scheduling phase, just before the departure phase. This completes the proof of the theorem. z

Proof of Lemma 3: Suppose the lemma has been true till the beginning of time slot $t-1$. We prove that the lemma holds at the end of the first scheduling phase and at the end of the departure phase in time slot t.

We first consider the end of the first scheduling phase. Cells which were already present on the input side at the beginning of time t satisfy $L \geq NTS$, as NTS does not change (a property of FIFO -- the departure time of a cell from the OQ switch gets fixed upon arrival, and does not change), and L can only go up (see Lemma 1 for an explanation of why L can not decrease) during the arrival and the scheduling phases. Now consider a cell X which arrives during time slot t. Let $k = NTS(X)$. Since the slackness of a cell is at least zero upon arrival (remember that the input thread of an arriving cell is zero in LIHP), the slackness at the end of the first scheduling phase must be at least one. Therefore X trivially satisfies the lemma if $k \leq 1$. Suppose $k > 1$. At most N cells could have arrived during the current time slot, and therefore, there must have been a cell Y in the system with a NTS of $k-1$, and the same output port as X, at the *beginning* of time t (this is where we use the fact that the truncated time slots are spaced at least N apart). If Y is waiting on the input side, then $OC(Y) \geq L(Y) \geq k-1$. Since the OQ switch is FIFO, $OC(X) \geq OC(Y)$. But the input thread of the arriving cell X must be zero. Hence, the slackness of X is at least $k-1$ after the arrival phase, and consequently, at least k after the first scheduling phase. The case where Y is waiting at the output side is similar, and we omit the details.

Now concentrate on the end of time slot t. If this time slot turns out to be normal, then the slackness of any cell does not decrease during the second scheduling phase and the departure phase. Else, the slackness of any cell can go down by at most one. But the NTS value goes down by one for *all* cells in the system, and the lemma continues to hold.

Appendix C: Proof of Lemma 2.

The proof is by contradiction. Assume there does exist a cycle in the dependency graph on active cells. Pick a smallest cycle in this graph. If there is an edge from cell X to cell Y, then Y must be ahead of X either in the input queue ordering or in the output queue ordering. We call the edge an “input” edge in the former case and an “output” edge in the latter; ambiguities are resolved arbitrarily. The smallest cycle must have alternating input and output edges, because two successive input or output edges could be collapsed into one resulting in a smaller cycle. If there is an output edge from X to Y, then the output cushion of Y is at most as large as that of X. But X and Y are both active, and the input thread of an active cell must equal its output cushion. Therefore, the input thread of Y is no larger than the input thread of X. Also, if there is an input edge from X to Y then the input thread of Y must be strictly smaller than that of X; that is, X appears in Y’s input thread. The smallest cycle must have at least two edges, as there can be no self loops in the dependency graph. Consequently, the cycle must contain at least one input edge. But this implies that as we traverse the cycle once the input thread of the cell where we start must be larger than the input thread of the cell where we end. Since we start and end at the same cell as we traverse a cycle, this implies that the input thread of this cell must be less than itself. This is clearly impossible. Hence our assumption that there exists a cycle in the graph cannot be true, and the lemma is proved.