Matching Output QUeueing with a Combined Input and Output Queued Switch

Shang-Tse Chuang
Ashish Goel
Nick McKeown
Balaji Prabhakar[1]

Stanford CSL-TR-98-758

Abstract —

The Internet is facing two problems simultaneously: we need a faster switching/routing infrastructure, and we need to introduce guaranteed qualities of service (QoS). As a community, we have solutions to each: we can make the routers faster by using input-queued crossbars, instead of shared memory systems; and we can introduce QoS using WFQ-based packet scheduling. But we don't know how to do both at the same time. 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. We demonstrate 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 a "speedup" of $2 - 1/N$ *is both necessary and sufficient for this precise emulation. We introduce a variety of algorithms that configure the crossbar so that emulation is achieved with a speedup of two, and consider their running time and implementation complexity. We believe that, in the future, these results will make possible the support of QoS in very high bandwidth routers.*

1. Balaji Prabhakar is with MIT LIDS.

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.

Perhaps more importantly, 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,[2] 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. 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, 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

1. When we refer to output-queueing in this paper, we include designs that employ centralized shared memory.
2. By using schemes as proposed in [1], [2], for example.

readily achievable with commercially available memories. 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.

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% [10]. 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. While these results are academically interesting, they do not give us the principal benefit of output queueing: the ability to control the delay of individual packets. We believe that a well-designed network switch should perform predictably in the face of *all* types of arrival process, allowing the delay of individual packets to be controlled. Hence our approach is quite different, and our result subsumes previous work. 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. Further, we place no restrictions on arrivals; our results apply equally for any type of traffic, even if it saturates 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 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.

In this respect the formulation presented here is both novel and powerful: It allows us to obtain an algorithm that enables 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.

2 Background

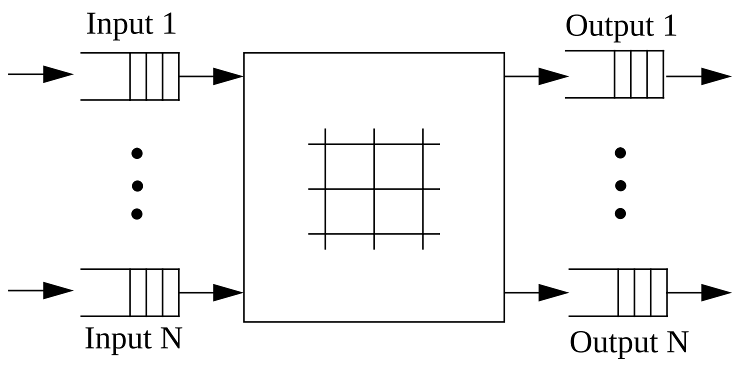
Consider the single stage, $N \times N$ switch shown in Figure 1. Throughout the paper we assume that packets begin to arrive at the switch from time $t = 1$, the switch having been empty before that time. Although packets arriving to the switch or router may have variable length, we will assume that 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. 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 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: General Combined Input and Output Queued (CIOQ) switch.



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 have the *identical behavior* of an OQ switch in the following sense:

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.

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 our CIOQ switch. As we will see later, the key to solving the speedup problem is a scheduling algorithm that keeps track of the cells in the CIOQ switch. The scheduling algorithms decides the order in which cells at the input are transferred across the switch fabric to the output in such a

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.

way that the cells may depart from the switch at the same time as they do in the shadow OQ switch. Each time cells are to be transferred, the scheduling algorithm selects a matching between inputs and outputs so 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 during each time slot.

2.1 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 arriving customers are added to an arbitray location in the queue based on some metric. For example, each customer may carry with them a departure time, and is placed in the queue ahead of all customers with a later departure time, yet behind customers with an earlier departure time. The only property that defines a push-in queue is that once placed in the queue, customers may not switch places with other customers. 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 customers are placed at an arbitrary location, and the customer at the head of the queue is always the next to depart. PIFO queues are quite general — for example, of a WFQ scheduling discipline operating at an output queued switch is a special case of a PIFO queue. (2) “Push-In Random-Out” (PIRO) queues, in which customers are removed from the queue in an arbitrary order. i.e. it is not necessarily the case that the next customer to depart is the one currently at the head of the queue. Later, we will use PIRO queues as a buffering mechanism at the input of a CIOQ switch.

We will assume that each input of the CIOQ switch maintains an input queue: an ordered set of cells waiting at the input port. In general, the CIOQ switches that we consider, can all be described using PIRO input queues.[1] Many orderings of the cells are possible — in fact, we will see that the exact nature of the ordering leads to a variety of interesting switch scheduling algorithms. The scheduling algorithms described in this paper differ only in the ordering of their input queues. So later, we will discuss different orderings in detail.

Similarly, each output maintains an output queue of the cells waiting to depart; we say that these cells form part of the output buffer. In addition, each output 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 always arranged in the order in which the cells would depart from the shadow OQ switch. This priority list will depend on the queueing policy followed by the OQ switch (FIFO, WFQ, strict priorities etc.).

2.2 Definitions

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

Definition 1: Time to Leave — *TL(***c***) is the time slot in which cell* **c** *would leave the shadow OQ switch. Of course, TL(***c***) is also the time slot in which it must leave from our CIOQ switch for the identical behavior to be achieved.*

Definition 2: Output Cushion — *OC(***c***) is the number of cells waiting in the output buffer at cell* **c***'s output port which have a lower time to leave value than cell* **c***.*

Notice that if a cell has a small (or zero) output cushion, then the scheduling algorithm must urgently deliver the cell to its output so that it may depart when its time to leave is reached. Conversely, if a cell has a large output cushion, the scheduling algorithm may temporarily set the cell aside while more urgent cells are delivered to their outputs. Note that because the switch is work-conserving, a cell's output cushion is decremented during

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

every time slot. A cell's output cushion can only be increased by newly arriving cells that are destined to the same output and have a more urgent time to leave.

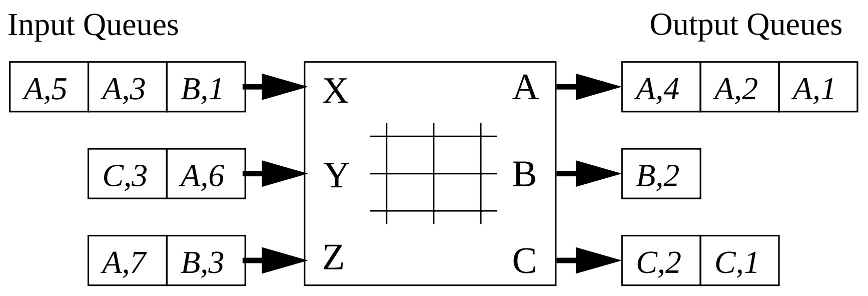
Definition 3: Input Thread — *IT(***c***) is the number of cells ahead of cell* **c** *in its input priority list.*

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 — *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 urgency 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.

Figure 2: A snapshot of a our CIOQ switch



To illustrate our definitions, Figure 2 shows a snapshot of our CIOQ switch with a number of cells waiting at its inputs and outputs. For notational convenience, we define the time of the snapshot to be time slot 1. We use the notation (P, t) to represent a cell that, in the shadow switch, will depart from output port P at time t, its time to leave. 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 $L(\mathbf{c}) = OC(\mathbf{c}) - IT(\mathbf{c}) = 1$.

2.3 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.

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.

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. **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 [12]— 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: *Input Queue Ordering is PIRO: 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 customers are in the queue, i.e. customers in the queue cannot switch places, and

3. Customers may be selected to depart from the queue from any location. of the cells in the the relative ordering of cells waiting in the queue does not change over time.

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.

At 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 our 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.

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

Having defined speedup, we now address the next natural question: what is the minimum possible speedup, S, of a CIOQ switch that emulates an OQ switch. We answer this question with the following theorem.

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

Proof: The proof is by counterexample — see 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 2: *(Sufficiency). A $N \times N$ CIOQ switch with a speedup of $2 - \frac{1}{N}$ can exactly emulate a $N \times N$ FIFO OQ switch.*

Proof: The proof is based on the insertion policy Last In Highest Priority (LIHP) as described in Appendix B. ∎

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

Our proof of Theorem 2 uses a simple input queue insertion policy (LIHP), but unfortunately the proof is complex and, in our opinion, couterintuitive. Further, LIHP is quite inefficient. In an attempt to provide a more intuitive understanding of the speedup problem, we present a simple and more efficient insertion policy that mimics an OQ switch with a FIFO queueing discipline with a speedup of two. 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. Since this decision is crucial, we restate CCF more formally.

CCF:. Suppose cell C arrives at input port P. Let X be the output cushion of C. Insert cell C into the $(X+1)$ 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. Note that upon arrival, C's slackness is non-negative.

The intuition behind this insertion policy is that a cell with a small output cushion is approaching its time to leave (i.e. it becomes "more critical"), and 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 need not be so close to the head of its input queue. Informally, our proof will proceed as follows. We first show an important 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 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, we can conclude that the slackness of a cell 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 3: *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 a 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 can depart on time. ▮

5 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 queueing policies called PIFO (Push-In First-Out); a class that includes the widely-used queueing policies such as WFQ and Strict Priority queueing.

Thus an OQ switch that follows a PIFO queueing policy can insert a cell anywhere in its output queue but it can not change the relative ordering of cells that are already waiting in the queue. 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.

We can use CCF to mimic not just a FIFO OQ switch but any OQ switch that follows a PIFO queueing policy. 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 4: *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 4 is almost identical to that of Theorem 3, and is omitted. ∎

5.1 Network Stability

Theorem 4 also has interesting implications for network stability, as defined in [13]. It is known that several commonly used queueing disciplines, FIFO being the most notable example, can result in global instability in the network [13]. Further, such instability can occur at relatively low congestion and in very simple networks [15]. It is also known that LIS (Longest In System), a queueing policy that gives preference to cells that have been in the network the longest, results in global stability even at high congestion [13]. Since LIS is also a PIFO queueing policy, we can use CCF to ensure global stability in a network comprised of CIOQ switches operating at a speedup of 2.

6 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] Further, the stable matching algorithm must consider all of the cells present in the input queue. We remove both disadvantages in Section 6, where we show how to do stable matchings in N iterations, and how an algorithm can use VOQs to consider many fewer cells in the input queues.

The Just In Time (JIT) strategy reduces the number of iterations needed to compute a stable matching to N (from N^2). The Group By Virtual Output Queue (GBVOQ) algorithm ensures that the number of input cells considered by the stable matching algorithm is equal to the number of active virtual output queues rather than the total number of cells. These two schemes, when combined, are designed to allow an implementation of a CIOQ switch that mimics an OQ switch with PIFO output scheduling.

6.1 The Just In Time (JIT) strategy:

Before we describe the JIT strategy, we explain the efficiency bottleneck that JIT is 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 A to B 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 above, there may be cycles in this dependency graph. These cycles are the main cause of inefficiency in finding stable matchings, and the Just in Time strategy is aimed at getting rid of these cycles.

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.

The Just In Time strategy is simple: *During each scheduling phase, mark as active all cells with a slackness of zero, and mark all other cells inactive*. The stable matching algorithm considers only active cells. Since the slackness of a cell can never become negative[1], CCF combined with JIT strategy can emulate any OQ switch that follows a PIFO queueing policy. The dependency graph is now limited to only the active cells.

Lemma 2: *If JIT is used in conjunction with CCF, the resulting dependency graph G is acyclic.*

We omit the proof of Lemma 2, and instead focus on its implications. Since there are no cycles, there has to be at least one sink (i.e. a vertex with no outgoing edges) in G. Let X be the cell corresponding to 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. Having matched cell X, we remove from the graph all cells which have the same input or output port as X. 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. An algorithm to find a stable matching in $O(N)$ parallel iterations was given in [14]; however their algorithm is quite complicated and not efficient for realistic values of N.

We now address the second disadvantage of CCF, i.e. many cells must be considered by the stable matching algorithm.

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

With CCF, the stable matching algorithm may need to consider as many cells as are contained in the input queues. However, we can simply group incoming cells into Virtual Output Queues to obtain an upper bound of N on the number of cells that need to be con-

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.

sidered at any input port. The algorithm, GBVOQ, which achieves this bound is described below.

We explain here how GBVOQ can be used to emulate a FIFO OQ switch. This technique can, in general, be extended to a system with PIFO departure order. GBVOQ maintains a VOQ for each input-output port pair. When a new cell arrives at an input port, GBVOQ checks to see if the corresponding VOQ is empty. If it is, then the incoming cell is also placed at the head of the input queue. If, on the other hand, the VOQ corresponding to the new arrival is non-empty, the new cell is placed at the tail of its VOQ: i.e. it is inserted in the input priority list just behind the last cell which belongs to the same VOQ. It is easy to see that all cells that are in the same VOQ occupy contiguous positions in the input queue. Therefore it is sufficient to just keep track of the relative priority ordering of VOQs. Since there are at most N VOQs in a FIFO switch, we get the requisite bound on the size of the input priority list. Since GBVOQ does not assign a 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 also has several other desirable properties. First, the decision of where an incoming cell needs to be inserted is much simpler for GBVOQ than CCF. Like CCF, GBVOQ too can be used in conjunction with the JIT strategy to reduce the number of iterations needed to compute a stable matching. In fact, JIT 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.

7 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, D. Wilson, “On the self-similar nature of Ethernet traffic”, *Proc. of Sigcomm,* San Francisco, pp.183-193. Sept 1993.

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

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

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

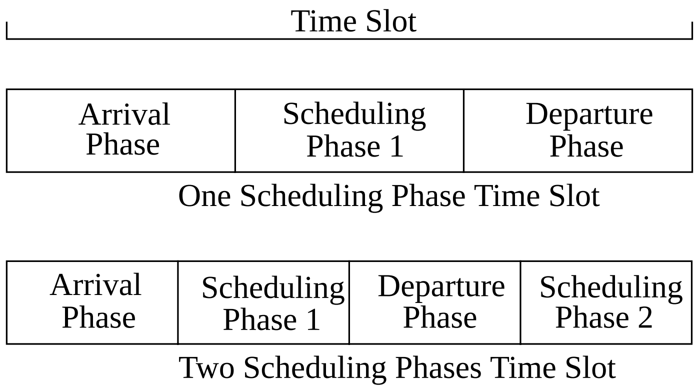
[15] Goel,A. “Stability of Networks and Protocols in the Adversarial Queueing Model for Packet Routing.” Stanford University Technical Note STAN-CS-97-59.

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 2.1 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 a $N \times N$ 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 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 4: Lower Bound Input Traffic Pattern for a 4x4 switch

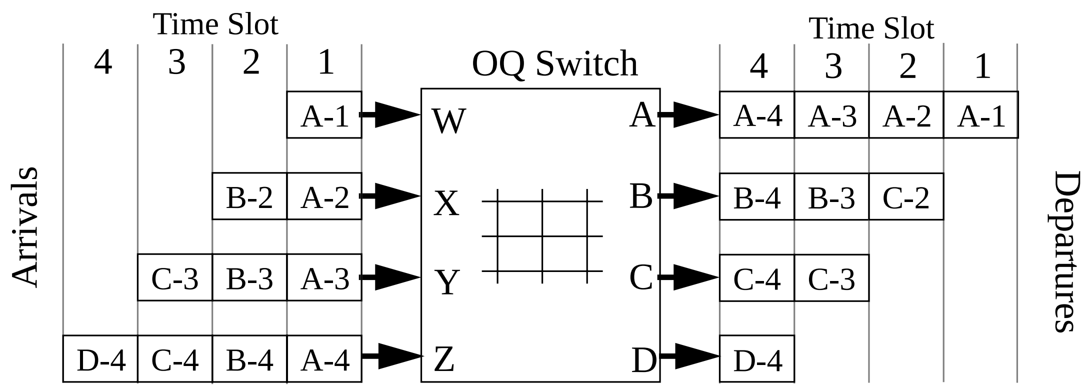


Figure 5 shows the only possible schedule for transferring these cells across in seven phases. Of the four time slots, the last one is truncated, giving a total of seven phases. Cell A-1 must leave the input side during the first phase, since the CIOQ switch does not know whether the first time slot 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 $2 - \frac{1}{4}$).

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

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		B-2	**A-3**	
4			B-3	
5			C-3	**B-4**
6				C-4
7				D-4

(c)

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

(d)

Theorem 5: *A minimum speedup of* $2 - \frac{1}{N}$ *is necessary for a* $N \times N$ *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 5 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 4.

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.

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 of a CIOQ switch that uses LIHP and a speedup of $2 - \frac{1}{N}$, .*

The following theorem is a consequence of Lemma 3 — we defer the proof of the lemma itself to the end of this section.

Theorem 6: *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. ∎

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.