Maintaining Packet Order in Two-Stage Switches

Isaac Keslassy, Nick McKeown
Computer Systems Laboratory, Stanford University
Stanford, CA 94305-9030
{keslassy, nickm}@stanford.edu

***Abstract* -- High performance packet switches frequently use a centralized scheduler (also known as an arbiter) to determine the configuration of a non-blocking crossbar. The scheduler often limits the scalability of the system because of the frequency and complexity of its decisions. A recent paper by C.-S. Chang et al. introduces an interesting two-stage switch, in which each stage uses a trivial deterministic sequence of configurations. The switch is simple to implement at high speed and has been proved to provide 100% throughput for a broad class of traffic. Furthermore, there is a bound between the average delay of the two-stage switch and that of an ideal output-queued switch. However, in its simplest form, the switch mis-sequences packets by an arbitrary amount. In this paper, building on the two-stage switch, we present an algorithm called Full Frames First (FFF), that prevents mis-sequencing while maintaining the performance benefits (in terms of throughput and delay) of the basic two-stage switch. FFF comes at some additional cost, which we evaluate in this paper.**

I. Introduction

Most high performance packet switches today use input queueing, and a non-blocking (usually crossbar) switch fabric [1][2]. To overcome head-of-line blocking and enable high throughput, the input buffers are arranged as virtual output queues (VOQs) [3]. To simplify the tasks of memory management and scheduling, a fixed sized time slot is used, and hence arriving variable length packets are segmented into fixed size packets, or "cells". Each time slot a centralized scheduler examines the contents of the VOQs to determine the configuration of the switch fabric for the next time slot. Numerous papers have studied this approach, and have proposed new scheduling algorithms that are simple to implement [4][5], provide throughput guarantees [6][7][8] or provide delay guarantees [9][10].

In 1993, Anderson et al. [4] observed that the job of the scheduler is equivalent to finding a matching in a bipartite graph. McKeown et al. [6] showed that 100% throughput could be guaranteed if a maximum weight matching is found, which has a complexity of $O(N^{2.5}logN)$, where N is the number of switch ports [11]. This has proved too complex for use in existing high performance packet switches. With the number of ports increasing (hence increasing the complexity) and line rates increasing (hence reducing the time in which the algorithm must complete), maximum weight matching algorithms will continue to be impractical.

Some maxim*al* size matching algorithms and heuristics have been proposed that have a complexity of $O(NlogN)$ or lower [4][5][12]. While these algorithms have been widely used, the need for switches with more ports and faster line rates makes these algorithms harder and harder to implement. In fact, it appears that the scalability of most input-queued switches today is limited by the scheduling algorithm.

At least four different approaches have been proposed in the literature to improve scalability. The first approach is to use a simple randomized scheduling algorithm that exploits the correlation between successive matchings [13][14]. Tassiulas et al. [13] showed that a simple $O(logN)$ randomized scheduling algorithm could guarantee 100% throughput for Bernoulli i.i.d. arrivals, although packet delay is large. Shah et al. [14] recently introduced an alternative $O(1)$ algorithm that leads to lower delays. The second approach is to increase the length of a cell, which in turn increases the time slot and gives the scheduler more time to complete [15]. The third approach attempts to pipeline the scheduler, allowing it to use out-of-date information [7]. Although this approach does not reduce the throughput, it increases packet delay.

The fourth approach, that motivated this paper, adopts a novel structure proposed by C.-S. Chang et al. [16]. Their switch consists of two stages, but has no scheduler. Both stages of the switch follow a deterministic sequence of N different configurations. All that is required is that each input is connected to each output exactly once in the sequence. For example, the sequence that we will assume throughout this paper is one in which input i is connected to output $((n+i)$ modulo $N)$ at time n in the first stage, and input j is connected to output $((n-j)$ modulo $N)$ at time n in the second stage. A cell arriving to the first stage is immediately transferred without buffering to an input of the second stage switch. The cell is placed in a VOQ according to its output. The VOQs in the second stage are all serviced at the same rate by a second deterministic sequence.

The intuition behind the two-stage approach is as follows. It is known that a single-stage crossbar switch with VOQs that are served by such a deterministic sequence will provide 100% throughput for uniform[1] Bernoulli i.i.d. traffic; but no guarantees are possible when the traffic is non-uniform. In the two-stage switch, the first stage effectively makes non-uniform traffic uniform by spreading it evenly over the second stage. Hence the two stages might be expected to provide 100% throughput. In [16] this is proved rigorously, for a particular definition of throughput and for a broad class of arrival processes.

A disadvantage of the two-stage switch is that cells can be mis-sequenced by an arbitrary amount. Although strictly not

This work was supported by the Wakerly Stanford Graduate Fellowship, by the Powell Foundation, by the Stanford Networking Research Center, by MARCO contract# 2001-CT-888 and by DARPA contract# MDA972-02--1-0004.

[1] Throughout this paper, "uniform" refers to traffic in which the destination of each cell is chosen uniformly and at random from among the set of outputs.

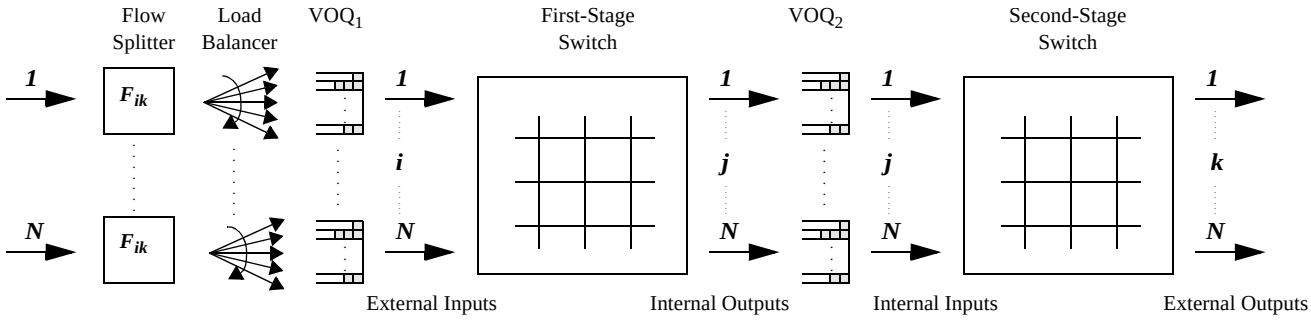


Figure 1: Switch Architecture

disallowed in an Internet router [17], mis-sequencing can cause problems for current versions of TCP [18][19], and so common rules of practice dictate that routers should not mis-sequence packets.

In a second paper, Chang et al. [20] propose two different solutions that bound the amount of mis-sequencing, enabling the addition of a finite resequencing buffer after the second stage. This is similar to the parallel packet switch (PPS) [21]. Nevertheless, the first scheme proposed in [20] either requires up to N memory accesses per time slot in any second-stage input (for packets that arrived to the switch at the same time), or needs to use a complex buffering mechanism. The second scheme, EDF, needs to retrieve the packet with the smallest timestamp from a queue, making it hard (but not impossible) to implement in a high performance switch.

In this work, our goal is to design a two-stage switch with the same throughput advantages. Instead of bounding the amount of mis-sequencing, our approach prevents mis-sequencing from taking place, eliminating the need for a resequencing buffer.

In the remainder of this paper, we present an algorithm, called "full frames first" (FFF) that leads to an average packet delay within a constant from the ideal output queuing (OQ), and therefore reaches the same throughput as OQ. It uses three-dimensional queues (3DQs) (which are an extension of VOQs) to avoid packet mis-sequencing. FFF comes at a cost: the 3DQ queueing structure is more complex than simple VOQs; and although simple, FFF is not as trivial as the deterministic sequence of configurations.

This paper is organized as follows. We first describe Chang's switch architecture and the EDF algorithm. Then we introduce 3DQ and show how it helps prevent mis-sequencing by giving some choice to the external outputs. Finally, we present the FFF algorithm, showing that it has no mis-sequencing and proving some theorems on its delay and throughput.

II. Switch Architecture

A. Definitions

Throughout this paper, we'll use the terms "packets" and "cells" interchangeably to designate fixed-size cells. We'll denote the number of switch ports by N, and assume $N \geq 2$.

The switch architecture that we will use as the basis for this paper is taken from [20], and shown in Figure 1. Although it is more complex than the basic structure in [16], the additional queues in the first stage help to limit the amount of mis-sequencing. The switch architecture consists of two stages of switching. The inputs of the first stage are called external inputs (EIs), and numbered $i = 1, ..., N$. The outputs of the first stage, called internal outputs (IOs), are collocated with the inputs of the second stage, called internal inputs (IIs). IOs and IIs will be used interchangeably in this paper, and are numbered $j = 1, ..., N$. Finally, the outputs of the second stage, called external outputs (EOs), are numbered $k = 1, ..., N$.

Let's follow the path of packets through the switch.

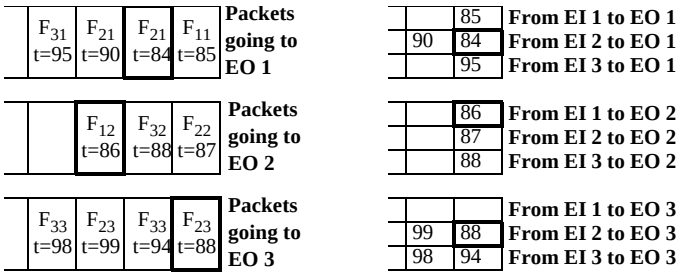
1. First, a flow splitter labels each packet in EI i as belonging to a given flow F_{ik}, where k is the EO to which this packet is destined. There are therefore N possible flows per EI representing the N different EOs to which the packets may be destined.
2. A load balancer sends all the packets from F_{ik} to the N VOQ_1s (corresponding to the N IOs), in a round-robin manner - i.e. the first packet from a given flow is sent to the VOQ_1 for IO 1, the second one is sent to the VOQ_1 for IO 2, and so on, independently of the packet arrival times. Because the load balancers are not necessarily synchronized with the sequence of configurations of the first-stage switch, arriving packets are buffered and do not necessarily immediately leave the VOQ_1s. Note that the inputs of the VOQ_1s are the EIs, their outputs are the IOs (collocated with the IIs), and there is a different load balancer for each flow.
3. The VOQ_1s are served in deterministic order by the first-stage switch, and when their turn comes the packets leave their VOQ_1 and pass through the first-stage switch.
4. After leaving the first-stage switch the packets are queued in the VOQ_2s. The inputs of the VOQ_2s are the IIs, and their outputs are the EOs.
5. The VOQ_2s are served in deterministic order by the second-stage switch, and when their turn comes the packets leave their VOQ_2 and pass through the second-stage switch.
6. Finally, the packets leave the second-stage switch and exit through the EO.

The following property of the switch will prove useful in this paper (proved in [20]).

Property 1 If a packet arrives to the switch at time t, it will arrive to the VOQ_2s no sooner than t, and no later than $t + N^2$.

B. EDF: Example of Algorithm Using This Switch Architecture

Suppose that two packets belonging to flow F_{ik} arrive back-to-back at EI i. Because they may be placed in different VOQ_2s at the second stage, they may both experience very different delays through the switch, and may become mis-sequenced by an arbitrary amount.



(a) (b)

Figure 2: (a)VOQ_2 vs. (b)3DQ structure of a given II j

The earliest deadline first (EDF) algorithm prevents mis-sequencing by serving cells in the VOQ_2s in the order that they arrived to the switch, rather than strictly from the head of line. EDF has the following properties, proved in [20].

Property 2 Packet mis-sequencing is bounded by $2N^2 + N$.

Note that it is therefore possible to add a finite resequencing buffer after the switch for each external output.

Property 3 The packet delay in EDF is bounded by the sum of the packet delay in a first-come-first-served (FCFS) OQ switch, and a constant equal to $2N^2 + N$.

This implies that the EDF algorithm has good delay and throughput properties, since it follows an FCFS OQ switch closely.

However, EDF requires up to Q_{max} timestamps to be compared at every time-slot in order to determine which cells to service, where Q_{max} is the maximum length of a VOQ_2. This makes the EDF algorithm difficult to implement in practice. In what follows, we will first show how to simplify the EDF algorithm, and then eliminate the need for a resequencing buffer at the external outputs.

III. 3DQ, An Extension of VOQ

A. The Return of HOL Blocking

Consider a packet, p, that sits in the VOQ_2 (j, k). We'll assume that p was the earliest arriving packet to the switch among all packets in its VOQ_2 (j, k), but that p is not currently sitting at head-of-line (HOL) in its VOQ_2. Packet p is obviously the earliest arriving packet of its flow F_{ik} in VOQ_2 (j, k), and therefore sits in front of the other packets of its flow in VOQ_2 (j, k). However, it is blocked by packets ahead of it that arrived later to different external inputs and are also scheduled to depart from EO k. This is classical HOL blocking, and the solution is to subdivide each VOQ_2 into a separate queue for each external input.

B. Three-Dimensional Queueing

VOQ_2s transform one-dimensional queues into two-dimensional queues, one per (input, output) pair. There are therefore N^2 VOQ_2s. In this switch, we will use three-dimensional queues (3DQs), with a different queue per (i, j, k); hence, there are now a total of N^3 3DQs. From hereon, we'll assume that we replace the VOQ_2s by 3DQs.

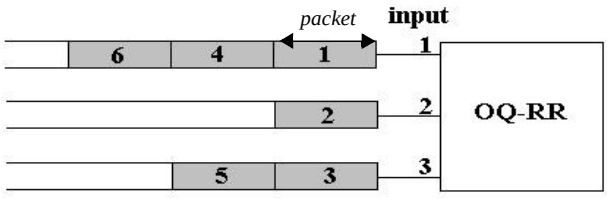


Figure 3: OQ-RR example

C. An Application of 3DQ: EDF-3DQ

With 3DQs, the earliest cell for (j, k) is always the HOL cell in its queue. Therefore, if we want to use the EDF algorithm with a 3DQ structure (we'll call it the EDF-3DQ algorithm), we only need a comparison among N timestamps, instead of a comparison among Q_{max} timestamps. This simplification comes at the cost of using N^3 3DQs instead of N^2 VOQ_2s.

Figure 2 compares a VOQ_2 structure with a 3DQ structure for a given II. The numbers on the packets represent their flow F_{ik} and their arrival time to the switch, and the packets with a bold border are the earliest ones in their VOQ_2. The figure illustrates how HOL blocking with VOQ_2s is solved using a 3DQ structure. For instance, in the VOQ_2 $(j, 1)$, packet *84* from F_{21} is blocked by the HOL packet *85* from F_{11}, arrived later ($85 > 84$). However, in the 3DQ structure, packet *84* is the HOL of its 3DQ and is not blocked anymore.

Our next step is to eliminate the resequencing buffer by preventing mis-sequencing from occurring in the first place.

IV. Full Frames First

A. Background

FFF (Full Frames First) is an algorithm that maintains packet order.

To understand how FFF works it helps to understand how the round-robin version of OQ works (called OQ-RR). Consider the illustration of OQ-RR for one output in Figure 3, where all packets are assumed to have the same output destination. The numbers on the packets correspond to the order in which they will be serviced, assuming no future arrivals. Therefore, OQ-RR will service packet 5 before packet 6, even if packet 5 arrived later. Note that because the average delay is independent of the order in which the packets are serviced, OQ-RR will have the same average delay as OQ-FCFS (the FCFS version of OQ). Also, note that OQ-RR is work-conserving (i.e., if there is at least one packet in the queue, then OQ-RR is not idle).

Now, assume that the algorithm doesn't deal with *packets*, but with *frames*, where one frame consists of N packets. The new algorithm, called Frames-RR, first services all full frames in round-robin order (where a frame is considered to be full if its N^{th} slot contains a packet). When there are no full frames requiring service, it services non-full frames in round-robin order. For instance, in Figure 4, the frames are serviced in the order indicated. First, the frames 1 through 5 are serviced because they are full, including frame 3 which is considered full because its last slot is occupied by a packet. Afterwards, the non-full frames 6 and 7 are serviced in a round-robin order.

Frames-RR is clearly not work-conserving for *packets*. How-

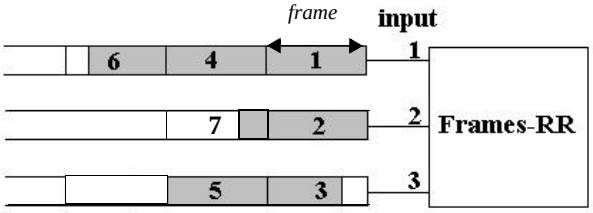


Figure 4: Frames-RR example

ever, it is work-conserving for *full frames*, in the sense that if there is at least one full frame left, then there is at least one full frame being serviced.

B. FFF: A Combination of Frames-RR and 3DQs

FFF applies Frames-RR to the 3DQs in the second-stage of the switch. To understand its operation, consider external output k. We'll define a *cycle* to be the set of N consecutive time slots during which EO k receives cells successively from IIs *1* through N, and we'll define the *candidate set* of 3DQs for (i, k) as $\{(i, 1, k), (i, 2, k), ..., (i, N, k)\}$.

Assume that the last serviced cell in the candidate set came from II j_{last}. Then, because of the properties of the load-balancer, we know that the next in-order cell for the flow F_{ik} will come from II $j_{last} + 1$ (modulo N). Let p_{ik} be the pointer to the II of the next in-order cell: $p_{ik} \leftarrow j_{last} + 1 (mod\ N)$.

For instance, if the last cell was read from II *2*, then the next in-order cell will necessarily be read from the II numbered: $p_{ik} = 2 + 1 = 3$. Further, we know that if cell C is in front of cell C' in any 3DQ, then C necessarily arrived to the switch earlier than C'. Therefore, if there is any cell from flow F_{ik} that is head-of-line of its 3DQ in II *3*, then this cell must be the next in-order one.

We define the *frame* for (i, k) as $f(i, k) = \{(i, p_{ik}, k), (i, p_{ik} + 1, k), ..., (i, N, k)\}$, and we will say that frame $f(i, k)$ is *full* if every 3DQ (i, j, k) for $j = \{p_{ik}, \ldots, N\}$ is non-empty. We can see that if the frame is full then its next in-order cell is in 3DQ (i, p_{ik}, k), the one after is in $(i, p_{ik} + 1, k)$, and so on, up until (i, N, k). In other words, a frame $f(i, k)$ is said to be full if, and only if, it is possible to transfer in-order cells from (i, p_{ik}, k) up until (i, N, k). This is the key to preventing the cells within a frame from becoming mis-sequenced.

In FFF, an external output reads all the cells in a full frame from one external input, before moving on to read a full frame from the next external input. External output k uses the round-robin pointer $p_{ff}(k)$ to remember which EI the last full frame came from. In this manner, each external output gives an opportunity to each external input in turn to send a full frame to it. When there are no more full frames, EO k serves the non-full frames in round-robin order, using the pointer $p_{nff}(k)$.

More precisely, the following three computations are performed by external output k at the beginning of every cycle:

1. Determine which of the frames $f(i, k)$ is full, where $i \in \{1, \ldots, N\}$.
2. Starting at $p_{ff}(k)$, find the first full frame. If the first full frame arrived from EI i_{ff}, then $p_{ff}(k) \leftarrow i_{ff} + 1$, modulo N. If there is no full frame, $p_{ff}(k)$ doesn't change.
3. If there is no full frame, starting at $p_{nff}(k)$, find the first non-full frame. Update $p_{nff}(k) \leftarrow p_{nff}(k) + 1$, modulo N.

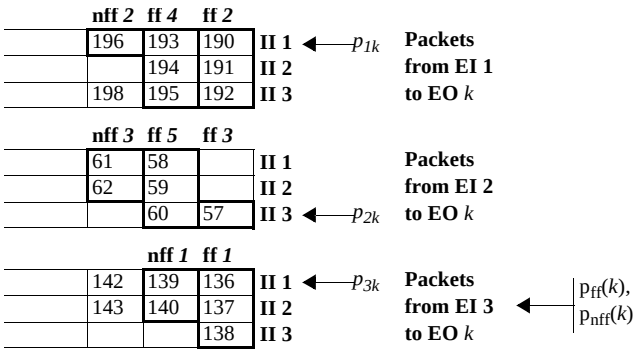


Figure 5: Illustration of the FFF algorithm for EO *k*. Note that the 3DQs have been rearranged so that all of the queues containing cells from a given external input are adjacent to each other.

C. Illustration of the FFF Algorithm

Assume that at the beginning of a cycle for a given EO k, the 3DQs are in the states shown in Figure 5. In the figure, the 3DQs have been rearranged so that all of the queues containing cells from a given external input are adjacent to each other. In practice, of course, the queues are not arranged like this, but they have been redrawn to help explain the algorithm. The number in each packet represents its sequence number within its (i, k) flow. The numbers above the frames (in bold) indicate the order in which they will be served. Assume that there are no further arrivals.

- Initially in the example, $p_{ff}(k) = p_{nff}(k) = 3$, and frame pointers are $p_{1k} = 1$, $p_{2k} = 3$ and $p_{3k} = 1$.
- At the first time-slot, FFF serves the first full frame that arrived from external input $p_{ff}(k) = 3$. The first full frame is $ff1 = \{136, 137, 138\}$, and so FFF serves it over three consecutive cell times, delivering the three cells in order to EO *k*. Pointers are updated: $p_{3k} \leftarrow 1$, $p_{ff}(k) \leftarrow 1$.
- FFF then serves the three cells from external input 1 in frame $ff2 = \{190, 191, 192\}$, then updates $p_{1k} \leftarrow 1$, $p_{ff}(k) \leftarrow 2$. According to our definition, $ff3 = \{57\}$ is a full frame from external input 2, even though it only contains one packet. FFF serves it and updates the pointers. Since there is no full frame from external input 3, the next served full frame is $ff4$, and then $ff5$. The pointers are now: $p_{ff}(k) = p_{nff}(k) = 3$, and $p_{1k} = p_{2k} = p_{3k} = 1$.
- There are no full frames left. FFF serves the non-full frames in round-robin order: $nff1$, $nff2$ and $nff3$. Pointers are updated to $p_{ff}(k) = p_{nff}(k) = 3$, $p_{1k} = 2$, and $p_{2k} = p_{3k} = 3$. Note that the cell numbered *198* is not

serviced, because there is no ordered cell in its frame at II $p_{1k} = 2$ (the expected cell numbered *197* is still queued in its VOQ_1). Similarly, *142* and *143* will not be serviced as long as there is no cell at II $p_{3k} = 3$.

D. Pros and Cons of FFF

The main advantage of FFF is that packets are not mis-sequenced, and so we can eliminate the resequencing buffer at the external outputs.

It is also interesting to compare FFF with *i*SLIP [5], which is a widely used practical heuristic for single-stage crossbar switches:

1. FFF has 100% throughput whenever OQ has 100% throughput (proved in Section V.C). *i*SLIP can only guarantee 100% throughput for Bernoulli i.i.d. uniform traffic.
2. The average packet delay for FFF is bounded by the sum of the average packet delay for OQ (with the same traffic) and a constant delay (proved in Section V.B). No delay bounds exist for *i*SLIP.
3. FFF appears straightforward to implement. For each time-slot t there is only one EO $k(t)$ that begins its cycle. During this time-slot $k(t)$ must first determine which, if any, frames are full. It determines independently and in parallel for each EI i whether or not the frame $f(i, k(t))$ is full, i.e. whether or not there is a cell belonging to every 3DQ $(i, j, k(t))$ for $j = \{p_{ik(t)}, \ldots, N\}$. In other words, $k(t)$ uses N bits for each EI i, masks them using $p_{ik(t)}$, and logically ANDs them together. Finally, $k(t)$ uses an N-bit programmable priority encoder to identify the first full frame. This is the same complexity as just one iteration of *i*Slip. It is also possible to use an $O(1)$ version of FFF by exploiting slightly out-of-date information, with similar delay and throughput properties. For brevity, this property is not developed in this paper.
4. Because of the predetermined, and non-conflicting schedule used by both stages of switching, FFF does not need a centralized scheduler. It is sufficient for each external output to schedule the frames (and hence cells) that it will receive. This is not practical in iterative algorithms such as *i*SLIP which need to be centralized because of the large amount of communications between inputs and outputs.
5. FFF does not require much information to be sent between each internal input and each external output. First, let's consider the communication from an internal input to the scheduler at an external output. Each II receives at most one new packet per time-slot, and it is known in advance from which EI it comes, because of the predetermined sequence of configurations. The II can tell the EO the packet's destination (and that a packet arrived) using $log_2(N) + 1$ bits. Now let's consider the communications from the external output to an II. Every time-slot t, $k(t)$ tells each internal input which frame (if any) it will be reading in this cycle, requiring $log_2(N) + 1$ bits.
6. FFF seems simple enough to be implemented in hardware.
7. FFF seems well suited to optical switch fabrics based on technologies such as MEMS [25][26], VCSELs [27], tunable lasers [28], electro-holography [29], etc. This is for two reasons. First, FFF allows the switch fabric to rotate through a simple deterministic sequence of configurations, that are known in advance. It seems reasonable to expect that for most optical technologies, a fixed rotational pattern of configurations is easier to implement and can be reconfigured faster than if the pattern was unpredictable. For example, with MEMs mirrors one could imagine a mirror with N facets that rotates by a fixed amount each time-slot. Second, since both stages are configured according to a fixed sequence, it may be possible to replace them with a single switch that is configured once per time-slot, with two cells transferred per configuration. In the first half of a time-slot, the switch transfers cells for the first stage (from EIs to IOs), and in the second half, it transfers cells for the second stage (from IIs to EOs).

However, FFF has some drawbacks.

1. FFF uses two switching stages instead of one. On the face of it, this is similar to using a crossbar switch with a speedup of two. In this case, there is a spatial speedup rather than a speedup in time. However, notice that the two components that normally limit the speed of the system — the bandwidth of the memories at each stage, and the scheduler — run at the same speed as the external line.
2. FFF needs $2N$ buffers (first- and second- stage) instead of N buffers. While it is possible to combine the buffers into N shared buffers (if EIs and IIs share the same linecard), this would double the memory bandwidth.
3. FFF uses N^3 3DQs in the internal buffer, while single stage switches usually use only N^2 VOQs (thus requiring more pointers, and a more complicated buffer management algorithm).
4. FFF requires a load balancer at the first stage.

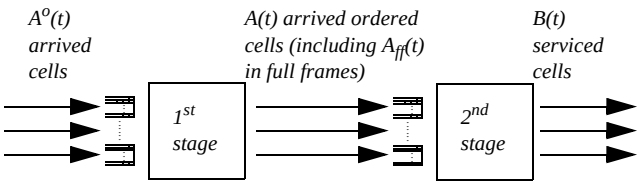
V. FFF Performance

In this section we show that the average delay for the FFF algorithm is less than the average delay for OQ plus a constant, and that FFF has the same throughput as OQ. The proofs rely on the observation that FFF is work-conserving for full frames.

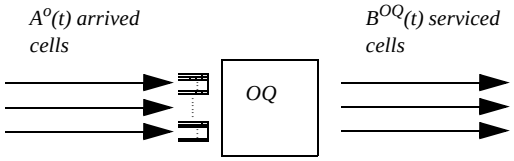
A. Definitions

For simplicity, we will only consider the cells destined to a given EO k. We'll define the following values, as illustrated in Figure 6.

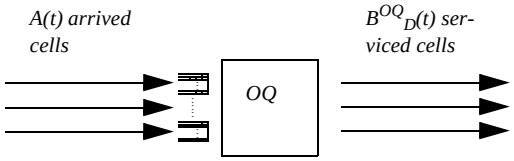
1. $A_i^o(t)$ is the cumulative number of cells destined to EO k that have arrived to EI i up to and including time-slot t. It is therefore the index of the last cell from F_{ik} that has arrived to EI i.
2. $A^o(t) = \sum_{i=1}^{N} A_i^o(t)$ is the total number of cells destined to k that have arrived to the switch up until t.
3. $A_i(t)$ is the index of the last cell in flow F_{ik} to have arrived to an II by the beginning of the current cycle, without any cells missing from the flow's FIFO ordering. We call the cells that are in their FIFO order, without any cells missing in front of them, the *ordered cells*. The total number of ordered cells



(a) Two-stage switch with FFF



(b) OQ model for first stage (with $A^o(t)$ arrivals)



(c) OQ model for second stage (with $A(t)$ arrivals)

Figure 6: Illustration of terminology

is $A(t) = \sum_{i=1}^{N} A_i(t)$. For instance, in Figure 5, $A(t) = 196 + 62 + 140$.

By definition, if t' is the beginning of a cycle for EO k (that is, $t' = k + 1 + l \cdot N$, with l an integer), then $A_i(t) = A_i(t')$ when $t' \le t \le t' + N - 1$. In addition, as shown in Property 1, $A_i^o(t' - N^2) \le A_i(t') \le A_i^o(t')$ (assume that these values are zero for $t' \le 0$).

4. $B_i(t)$ is the number of ordered cells that have already been served by the second stage up until time-slot t, and $B(t) = \sum_{i=1}^{N} B_i(t)$. In Figure 5, $B(t) = 189 + 56 + 135$.
5. $q(t) = A(t) - B(t)$ is the number of ordered cells queued in the 3DQs that are destined to k (e.g., $q(t) = 7 + 6 + 5$).
6. $A_{ff}(t) = \sum_{i=1}^{N} N \cdot \left\lfloor \frac{A_i(t)}{N} \right\rfloor$ is the number of cells in full frames already arrived to the 3DQs and destined to k.
7. Likewise, $B_{ff}(t) = \sum_{i=1}^{N} N \cdot \left\lfloor \frac{B_i(t)}{N} \right\rfloor$ is the number of cells in full frames that have already been served by the second stage.
8. $ff(t) = \frac{A_{ff}(t)}{N} - \frac{B_{ff}(t)}{N} = \sum_{i=1}^{N} \left\lfloor \frac{A_i(t)}{N} \right\rfloor - \left\lfloor \frac{B_i(t)}{N} \right\rfloor$ is the number of full frames queued in the 3DQs, and $q_{ff}(t) = N \cdot ff(t) = A_{ff}(t) - B_{ff}(t)$. For example, in Figure 5, $A_{ff}(t) = 195 + 60 + 138$, $B_{ff}(t) = 189 + 54 + 135$, $ff(t) = 2 + 2 + 1$ and $q_{ff}(t) = 6 + 6 + 3$.
9. $B^{OQ}(t)$ and $q^{OQ}(t)$ are respectively the cumulative number of services and the length of the queue in an FCFS OQ switch of speed-up 1, where the cumulative number of arrivals is $A^o(t)$. In other words, this represents an OQ switch with arrivals as seen by the external inputs.
10. Similarly, $B_D^{OQ}(t)$ and $q_D^{OQ}(t)$ correspond to what we will call *the delayed version of OQ*, which is computed with a cumulative number of arrivals equal to $A(t)$. In other words, it represents an OQ switch of speed-up 1 with the same ordered arrivals as seen by the IIs in the two-stage switch. Note that some well-known properties of OQ apply:

$$q_D^{OQ}(t) = A(t) - B_D^{OQ}(t),$$

$$q_D^{OQ}(t) = \max_{0 \le s \le t} (A(t) - A(s) - (t - s)),$$

$$B_D^{OQ}(t) = \min_{0 \le s \le t} (A(s) + (t - s)), \text{ and}$$

$B_D^{OQ}(t) - B_D^{OQ}(s) \le t - s$ whenever $s \le t$ [24].

11. $t' = k + 1 + l \cdot N$, where l is an integer, is any time-slot when the cycle for EO k begins.

B. FFF Average Delay Within a Constant from OQ

In this section we will show that the average delay for FFF is within a constant delay of the average delay for an OQ switch for the same arriving traffic.

We will first compare FFF with the *delayed OQ*, which is an OQ having the same ordered arrivals as the second stage. We will show that FFF is work-conserving for full frames, and therefore services nearly as many full frames and as many cells as the delayed OQ model, with a queue size almost as small. This results in a bounded average delay difference with the delayed OQ model (Theorem 1). Then we compare the delayed OQ model with a regular OQ switch having the same packet arrivals as the first stage. Using a delay bound, we finally show that there exists a bounded average delay difference between FFF and an OQ switch (Theorem 2).

We start by establishing that whenever there is at least one full frame, the number of serviced full frames increases by one in the next cycle.

Lemma 1 If $ff(t') > 0$, then $B_{ff}(t' + N) = B_{ff}(t') + N$.

Proof: If $ff(t') > 0$, then there is at least one full frame in the IIs. Thus, according to the FFF algorithm, at least one full frame will be serviced.

❐

Since Lemma 1 shows that FFF is work-conserving for full frames, Lemma 2 shows that the number of serviced packets is close to the number of packets serviced by an OQ switch.

Lemma 2 $B_{ff}(t') \geq B_D^{OQ}(t') - N^2$

Proof: By induction on:
$t' = \{k+1-N, k+1, ..., k+1+l \cdot N\}$.

If $t' = k+1-N$, $B_{ff}(t') = 0$ and $B_D^{OQ}(t') \leq t' \leq 1 \leq N^2$, so the inequality holds. Now, let the inequality be true for t'.

Case 1: $ff(t') = 0$. Then,

$$B_{ff}(t') = A_{ff}(t')$$
$$= \sum_{i=1}^{N} N \cdot \left\lfloor \frac{A_i(t')}{N} \right\rfloor$$
$$\geq \left\lfloor \sum_{i=1}^{N} A_i(t') \right\rfloor - N(N-1)$$
$$\geq A(t') - N(N-1)$$
$$\geq B_D^{OQ}(t') - N(N-1)$$

$$B_{ff}(t'+N) \geq B_D^{OQ}(t') - N(N-1)$$
$$\geq (B_D^{OQ}(t'+N) - N) - N(N-1)$$
$$\geq B_D^{OQ}(t'+N) - N^2$$

Case 2: $ff(t') > 0$. Using Lemma 1,

$$B_{ff}(t'+N) = B_{ff}(t') + N$$
$$\geq B_D^{OQ}(t') - N^2 + N$$
$$\geq B_D^{OQ}(t'+N) - N^2$$

❒

Since Lemma 2 shows that the number of packets serviced by FFF is close to the delayed OQ in some sense, Lemma 3 concludes that the queue size for the delayed OQ and a constant.

Lemma 3 $q(t') - q_D^{OQ}(t') \leq N^2$ and $q_{ff}(t') - q_D^{OQ}(t') \leq N^2$.

Proof: Using the definitions and Lemma 2:

$$q_{ff}(t') - q_D^{OQ}(t') = [A_{ff}(t') - A(t')] + [B_D^{OQ}(t') - B_{ff}(t')]$$
$$\leq 0 + N^2$$

Similarly,

$$q(t') - q_D^{OQ}(t') = B_D^{OQ}(t') - B(t') \leq B_D^{OQ}(t') - B_{ff}(t') \leq N^2.$$

❒

The next two lemmas show that FFF efficiently uses bandwidth in order to remove the packets from the queues. Since FFF is work-conserving for full frames, Lemma 4 shows that if there are $ff(t')$ full frames at time t', then exactly $ff(t')$ full frames will be served in the next $ff(t')$ cycles. Lemma 5 generalizes this idea and considers what FFF does with the remaining packets when there are no full frames.

Lemma 4 $B_{ff}(t' + q_{ff}(t')) = A_{ff}(t')$

Proof: If $ff(t') = 0$, then by definition $q_{ff}(t') = 0$ and $B_{ff}(t') = A_{ff}(t')$. Otherwise, $ff(t') \geq 1$ and we can iteratively apply Lemma 1 $ff(t')$ times:

1. $ff(t') \geq 1$, thus $B_{ff}(t'+N) = B_{ff}(t') + N$ (from Lemma 1).
2. $ff(t'+N) \geq ff(t') - 1$, thus $B_{ff}(t'+2N) = B_{ff}(t') + 2N$,
3. $ff(t'+2N) \geq ff(t') - 2$, thus $B_{ff}(t'+3N) = B_{ff}(t') + 3N$,

etc.

Finally, $ff(t' + [ff(t') - 1] \cdot N) \geq 1$, thus

$B_{ff}(t' + ff(t') \cdot N) = B_{ff}(t') + ff(t') \cdot N = A_{ff}(t')$.

Hence $B_{ff}(t' + q_{ff}(t')) = A_{ff}(t')$.

❒

Lemma 5 $B(t' + q_{ff}(t') + N(2N-2)) \geq A(t')$

Proof: We already know from Lemma 4 that $B_{ff}(t' + q_{ff}(t')) = A_{ff}(t')$. Now we need to show that during the next $(2N-2)$ cycles, at least $A(t') - A_{ff}(t')$ cells will be serviced. Let's distinguish between two cases.

Case 1: during these $(2N-2)$ cycles, the N EIs are each serviced at least once in a round-robin fashion as *non-full* frames (i.e., $p_{nff}(k)$ is incremented at least N times).

Note that there is no full frame to service any more if non-full ones are serviced (since full frames have priority over non-full ones). This implies that every cell that is in a non-full frame at time t' is either serviced in the round-robin among non-full frames, or has been already serviced as part of a full frame that has been formed since.

Case 2: during these $2N-2$ cycles, there are at least $N-1$ full frames serviced (note that $N + (N-1) > 2N-2$, so there is no other case by Dirichlet's pigeon-hole principle).

Therefore, using Lemma 1 and Lemma 4:

$$B_{ff}(t' + q_{ff}(t') + N(2N-2)) \geq A_{ff}(t') + N(N-1)$$
$$\geq A(t')$$

By definition $B_{ff}(t) \leq B(t)$ for all t, and the result follows.

❒

We have shown that $B(t')$ was tracking $A(t')$ with a delay dependent on $q_{ff}(t')$. Since we have linked $q_{ff}(t')$ with $q_D^{OQ}(t')$, we can find a first bound on the average delay for FFF as a function of the average delay in the delayed OQ model. This bound will be useful in order to compare FFF with the regular (non-delayed) OQ model.

Theorem 1 The average delay for FFF is less than the average delay for the delayed OQ plus a constant $(3N^2 - N - 1)$.

Proof: We know that $q_{ff}(t') \leq N^2 + q_D^{OQ}(t')$ and $B(t' + q_{ff}(t') + N(2N-2)) \geq A(t')$ (Lemma 3 and Lemma 5).

Therefore, $B(t' + [q_D^{OQ}(t') + N^2] + N(2N-2)) \geq A(t')$ (because $B(t)$ is a non-decreasing function), i.e. $B(t' + q_D^{OQ}(t') + N(3N-2)) \geq A(t')$. But we also know that: $B_D^{OQ}(t' + q_D^{OQ}(t')) \leq B_D^{OQ}(t') + q_D^{OQ}(t') = A(t')$.

Hence, $B(t' + q_D^{OQ}(t') + N(3N-2)) \geq B_D^{OQ}(t' + q_D^{OQ}(t'))$.

This implies that the average delay for a cell coming at time t' is the delay that it would have under the delayed OQ algorithm, plus at most $N(3N-2)$. To see this, note that the average delay does not depend on the order in which the cells are picked. Thus, FFF has the same average delay as the FCFS algorithm which has the same cumulative number of arrivals and departures as FFF (hereafter called FFF-FCFS). FFF-FCFS would obviously satisfy the last formula. Therefore, the time spent by a new packet in the internal outputs is $q_D^{OQ}(t')$ with the delayed OQ, and at most $q_D^{OQ}(t') + N(3N-2)$ with FFF-FCFS. Hence the difference is bounded by $N(3N-2)$.

Finally, note that all computations up until now were for a t' that begins a cycle for output k. If we choose any nonnegative integer t, then let $t' = N \cdot \left\lfloor \frac{t-(k+1)}{N} \right\rfloor + k + 1$ be the beginning of the cycle to which t belongs, and let $C = N(3N-2) + (N-1) = 3N^2 - N - 1$. We get:

$$\begin{aligned} B(t + q_D^{OQ}(t) + C) &\geq B(t' + [A(t) - B_D^{OQ}(t)] + C) \\ &\geq B(t' + A(t') - B_D^{OQ}(t') - (N-1) + C) \\ &\geq B(t' + q_D^{OQ}(t') + N(3N-2)) \\ &\geq A(t') = A(t) \end{aligned}$$

Hence, $B(t + q_D^{OQ}(t) + C) \geq B_D^{OQ}(t + q_D^{OQ}(t))$ and the result is thus applicable to any time-slot t.

❐

Theorem 2 The average delay for FFF is less than the average delay for OQ plus a constant $4N^2 - 2$.

Proof: We compare the delays for OQ and for the delayed version of OQ. Let $D = N^2 + N - 1$. We'll first show that the delay for any cell in the delayed OQ is less than its delay for OQ plus D.

For any time-slot s, let $s'(s)$ be the time-slot that marks the beginning of the cycle to which s belongs: $s'(s) = N \cdot \left\lfloor \frac{s-(k+1)}{N} \right\rfloor + k + 1$, and $s - (N-1) \leq s'(s) \leq s$.

Then, according to the properties of the delayed OQ we have:

$$\begin{aligned} B_D^{OQ}(t) &= \min_{0 \leq s \leq t} [A(s) + (t-s)] \\ &= \min_{0 \leq s \leq t} [A(s'(s)) + (t-s)] \\ &\geq \min_{0 \leq s \leq t} [A^o(s'(s) - N^2) + (t-s)] \\ B_D^{OQ}(t) &\geq \min_{0 \leq s \leq t} [A^o(s-D) + (t-s)]. \end{aligned}$$

Let $\tau = s - D$. Since packets don't arrive before time-slot 0:

$$\begin{aligned} B_D^{OQ}(t) &\geq \min_{-D \leq \tau \leq t-D} [A^o(\tau) + (t-D) - \tau] \\ &\geq \min_{0 \leq \tau \leq t-D} [A^o(\tau) + (t-D) - \tau] \\ &\geq B^{OQ}(t-D) \end{aligned}$$

Hence, since the delayed OQ and OQ are both FCFS, the difference of delay for each cell between those two systems will be at most D, and the difference of average delay between FFF and OQ will be at most $(3N^2 - N - 1) + N^2 + N - 1 = 4N^2 - 2$.

❐

It is worth asking if the delay difference (approximately $4N^2$) is significant. For a high-speed router with 32 ports, OC768 (40 Gb/s line-rates) and a cell size of 64 bytes, $4N^2 = (4 \cdot 32^2) \cdot (13ns) = 52\mu s$ (the time taken for light to travel approximately 10 miles).

It is possible to improve this bound using a different algorithm that would take into account the number of cells present in the non-full frames, which FFF does not do. However, this would increase the complexity and the communication in the switch, and we believe that the trade-off is not worth it.

C. FFF Has the Same Throughput As OQ

Let's first provide a few definitions. Consider a switch with traffic arrival rates (ρ_{ik}) (from EI i to EO k), and total queueing size $Q(t)$, where t is the current time-slot.

1. The load of the arrival traffic is: $\rho = max\left(\max_k\left(\sum_{i=1}^{N} \rho_{ik}\right), \max_i\left(\sum_{k=1}^{N} \rho_{ik}\right)\right)$. The arrival traffic is said to be admissible if $\rho < 1$.
2. The switch is said to be strongly stable if $\lim_{t \to \infty} sup(E[Q(t)]) < \infty$ [6][22].
3. The switch is said to have 100% throughput if it is strongly stable whenever the arrival traffic is admissible. Similarly, it is said to have a throughput of x if it is strongly stable whenever $\rho < x$.

We have seen that there exists a bounded average delay difference between FFF and an OQ switch. As a consequence, FFF has the same throughput as OQ, as Theorem 3 illustrates.

Theorem 3 FFF and OQ have the same throughput.

Proof: Let $Q^{OQ}(t)$ and $Q^{FFF}(t)$ be the total queueing size at time-slot t in a switch implementing OQ and FFF, respectively. Also, let $C = 4N^3 + N^2 - 2N$, and assume that the cycle for EO k begins at $t'(t)$, with $t - (N-1) \leq t'(t) \leq t$. Using Lemma 3 and the proof for Theorem 2, we get:

$$
\begin{aligned}
q(t) &\le q(t'(t)) + N(N-1) \\
&\le q_D^{OQ}(t'(t)) + N^2 + N(N-1) \\
&\le [A(t'(t)) - B_D^{OQ}(t'(t))] + 2N^2 - N \\
&\le [A(t) - B_D^{OQ}(t)] + (N-1) + 2N^2 - N \\
&\le A(t) - B^{OQ}(t-D) + 2N^2 - 1 \\
&\le A^o(t) - B^{OQ}(t) + D + 2N^2 - 1 \\
&\le q^{OQ}(t) + 3N^2 + N - 2
\end{aligned}
$$

Taking into account both the buffering in the 3DQs for the N external outputs and the buffering in the VOQ_1s from the N external inputs, we get (using $q_k(t) \equiv q(t)$ for EO k):

$$
\begin{aligned}
Q^{FFF}(t) &\le N \sum_{k=1}^{N} q_k^{FFF}(t) + N \cdot N^2 \\
&\le N \sum_{k=1}^{N} q_k^{OQ}(t) + N(3N^2 + N - 2) + N^3 \\
&\le Q^{OQ}(t) + C
\end{aligned}
$$

Thus, $Q^{OQ}(t) \le Q^{FFF}(t) \le Q^{OQ}(t) + C$, since OQ is work-conserving. Hence the result.

❐

Note that this theorem is quite strong, because OQ is an ideal switch from a throughput point of view. In addition, note that the proof shows that at any time, the buffering needed with FFF is within a constant from the ideal buffering needed with OQ.

Property 4 FFF has 100% throughput with admissible Bernoulli i.i.d. arrival traffic.

Proof: OQ is known to have 100% throughput with admissible Bernoulli i.i.d. arrival traffic (this can be either proved directly, or by using the fact that OQ is work-conserving, thus $Q^{OQ}(t) \le Q^{MWM}(t)$, with Maximum Weight Matching (MWM) having 100% throughput [7]). The result follows using Theorem 3.

❐

Property 5 Assume that the arrival traffic patterns from EI i to EO k are (σ, ρ_{ik})-upper constrained. If the traffic is admissible, then FFF has 100% throughput.

Proof: Let $A_{ik}(n)$ be the cumulative number of arrivals for cells going from i to k at time-slot n, and let $A_k(n) = \sum_{i=1}^{N} A_{ik}(n)$. Then, by definition, for all $1 \le i, k \le N$ and $0 \le m \le n$, we have $A_{ik}(n) - A_{ik}(m) \le \rho_{ik} \cdot (n-m) + \sigma$. This implies that $A_k(n) - A_k(m) \le \rho \cdot (n-m) + N \cdot \sigma$, thus arrivals to any external output are $(N\sigma, \rho)$-upper constrained. Since $\rho < 1$, the maximum queue length size in OQ is $N\sigma$ [24]. The result follows using Theorem 3.

❐

VI. Conclusion

Over the last few years, there have been many results that show the conditions under which a single-stage crossbar switch with input queues and no speedup can achieve 100% throughput. To our knowledge, there have been no results that bound the difference in average delay between an ideal output queued switch and an input queued switch without speedup. Such bounds have only been possible when the switch runs with a speedup of at least two, has two stages of buffering (input and output queues) and uses a complicated (impractical) scheduling algorithm.

The two-stage switch introduced by Chang achieves a 100% throughput as well as a bound on the delay between it and an output queued switch. This is achieved without speedup and without a complicated scheduling algorithm, and therefore represents an important step towards efficient, high capacity switches with delay guarantees.

In its simplest form, the two-stage switch mis-sequences packets, hence motivating the work presented in this paper. The Full Frames First algorithm prevents mis-sequencing while maintaining the throughput and delay properties of the basic switch. While it clearly introduces more complexity, the algorithm appears practical at high speed.

We believe that the most interesting application of the two-stage switch is for use as the optical switching fabric in an otherwise electronic Internet router. The switch fabric in a router is generally limited by its power consumption, its size and the need for a complex scheduler. While optics can reduce both size and power, a single stage optical crossbar switch still requires an electronic scheduler. The two-stage switch can be incorporated without the need for a separate scheduler; because the switch moves through a deterministic sequence of configurations, and so scheduling packets consists only of distributing a timing reference to the linecards. Furthermore, since the two stages of the switch are configured according to a fixed sequence, it may be possible to replace them by a single switch that is configured once per time slot, with two cells transferred per configuration.

Acknowledgments

The authors would like to thank Balaji Prabhakar, Rui Zhang, Shang-Tse Chuang, Devavrat Shah, and the anonymous referees for their valuable comments.

References

[1] N. McKeown, M. Izzard, A. Mekkittikul, B. Ellersick and M. Horowitz, "The Tiny Tera: a packet switch core," *Hot Interconnects V*, Stanford University, August 1996.

[2] N. McKeown, "A fast switched backplane for a Gigabit Switched Router," *Business Communications Review*, 27(12), December 1997.

[3] Y. Tamir and G. Frazier, "High performance multi-queue buffers for VLSI communication switches," *Proc. of 15th Ann. Symp. on Comp. Arch.*, pp. 343-354, June 1988.

[4] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local-area net-

works," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319-352, 1993.

[5] N. McKeown, "iSLIP: a scheduling algorithm for input-queued switches," *IEEE Transactions on Networking*, Vol 7, No.2, April 1999.

[6] N. McKeown, V. Anantharam and J. Walrand, "Achieving 100% throughput in an input-queued switch," *Proc. of IEEE Infocom '96*, San Francisco, March 1996.

[7] A. Mekkittikul and N. McKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," *IEEE INFOCOM 98*, pp. 792-799, 1998.

[8] J.G.Dai and B.Prabhakar, "The throughput of data switches with and without speedup," *Proc. of IEEE Infocom 2000*, Tel Aviv, Israel, March 2000.

[9] S-T. Chuang, A. Goel, N. McKeown and B. Prabhakar, "Matching output queueing with a combined input output queued switch," *IEEE Journal of Selected Areas in Communication*, 17:1030--1039 [short version in *Proc. Infocom '99*].

[10] P. Krishna, N. Patel, A. Charny and R. Simcoe, "On the speedup required for work-conserving crossbar switches," *IEEE J. Sel. Areas in Communications*, Vol. 17, no. 6, pp. 1057-1066, June 1999.

[11] H.N. Gabow and R.E. Tarjan, "Faster scaling algorithms for network problems," *SIAM Journal on Computing*, 18:1013-1036, 1989.

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

[13] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," *IEEE INFOCOM 98*, vol. 2, pp. 533-539, 1998.

[14] D. Shah, P. Giaccone, and B. Prabhakar, "An implementable parallel scheduler for input-queued switches," *Proc. of Hot Interconnects IX*, Stanford, August 2001.

[15] K. Kar, T. V. Lakshman, D. Stiliadis and L. Tassiulas, "Reduced complexity input-buffered switches," *Proc. of Hot Interconnects VIII*, Stanford, August 2000.

[16] C.S. Chang, D.S. Lee and Y.S. Jou, "Load balanced Birkhoff - von Neumann switches, part I: one-stage buffering," *IEEE HPSR Conference*, Dallas, May 2001 [www.ee.nthu.edu.tw/~cschang/PartI.ps].

[17] F. Baker, "Requirements for IP version 4 routers", RFC 1812, June 1995.

[18] J.C.R. Bennett, C. Partridge and N. Shectman, "Packet reordering is not pathological network behavior," *IEEE/ACM Transactions on Networking*, Vol. 7, No. 6, December 1999, pp. 789-798.

[19] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," *ACM Computer Communication Review*, 32(1), January 2002.

[20] C.S. Chang, D.S. Lee and C.M. Lien, "Load balanced Birkhoff - von Neumann switches, part II: multi-stage buffering," *unpublished* [www.ee.nthu.edu.tw/~cschang/PartII.ps].

[21] S. Iyer and N. McKeown, "Making parallel packet switches practical," *Proc. of IEEE Infocom 2001*, Anchorage, Alaska, March 2001.

[22] E. Leonardi, M. Mellia, F. Neri and M. A. Marsan, "On the stability of input-queued switches with speed-up," *IEEE/ACM Transactions on Networking*, Vol.9, No.1, pp.104-118, February 2001.

[23] N. McKeown, A. Mekkittikul, V. Anantharam and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, Vol.47, No.8, August 1999.

[24] C.S. Chang, *Performance Guarantees in Communication Networks*, London: Springer-Verlag, 2000.

[25] U. Krishnamoorthy, P.M. Hagelin, J.P. Heritage, and O. Solgaard, "Surface-micromachined mirrors for scalable fiber optic switching applications," *Proc. of the SPIE Conference on MOEMS and Miniaturized Systems*, Santa Clara, California, September 2000, SPIE vol. 4178, pp. 270-277.

[26] A. Neukermans and R. Ramaswami, "MEMS technology for optical networking applications," *IEEE Communications Magazine*, vol. 39:1, Jan. 2001, pp. 62-69.

[27] C.J. Chang-Hasnain, "Tunable VCSEL," *IEEE Journal on Selected Topics in Quantum Electronics*, vol. 6:6, Nov.-Dec. 2000, pp. 978 -987.

[28] H. Yasaka, H. Sanjoh, H. Ishii, Y. Yoshikuni and K. Oe, "Repeated wavelength conversion of 10 Gb/s signals and converted signal gating using wavelength-tunable semiconductor lasers," *IEEE Journal of Lightwave Technology*, vol. 14, pp. 1042-1047, 1996.

[29] B. Pesach et al., "Free-space optical cross-connect switch by use of electroholography," *Applied Optics* 39(5), pp. 746-758, Feb. 2000.