The Inherent Queuing Delay of Parallel Packet Switches

Hagit Attiya and David Hay

Abstract—The *parallel packet switch (PPS)* extends the *inverse multiplexing* architecture and is widely used as the core of contemporary commercial switches. This paper investigates the inherent queuing delay introduced by the PPS's demultiplexing algorithm, responsible for dispatching cells to the middle-stage switches, relative to an optimal work-conserving switch. We first consider an $N \times N$ PPS without buffers in its input ports, operating at external rate R, internal rate $r < R$, and speedup (or overcapacity) S. We show that the inherent queuing delay of a symmetric and fault-tolerant PPS, where every demultiplexor may dispatch cells to all middle-stage switches, is $\Omega(N\frac{R}{r})$ if no information is shared between the input ports. Sharing information between the input ports significantly reduces this lower bound, even if the information is outdated. These lower bounds indicate that employing algorithms using slightly out-of-date information may greatly improve the PPS performance. When the PPS has buffers in its input ports, an $\Omega(N/S)$ lower bound holds if the demultiplexing algorithm uses only local information or the input buffers are small relative to the time an input port needs to learn the switch global information.

Index Terms—Internetworking, ATM, packet-switching networks.

1 Introduction

The need to support a large variety of applications with quality of service (QoS) guarantees demands high-capacity high-speed switching technologies [1]. An $N \times N$ *packet switch* routes packets arriving on N input ports at rate R to N output ports working at rate R. Packets are stored and transmitted in the switch as fixed-size *cells*; fragmentation and reassembly are done outside of the switch.

Switching cells in parallel is a natural approach to building switches with a very high external line rate and a large number of ports. A prime example of this approach is the *parallel packet switch* (in short, PPS) model [2], which is based on an architecture used in *inverse multiplexing* systems [3], [4], especially for ATM [5] (e.g., SCIMA [6]). The feasibility of these architectures motivated commercial vendors to use them at the heart of their switching architectures (for example, [7], [8], [9], [10]).

Specifically, a *parallel packet switch (PPS)* is a three-stage Clos network [11], with $K < N$ switches in its center stage, also called *planes*. Each plane is an $N \times N$ switch operating at rate $r < R$ and is connected to all the input ports on one side and to all the output ports on the other side (see Fig. 1). Since the planes operate at a lower rate, aggressive architectures can be supported, e.g., output-queued switches. The *speedup*, $S = \frac{Kr}{R}$, of the switch is its overcapacity; that is, the ratio of the aggregate capacity of the internal traffic lines, connected to an input or output port, to the capacity of its external line. Iyer and McKeown [2] consider a variant of the PPS architecture, called *input-buffered* PPS, having finite buffers in its input ports in addition to buffers in its output ports.

One of the key issues in the design of a PPS is balancing the load of switching operations among the middle-stage switches and, by that, utilizing the parallel capabilities of the switch. Load balancing is performed by a *demultiplexing algorithm* whose goal is to minimize the concentration of a disproportional number of cells in a small number of middle-stage switches.

Demultiplexing algorithms can be classified according to the amount and type of information they use. The strongest type of demultiplexing algorithms are *centralized* and make demultiplexing decisions based on global information about the status of the switch. Unfortunately, these algorithms must operate at a speed proportional to the aggregate incoming traffic rate and, therefore, they are impractical. At the other extreme, *fully distributed demultiplexing algorithms* rely only on the local information in the input port.[1] Due to their relative simplicity, they are common in contemporary switches. A realistic middle ground is what we call *u real-time distributed (u-RT) demultiplexing algorithms*, in which a demultiplexing decision is based on the local information and global information older than u time slots. Obviously, every fully distributed algorithm is also a u real-time distributed algorithm.

Switch architectures can be evaluated by their ability to provide different QoS guarantees. Important figures are the maximum/average queuing delay of cells, due to queuing cells within the switch, and the switch's throughput [13], [14], [15]. Another interesting performance figure is the *per-flow delay jitter* (or *cell delay variation*), namely, the maximal difference in queuing delay between two cells originating from the same input port and destined for the same output port [13], [16], [17].

• *The authors are with the Department of Computer Science, Technion—Israel Institute of Technology, Haifa 32000, Israel.*
E-mail: {hagit, hdavid}@cs.technion.ac.il.

Manuscript received 5 Sept. 2004; revised 29 Apr. 2005; accepted 26 July 2005; published online 26 July 2006.
Recommended for acceptance by A. Pietracaprina.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0221-0904.

1. These are also called *independent demultiplexing algorithms* [12].

1045-9219/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society

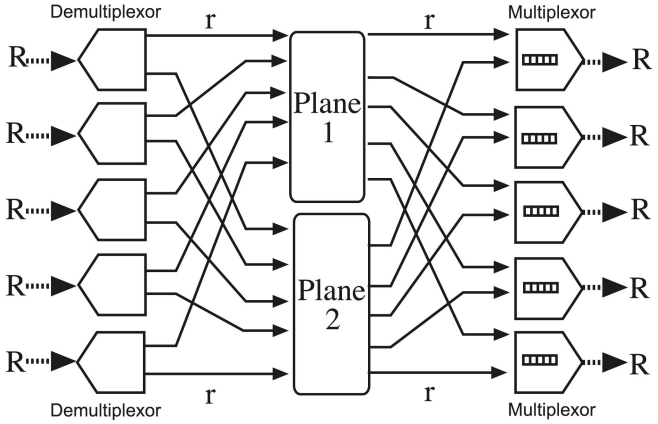

Fig. 1. A 5×5 PPS with two planes in its center stage and without buffers in the input ports.

The performance of a PPS is measured by comparison to an optimal *work-conserving (greedy)* switch that operates at rate R [18], [19], [20]. A work-conserving switch guarantees that if a cell is pending for output port j at time slot t, then some cell leaves from output port j at time slot t. This property prevents an output port from being idle unnecessarily and, by that, maximizes the switch throughput. The switch used for the comparison is called a *shadow* switch or a *reference* switch and it receives exactly the same incoming traffic as the PPS, namely, at any given time, the two switches receive the same cells, with the same destinations, on the same input ports. We assume that this shadow switch minimizes the queuing delay of cells (or minimizes the delay jitter, in case we measure the relative delay jitter). A primary candidate for a shadow switch is an output-queued switch operating at rate R. This is the reason this comparison is sometimes referred to as the ability of the PPS to *mimic an output-queued switch* [2], [21].

The *relative queuing delay* captures the influence of the parallelism of the PPS on the performance of the switch, depending on the different demultiplexing algorithms, and ignores the specific PPS *hardware* implementation. Our main contributions are lower bounds on the relative queuing delay and relative delay jitter of the PPS. Our lower bounds hold even when the PPS has to deal only with well-behaved traffic that obeys the *leaky-bucket* model [22], which makes our results stronger.

A *bufferless PPS* (i.e., without buffers at the input ports) with a fully distributed demultiplexing algorithm incurs the highest relative queuing delay and relative delay jitter. If some plane is utilized by all the demultiplexors, we prove a lower bound of $\left(\frac{R}{r}-1\right)N$ time slots on the relative queuing delay and relative delay jitter. Even in the unrealistic and failure-prone case where the planes are statically partitioned among the demultiplexors, the relative queuing delay and relative delay jitter are at least $\left(\frac{R}{r}-1\right)\frac{N}{S}$ time slots. Both lower bounds employ leaky-bucket flows with no bursts.

A bufferless PPS with a u-RT demultiplexing algorithm (for any u) has relative queuing delay and relative delay jitter of at least $\left(1-\overline{u}\frac{r}{R}\right)\frac{\overline{u}N}{S}$ time slots, where $\overline{u}=\min\{u,\frac{1}{2}\frac{R}{r}\}$. In contrast, Iyer and McKeown [2] present a centralized demultiplexing algorithm for a bufferless PPS with speedup $S \geq 2$, which achieves zero relative queuing delay.

TABLE 1
The Relative Queuing Delay (in Time Slots) of a Bufferless OC-192 PPS with $N = 1,024$ Ports and Speedup $S = 2$

Demultiplexor Type		Planes		
		OC-3	OC-12	OC-48
Fully-distributed	unpartitioned	64,512	15,360	3,072
	partitioned	32,256	7,680	1,536
1-RT		504	480	384
Centralized		0	0	0

An *input-buffered PPS* can support more elaborate demultiplexing algorithms since an arriving cell can either be transmitted to one of the middle stage switches or be kept in the input buffer. Under a u-RT demultiplexing algorithm, a switch with speedup $S \geq 2$ and input buffers larger than u can employ a centralized algorithm (e.g., [2]). In contrast, a fully distributed demultiplexing algorithm introduces relative queuing delay and relative delay jitter of at least $\left(1-\frac{r}{R}\right)\frac{N}{S}$ time slots, for any buffer size, under leaky-bucket flows with no bursts.

Our lower bound results show that the PPS architecture does not scale with increasing number of external ports (see Table 1 for specific instances). This is significant since great effort is currently invested in building switches with a large number of ports. Note that large relative queuing delays usually imply that the buffer sizes at the middle-stage switches and at the external ports should be large as well so that the cells can be queued.

For a bufferless PPS, it is important to notice that using a u-RT demultiplexing algorithm significantly reduces the lower bound on the relative queuing delay compared with a fully distributed demultiplexing algorithm. u-RT demultiplexing algorithms correspond to commercially used policies like *arbitrated crossbar switches* [23], in which a request is made by the input port and the cell is sent once a grant is received back from the arbiter. The separation between the lower bounds implies that employing u-RT demultiplexing algorithms in PPS may decrease the relative queuing delay dramatically and still be feasible for high-speed switches.

2 Related Work

Diverse QoS guarantees were first provided by an *output-queued switch*, in which the arriving cells are queued at the output side of the switch. However, output-queued switches require that the switch fabric and the output-side memory run at least at the aggregate speed of all the inputs and, hence, they are simply impractical for large and high-speed switches [20], [24], [25]. A *crossbar input-queued switch* overcomes the speed constraint by queuing the arriving cells at the input side of the switch and then forwarding them to the output side. Hybrid approaches have also been suggested: *virtual output queuing* in the input ports [1], [23] or queues in both input ports and output ports (*combined input-output queued* switches) [18], [21]. In all these architectures, the switch fabric needs to

switch cells at least at the external line rate, which may still be impractical for high-speed switches.

The *parallel packet switch* architecture was first considered by Iyer et al. [26], [27], [2], who evaluated its ability to mimic output-queued switches. Iyer et al. [26] introduced a centralized demultiplexing algorithm, called *CPA*, which allows a PPS with speedup $S \geq 2$ to mimic an FCFS output-queued switch with zero relative queuing delay. A centralized algorithm for multicast traffic was presented in [28]: A PPS with speedup $S \geq 2\sqrt{m}+1$ can handle multicast traffic with maximum fanout m and mimic an FCFS output-queued switch with zero relative queuing delay.

Unfortunately, these algorithms are impractical for real switches because they gather information from all the input ports in every scheduling decision. To overcome this problem, Iyer and McKeown [2] suggest a fully distributed algorithm that works with speedup $S = 2$ and mimics an FCFS output-queued switch with a relative queuing delay of $\lceil N\frac{R}{r} \rceil$ time slots. Another family of fully distributed algorithms, called *fractional traffic dispatch (FTD)* [29], works with switch speedup $S \geq \frac{K}{\lceil K/2 \rceil}$ and their relative queuing delay is at least $2NR/r$ time-slots.

The requirement for additional speedup is relaxed by adding buffers in the demultiplexors. For such an input-buffered PPS, Iyer and McKeown [2] suggest a fully distributed algorithm that allows a PPS with speedup $S = 1$ to mimic an FCFS output-queued switch with relative queuing delay of $\lceil 2N\frac{R}{r} \rceil$ time slots.

Mechanisms for efficient reassembly of cells in the multiplexors are discussed in [30], [31]. Although these mechanisms are crucial for implementing a PPS, our results are orthogonal to their performance.

Two proposed architectures have topology similar to the PPS. The *Parallel Switching Architecture (PSA)* [32] has several combined input-output queued (CIOQ) switches operating in parallel with no speedup. It is shown that an output-queued switch can be emulated with relatively low communication complexity. Our lower bounds extend to this architecture as well, although, in some cases (e.g., when the external rate is equal to the internal rate), the resulting lower bounds are trivial.

The *Switch-Memory-Switch (SMS)* routers [33], [34], [35] provide performance guarantees similar to an output-queued switch by $M > N$ parallel memories that reside between the input and output ports. Specifically, the authors present iterative scheduling algorithms that allow an SMS to emulate an output-queued switch with zero relative queuing delay. Since these algorithms use global information about the switch status in each time slot, they are considered centralized under our model and our results do not hold for them.

Leaky bucket traffic was employed by Giaccone et al. [36], in order to evaluate the relative queuing delay of specific scheduling algorithms for CIOQ switches with moderate speedup. Besides dealing with a different switching architecture, their work provides upper bounds while we prove lower bounds.

A lower bound on the speedup required for CIOQ switches to *exactly* mimic a shadow output-queued switch was presented in [18]. As in our lower bounds, this worst-case lower bound proof does not assume any statistical distribution on the arrival of packets at the switch; it uses specific leaky-bucket traffic with burstiness factor N.

3 A Model for Parallel Packet Switches

Cells arrive at the switch as a collection of *flows*, each from one input port to the same output port; the switch should preserve the order of cells within a flow and not drop cells. In addition, we assume that cells arrive at the switch and leave it in discrete *time slots*; in each time slot, at most one cell arrives at each input port and at most one cell leaves any output port. The internal lines of the switch operate at the lower rate $r < R$; for simplicity, we assume that $\frac{R}{r} = \lceil \frac{R}{r} \rceil$ and denote this value by r'.

The lower rate r enforces constraints on the switch [26]: A cell sent from an input port i to a plane k is transmitted over r' time slots; transmission takes place in the first time slot of this period and then the line between i and k is not utilized in the next $r'-1$ time slots. We refer to this constraint as the *input constraint*. Violating it causes a traffic rate greater than r on the internal line between an input-port and a plane. The *output constraint* is defined analogously, on the internal lines between the planes and the output ports.

In order to satisfy the input constraint, each input port should be connected to at least r' planes. In practice, the PPS is built with more planes in order to satisfy QoS demands and fault tolerance. The overcapacity of the PPS is its *speedup*, $S = K/r' = Kr/R$.

The relative queuing delay includes only queuing effects and excludes the different propagation delay between the two switches. This is achieved by assuming that cells are transmitted from/to the plane in the first time slot. To neglect delays caused by the additional stage of the PPS, a cell can leave the PPS in the same time slot it arrives at the output port, provided that no other cell is leaving the same output port on this time slot.

Definition 3.1. *The* relative queuing delay of a cell c *is the difference between the time slot at which cell* c *leaves the PPS and the time slot at which it leaves the optimal shadow switch. The* relative queuing delay of the PPS *is the maximum relative queuing delay of all cells* c *in any incoming traffic.*

The *per-flow delay jitter* of a switch is the maximal difference in queuing delay of cells originated in the same input port and destined for the same output port. The *relative* delay jitter is formally defined as follows:

Definition 3.2. *The* relative delay jitter of the PPS *is the difference between the per-flow delay jitter of the PPS and the per-flow delay jitter of an optimal shadow work-conserving switch.*

The behavior of the dispatching algorithm in every input port is modeled as a deterministic state machine, called a *demultiplexor*. Demultiplexors in different input ports may have different state sets; we denote by $\mathbb{S}_i$ the state set of the demultiplexor residing in input port i. A state is *applicable* if it can be reached in an execution of the demultiplexor.

A *switch configuration* is comprised of the states of all the demultiplexors and the content of all the buffers in the switch at a given time. A configuration is *applicable* if it can be reached in a legal execution of the switch. Since the switch does not have a predetermined initial configuration, we assume that, for every pair of applicable configurations C_1, C_2, there is an incoming traffic that causes the switch to transit from C_1 to C_2, namely, the set of applicable configurations induces a strongly connected graph.

When there are no buffers in the input ports, a cell arriving at the switch is immediately demultiplexed to one of the center stage switches, as modeled by the following definition:

Definition 3.3. *The demultiplexing algorithm of the demultiplexor residing at a bufferless input port i is the function $D_i : \{1, \ldots, N\} \times \mathbb{S}_i \rightarrow \{1, \ldots, K\}$, which gives a plane number, according to the incoming cell destination and the demultiplexor's state.*

This definition is easily extended to input-buffered PPS: When a cell arrives, the demultiplexor either sends the cell to one of the planes or keeps it in its buffer. In every time slot, the demultiplexor sends any number of buffered cells to the planes, provided that the rate constraints on the lines between the input port and any plane are preserved.

We refer to the buffer residing at input port i with finite size s as a vector $b_i \in \{1, \ldots, N, \perp\}^s$. An element of this vector contains the destination of the cell stored at the corresponding place in the buffer. Empty places in the buffer are indicated with $\perp$ in the vector. The size of the buffer at input port i is denoted $|b_i|$.

The demultiplexor state machine is changed to include the state of the input-port buffer. $\mathbb{B}_i$ denotes the set of the applicable states of the buffer residing in input port i. We refer to the set of states of the ith demultiplexor as $\mathbb{S}_i \times \mathbb{B}_i$. A switch configuration includes also the input buffer's content.

Definition 3.4. *The demutliplexing algorithm of the demultiplexor residing at input port i with input buffer is the function $D_i : \{1, \ldots, N, \perp\} \times \mathbb{S}_i \times \mathbb{B}_i \rightarrow \{1, \ldots, K, \perp\}^{|b_i|+1}$.*

This function receives as input the destination of the incoming cell ($\perp$ if no cell arrives), and the state of the demultiplexor. The function returns a vector of size $|b_i| + 1$ stating through which plane to send the cell in the corresponding place in the buffer; the last element of the vector refers to the incoming cell; $\perp$ indicates that the corresponding cell remains in the buffer.

4 The Relative Queuing Delay of a Bufferless PPS

The relative queuing delay of a PPS heavily depends on the information available to the demultiplexing algorithm. CPA [26] is a centralized demultiplexing algorithm with zero relative queuing delay, assuming the PPS has speedup $S \geq 2$ and the shadow output-queued switch follows a global FCFS discipline.[2] Practical demultiplexing algorithms must operate with local, or outdated, information about the status of the switch: flows waiting at other input ports, contents of the planes' buffers, etc. As we shall see, such algorithms incur nonnegligible queuing delay.

Our lower bounds are obtained even when flows do not saturate the switch. Such flows obey the *leaky-bucket constrained flows* model [22], which requires that the combined rate of flows sharing the same input port or the same output port does not exceed the external rate of that port by more than a fixed bound B, which is independent of time [1].

Definition 4.1. *An (R, B)* leaky-bucket traffic *satisfies the following inequality for every time slot t, integer $\tau \geq 1$, input port i, and output port j:*

$$A_i(t, t+\tau) \leq \tau R + B \text{ and } B_j(t, t+\tau) \leq \tau R + B,$$

where $A_i(t_1, t_2)$ is the number of cells arriving at input port i during time interval $[t_1, t_2)$, and $B_j(t_1, t_2)$ is the number of cells destined for output port j, arriving at the switch during time interval $[t_1, t_2)$.

The burstiness factor of the traffic B is also an upper bound on the size of the buffer needed for any work-conserving switch to handle such a traffic [37], [38].

In our lower bounds, the relative queuing delay is exhibited when cells that are supposed to leave the optimal shadow switch one after the other are concentrated in a single plane. The concentration scenario is captured by the following lemma:

Lemma 4.1. *Assuminge output port j's buffer is empty at time t and that m cells destined for the same output port j arrive at the switch during time interval $[t, t+s)$, out of them $c \leq m$ are sent through the same plane. Assume also that the incoming traffic is (R, B) leaky-bucket and no cells destined for j arrive at the switch during time interval $[t+s, t+m)$. Then:*

1. *The relative queuing delay of the PPS is at least $c\frac{R}{r} - (s + B)$ time slots.*
2. *The relative delay jitter of the PPS is at least $c\frac{R}{r} - (s + B)$ time slots.*

Proof. We compare the queuing delay of the cells in the PPS and in the shadow switch. Since the shadow switch is work-conserving and output port j's buffer is empty at time t, all m cells leave the switch exactly m time slots after the first cell is dispatched. On the other hand, a PPS completes this execution after at least cr' time slots because c cells are sent to the same plane and only one cell can be sent from this plane to the output port every r' time slots. Hence, the relative queuing delay is at least $cr' - m$ time slots. Since the incoming traffic is (R, B) leaky-bucket, $m \leq s + B$. Because $r' = \frac{R}{r}$, the relative queuing delay is at least $cr' - m \geq cr' - (s + B) = c\frac{R}{r} - (s + B)$ time slots, proving 1.

Let a be the last of the c cells sent from the plane to the output port and let i' be the input port from which a is sent. By the definition of a, it arrives at the PPS no later than time slot $t + s - 1$ and leaves the PPS not before time slot $t + cr'$.

Now, assume that there is a cell a' in the same flow (i', j), which arrives at the PPS when all the buffers are empty. Clearly, if no other cell destined for output port j arrives with cell a', a' leaves the PPS exactly one time slot

2. That is, cells should leave the switch in the same order they arrived, regardless of the flow they are in.

after its arrival. Hence, the delay jitter introduced by the PPS is at least $[(t+cr')-(t+s-1)]-1 = cr'-s$ time slots.

Recall that the maximum buffer size needed for any work-conserving switch to work under (R,B) leaky-bucket traffic is B. Therefore, a work-conserving switch which serves the incoming cells in an FCFS manner (e.g., an FCFS output-queued switch) introduces queuing delay and, therefore, also delay jitter of at most B time slots. Thus, the relative delay jitter between the PPS and the shadow switch is at least $(cr'-s)-B = c\frac{R}{r}-(s+B)$ time slots, proving 2. □

Lemma 4.1 does not depend on the scheduling policies of the planes and it only assumes that cells are not dropped.

4.1 Fully Distributed Demultiplexing Algorithms

We start by considering the simplest demultiplexing algorithm in which every input-port makes independent dispatching decisions.

Definition 4.2. *A* fully distributed demultiplexing *algorithm demultiplexes a cell arriving at time t according to the input port's local information in time interval $[0,t]$.*

The state transition function of the ith bufferless demultiplexor operating under a fully distributed demultiplexing algorithm is $S_i : \mathbb{S}_i \times \{1,\ldots,N\} \rightarrow \mathbb{S}_i$. That is, the demultiplexor state transitions depend only on the previous state of the demultiplexor and the destination of the incoming cell. If no cell arrives at a specific input port in a bufferless PPS, its demultiplexor does not change its state.

While the assumption that demultiplexors do not share *any* information is fairly strong, it allows us to illustrate our proof techniques.

The relative queuing delay of a PPS with a fully distributed demultiplexing algorithm strongly depends on the number of demultiplexors that can send a cell destined for the same output port through the same plane. To capture this switch characteristic, we call a demultiplexing algorithm d-partitioned if there is a plane k and an output port j such that at least d demultiplexors send a cell destined for output port j through plane k in one of their applicable configurations.

Lemma 4.2. *For every bufferless PPS, with a d-partitioned fully distributed demultiplexing algorithm, there is an incoming traffic without bursts in which d cells destined for the same output port j are sent through the same plane k during a time interval of length d. These cells are sent after output port j's buffer is empty.*

Proof. By the definition of a d-partitioned demultiplexing algorithm, there is an output port j and a plane k so that at least d demultiplexors send a cell destined for j through k in some applicable configuration. Let $I = \{i_1, i_2, \ldots i_d\}$ be the set of these demultiplexors and let $\sigma_i \in \mathbb{S}_i$ be the state of demultiplexor $i \in I$ in configuration C_i just before a cell is sent to plane k.

Consider traffic A from an arbitrary applicable configuration C which leads the switch to configuration C_i; such traffic exists since C and C_i are applicable and there is traffic that causes the switch to transit between any two applicable configurations. Let A_i be a traffic in which cells arrive at input port i exactly in the same time slots as in traffic A. Since the demultiplexing algorithm is fully distributed, demultiplexor i transits into σ_i. Note that, in A_i, at most one cell arrives at the switch in every time slot; therefore, this traffic has no bursts.

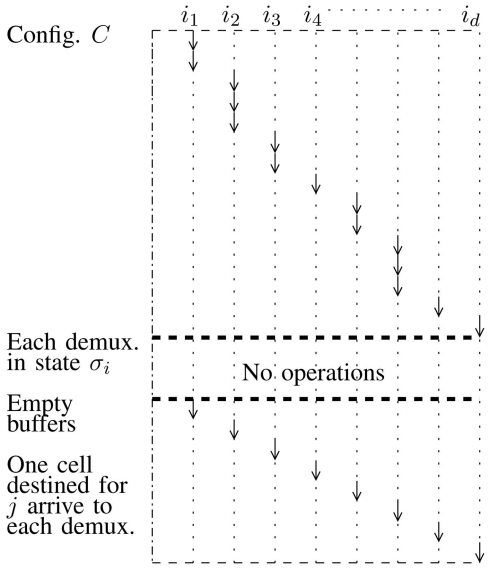

Fig. 2. Schematic view of the proof of Lemma 4.2.

Now, consider LB, a sequential composition of the traffics A_i, where $i \in I$. LB begins from configuration C and, sequentially for every $i \in I$, the same cells arrive at the switch in the same time slots as in traffic A_i until demultiplexor i reaches state σ_i. Then, no cells arrive at the switch until all the buffers in all the planes are eventually empty. Finally, d cells destined for output port j arrive, one after the other, at different input ports $i \in I$ (one cell in each time slot). Since the demultiplexing algorithm is fully distributed, each demultiplexor $i \in I$ remains in state σ_i and all the cells are sent through the same plane k (see Fig. 2).

LB has no bursts and the last d cells that arrive at the switch under traffic LB arrive during d consecutive time slots. These cells arrive at the switch after the buffer in output port j is empty. □

Combining Lemma 4.1 (with $c=d$, $s=d$, and $B=0$) and Lemma 4.2, we obtain the following theorem:

Theorem 4.3. *A bufferless PPS with a d-partitioned fully distributed demultiplexing algorithm has relative queuing delay and relative delay jitter of $\left(\frac{R}{r}-1\right)d$ time slots under traffic without bursts.*

The PPS input constraint implies that each demultiplexor must send incoming cells through at least r' planes. This implies that, even under static partitioning, each plane is used by $r'\frac{N}{K}$ demultiplexors, on the average. Hence, there is a plane k that is used by at least $r'\frac{N}{K} = \frac{R}{r}\frac{N}{K}$ demultiplexors in order to dispatch cells destined for a certain output port j. These observations imply the following lower bound:

Theorem 4.4. *A bufferless PPS with a fully distributed demultiplexing algorithm has relative queuing delay and relative delay jitter of* $(\frac{R}{r}-1)\frac{N}{S}$ *time slots under leaky-bucket traffic without bursts.*

Statically partitioning the planes among the different demultiplexors is failure-prone. For example, if a demultiplexor sends cells only through $d < K$ planes, damage in one plane (or the internal lines connected to it) causes more cell dropping than if all K planes are utilized. Therefore, fault tolerance dictates that each demultiplexor may send a cell destined for any output port through any plane. For such *unpartitioned* (or *N-partitioned*) *fully distributed demultiplexing algorithms*, Theorem 4.3 immediately implies:

Corollary 4.5. *A bufferless PPS with an unpartitioned fully distributed demultiplexing algorithm has relative queuing delay and relative delay jitter of* $(\frac{R}{r}-1)N$ *time slots under leaky-bucket traffic without bursts.*

Recall that Iyer and McKeown [27] present an unpartitioned fully distributed demultiplexing algorithm which allows a bufferless PPS with speedup $S \geq 2$ to mimic an FCFS output-queued switch with a relative queuing delay of $\lceil \frac{NK}{S} \rceil = \lceil \frac{R}{r} N \rceil$ time slots. Corollary 4.5 implies that this algorithm is optimal and no better result can be achieved for a bufferless PPS with speedup $S \geq 2$ unless information is shared among the demultiplexors.

4.2 u-RT Demultiplexing Algorithms

Centralized demultiplexing algorithms and fully distributed demultiplexing algorithms are both extreme, considering the amount of global information they use. The former has perfect knowledge of the switch status, while the latter uses no global information at all. It is more reasonable to consider an intermediate class of the demultiplexing algorithms in which only outdated global information is used:

Definition 4.3. *A* u real-time distributed (u-RT) demultiplexing algorithm *demultiplexes a cell arriving at time* t*, according to the input port's local information in time interval* $[0,t]$ *and to the switch's global information in time interval* $[0,t-u]$.

The state transition function of the ith bufferless demultiplexor operating under a u-RT demultiplexing algorithm is $S_i(t) : \mathbb{S}_i \times \mathbb{C}^{t-u+1} \times \{1,\ldots,N\} \to \mathbb{C}_i$, where t is the time slot in which S_i is applied, $\mathbb{C}$ is the set of all applicable switch configurations, and $\mathbb{C}^{t-u+1}$ is the cross product of $t-u+1$ such sets, one for each time slot in the interval $[0,t-u]$. Note that a demultiplexor state transition may depend on other demultiplexors' state transitions and on incoming flows to other input ports as long as these events occurred u time slots before the state transition. The state of a demultiplexor can change even if no cell arrives at the input port.

The additional global information allows a reduction of the relative queuing delay. For example, when a 1-RT demultiplexing algorithm receives $(R,0)$ leaky-bucket traffic, it has full information about the switch status and, therefore, it can emulate a centralized algorithm. Yet, lack of information about recent events yields a nonnegligible relative queuing delay caused by leaky-bucket traffic with a nonzero burstiness factor, as proven by the following lemma:

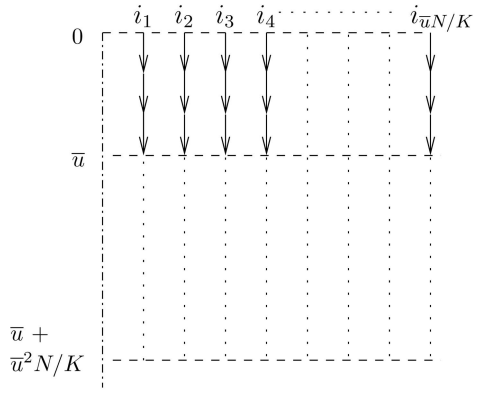

Fig. 3. Schematic view of the proof of Lemma 4.6.

Lemma 4.6. *For every bufferless PPS with a u-RT demultiplexing algorithm, there is an incoming traffic with burstiness factor* $\overline{u}^2\frac{N}{K}-\overline{u}$, $\overline{u}=\min\{u,\frac{1}{2}\frac{R}{r}\}$, *such that*

1. *At least $\overline{u}N/K$ cells destined for the same output port j are sent through the same plane k during a time interval of length $\overline{u}$.*
2. *Before these cells arrive at the switch, output buffer j is empty.*
3. *No cells destined for output port j arrive at the switch for another $\overline{u}^2N/K$ time slots.*

Proof. Let C be the switch configuration at time t and assume that, at this time, all the buffers in the switch are empty. Let σ_i be the state of each demultiplexor i in this configuration. Now, consider leaky-bucket traffic LB, starting in configuration C at time slot t. LB consists of one cell destined for output port j arriving at each input port in every time slot in the interval $[t,t+\overline{u})$. After time slot $t+\overline{u}-1$, no cells arrive at the switch for $\overline{u}N-1$ time slots. Note that LB is $(R,\overline{u}N-1)$ leaky-bucket traffic since, for any $\tau \geq 1$ and time interval $[t,t+\tau)$, the total number of cells arriving at the switch is bounded by $\tau+(\overline{u}N-1)$.

Since $\overline{u} \leq \frac{1}{2}\frac{R}{r} \leq \frac{R}{r}$, the input constraint implies that two cells that arrive at the same input port are not sent through the same plane. Hence, there is a plane k used by a set I of at least $\frac{\overline{u}}{K}N$ input ports. (Since the PPS speedup is at least 1, $\frac{\overline{u}}{K}N < \frac{R}{rK}N \leq N$.) Let t_i be the first time slot after t in which a cell destined for j is sent from input port i to plane k. Since $i \in I$, $t_i \leq \overline{u}$.

Consider now another traffic LB_I that begins in configuration C in which a cell arrives at input port i in time slot t' if and only if $i \in I$ and, in traffic LB, a cell arrives at i in the same time slot t'. Like LB, LB_I is an $(R,\overline{u}^2\frac{N}{K}-\overline{u})$ leaky-bucket traffic (see Fig. 3).

When LB_I begins, the state of every demultiplexor $i \in I$ is σ_i. Since $t_i \leq \overline{u} \leq u$, each demultiplexor $i \in I$ goes through the same state transitions as if traffic LB had arrived at the switch. Hence, at time slot t_i, each demultiplexor $i \in I$ sends the cell destined for output port j through plane k.

Note that, during time interval $[t,t+\overline{u})$, at least $\overline{u}N/K$ cells are sent through the same plane. □

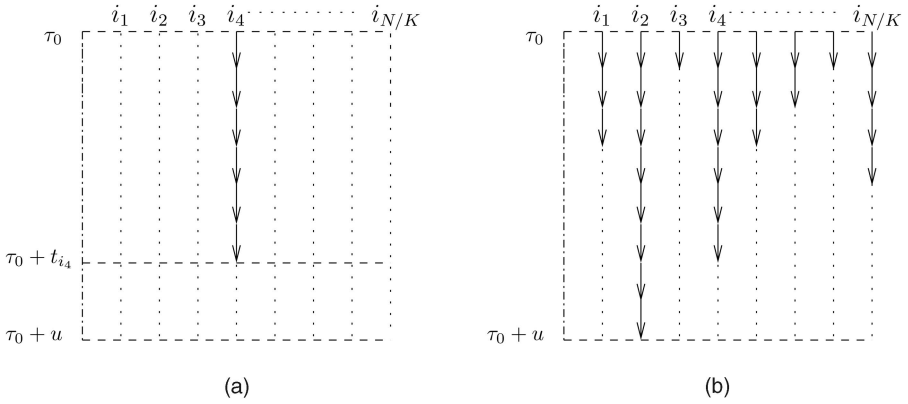

Fig. 4. Illustration for the proof of Theorem 5.2: (a) traffic LB_{i_4} and (b) traffic LB.

Applying Lemma 4.1 with $c = \overline{u}N/K$, $s = \overline{u}$ and $B = \overline{u}^2\frac{N}{K} - \overline{u}$ yields the following theorem:

Theorem 4.7. *A bufferless PPS with a u-RT demultiplexing algorithm has relative queuing delay and relative delay jitter of $\left(1 - \overline{u}\frac{r}{R}\right)\frac{\overline{u}N}{S}$ under leaky-bucket traffic with burstiness factor $\overline{u}^2\frac{N}{K} - \overline{u}$, where $\overline{u} = \min\{u, \frac{1}{2}\frac{R}{r}\}$.*

By substituting the minimal value $u = 1$, we get the following general result:

Corollary 4.8. *A bufferless PPS with u-RT demultiplexing algorithm, $u \geq 1$, has relative queuing delay and relative delay jitter of $\left(1 - \frac{r}{R}\right)\frac{N}{S}$ time slots under leaky-bucket traffic with burstiness factor $\frac{N}{K} - 1$.*

5 The Relative Queuing Delay of an Input-Buffered PPS

When measuring relative queuing delay in an input-buffered PPS, the queuing of cells both in the input ports' buffers and the planes' buffers of the PPS should be compared to the queuing of cells in the output ports' buffers of the shadow switch. Generally, input buffers increase the flexibility of the demultiplexing algorithms, which leads to smaller lower bounds.

The size of the input buffers affects the relative queuing delay in an input-buffered PPS under u-RT demultiplexing algorithms. A PPS that can store u cells in each input port is able to support a u-RT demultiplexing algorithm that guarantees relative queuing delay of at most u time slots by simulating the CPA algorithm [26]. Note that CPA assumes the PPS is a globally FCFS switch, i.e., cells leave an output port in an FCFS order, regardless of the input port from which they originate.

Theorem 5.1. *There is a u-RT demultiplexing algorithm for a globally FCFS input buffered PPS with buffer size $\geq u$, speedup $S \geq 2$, and a relative queuing delay of at most u time slots.*

This algorithm may be impractical, yet it demonstrates that a lower bound of $\Omega(N/S)$ time slots does not hold when the input buffers are sufficiently large. When buffers are smaller than u, we show that a globally FCFS input-buffered PPS has relative queuing delay of $\left(1 - \frac{r}{R}\right)\frac{N}{S}$ time slots under leaky-bucket traffic with burstiness factor $u(\frac{N}{K} - 1)$:

Theorem 5.2. *A globally FCFS input-buffered PPS with u-RT demultiplexing algorithm and input buffers' sizes smaller than u has relative queuing delay of $\left(1 - \frac{r}{R}\right)\frac{N}{S}$ time slots under leaky-bucket traffic with burstiness factor $u(\frac{N}{K} - 1)$.*

Proof. Let C be the switch configuration at time τ_0 and assume that, at this time, all the buffers in the switch are empty. We use, for every demultiplexor i, leaky-bucket traffic LB_i. In LB_i, cells destined for a fixed output port j arrive at input port i, one cell in each time slot, until the first cell destined for output port j is sent to one of the planes. This execution takes at most u time slots because, otherwise, demultiplexor i is queuing in its input buffer more cells than its capacity. Note that LB_i is a leaky-bucket traffic with no bursts.

Since the PPS is FCFS and only cells of traffic LB_i arrive at the switch, the first cell to leave de-multiplexor i's input buffer is the first cell of traffic LB_i. We denote this cell by a_i, the plane it is sent to by k_i, and the time it is sent by t_i.

Since $K < N$, there exists a plane k and a set of demultiplexors $I \subseteq \{1, \ldots, N\}$ of size N/K, such that $k_i = k$ for every $i \in I$. Let $T = \max\{t_i | i \in I\}$.

Now, compose all traffics LB_i for $i \in I$, as described in Fig. 4, and append the time interval $(T, T + u(N/K - 1)]$, in which no cells arrive at the switch. LB denotes the composite traffic, starting from configuration C at time τ_0, which is a $(R, u(\frac{N}{K} - 1))$ leaky-bucket traffic.

Every demultiplexor $i \in I$ goes through the same state transitions in response to LB_i and LB since composing the traffic does not change the switch configurations in time interval $(0, \tau_0)$, $t_i - u \leq \tau_0$, and its local information is identical in LB_i and LB. Hence, demultiplexor i sends the cell a_i to plane k in time slot $t_i \leq \tau_0 + u$.

Since the shadow switch is FCFS as well, the N/K cells $\{a_i | i \in I\}$ leave the switch consecutively and prior to all the other cells in LB. Because the shadow switch is work-conserving, the last of the these cells leaves the switch after N/K time slots. On the other hand, it takes $\frac{N}{K}\frac{R}{r}$ time slots to send the last cell from plane k to output port j. Hence, the relative queuing delay and relative delay jitter are $\frac{R}{r}\frac{N}{K} - \frac{N}{K} = \frac{R-r}{r}\frac{N}{K} = \left(1 - \frac{r}{R}\right)\frac{N}{S}$ time slots. □

Unlike in u-RT demultiplexing algorithms, where the relative queuing delay depends on the size of the buffers in the input-ports, if the demultiplexing algorithm is fully distributed a high concentration can happen, regardless of the buffer size. This situation is captured by the following lemma:

Lemma 5.3. *For every input-buffered PPS with a fully distributed demultiplexing algorithm, there is an incoming traffic without bursts, in which N/K cells destined for the same output port j are sent through the same plane k during time interval of length N/K. These cells are sent after output port j's buffer is empty.*

Proof. Let C be a switch configuration in which all the buffers in the switch are empty. Denote the state of demultiplexor i in configuration C by (σ_i^0, b_i^0). Clearly, $b_i^0 = \{\perp\}^{|b_i|}$.

For every demultiplexor i, consider traffic LB_i, starting in configuration C, in which one cell destined for a fixed output port j arrives at the switch in the first time slot and, then, no cells arrive until the first cell is sent to one of the planes. This execution takes $t_i \leq N/S$ time slots; otherwise, LB_i is causing a relative queuing delay greater then N/S time slots. Let k_i be the plane through which demultiplexor i sends the injected cell and (σ_i^f, b_i^f) be the demultiplexor state just before this cell is sent. LB_i is a leaky-bucket traffic with no bursts, since one cell arrives at the switch.

Since $K < N$, there exists a plane k and a set of N/K demultiplexors $I = \{i_1, \ldots, i_{N/K}\}$ such that $k_i = k$ for every $i \in I$.

Now, consider another leaky-bucket traffic, LB, which begins in configuration C. In LB, for every time-slot $t \in \{1, \ldots, N/K\}$, one cell, which is destined for output port j, arrives at input port $i_t \in I$. Note that, in every time slot, at most one cell arrives at the switch; therefore, this traffic has no bursts.

Since the PPS is working under a fully distributed demultiplexing algorithm and all the buffers are empty in configuration C, a demultiplexor $i \in I$ does not change its states until the first cell arrives. Before LB and LB_i begin, demultiplexor i is in state (σ_i^0, b_i^0), and its individual flow under LB is exactly the same as under LB_i (only one cell destined for output port j arrives). Therefore, demultiplexor $i \in I$ changes its state to (σ_i^f, b_i^f) and sends its cell to plane k. □

Applying Lemma 4.1 with $c = N/K$, $s = N/K$, and $B = 0$ yields the following theorem:

Theorem 5.4. *An input-buffered PPS, with a fully distributed demultiplexing algorithm, has relative queuing delay and relative delay-jitter of $\left(1 - \frac{r}{R}\right)\frac{N}{S}$ time slots under leaky-bucket traffic without bursts.*

6 Discussion

This paper studies the worst-case queuing delay and delay jitter induced by the demultiplexing algorithm of a parallel packet switch, relative to an optimal work-conserving switch. This *competitive* approach, typically used to evaluate online algorithms, is appealing because it does not require stochastic characterization of the incoming traffic. Recently, we extended the lower bounds to hold with high probability for the average relative queuing delay of randomized demultiplexing algorithms [39].

Our results imply that larger buffers are needed when building a PPS with an increasing number of external ports and have significant implications on the memory architecture of the switch. The gap between the lower bounds when demultiplexors share information and when they do not suggests that using outdated information improves the PPS performance. Our work leaves open the problem of designing practical algorithms that share only partial outdated information.

In an extreme case, the only global information used is a *shared clock*. It is interesting to investigate whether such synchronization reduces the relative queuing delay of fully distributed algorithms.

Leaky-bucket traffic [1], [22] is used to restrict the incoming traffic and show that the lower bounds hold even under well-behaved traffic. One can also use the metaphor of an adversary controlling the injection of cells, as was done in the context of switching networks. Two models were suggested to restrict the injected flows from flooding the network. Andrews et al. [40] restrict the adversary using the same concepts as leaky-bucket traffic; the flows causing the relative queuing delay results in this paper satisfy the constraints of Andrews et al.'s *bounded adversaries*. Borodin et al. [41] enforce strictly stronger restrictions on their adversaries [42]; our flows satisfy these stronger restrictions as well.

Traffic shaping with low jitter may favor nonwork-conserving switches [13], [43] and, therefore, it is interesting to compare with such switches. However, when cells are not dropped inside the switch, a nonwork-conserving shadow switch can degrade to work at rate r, making the comparison meaningless.

Jitter regulators that capture jitter control mechanisms use an internal buffer to shape the traffic [16], [44], [45]. Since we showed that the outgoing traffic from the PPS is not nicely shaped, it might be possible to derive lower bounds on the size of the internal buffer.

Acknowledgments

A previous version of this paper appeared in the *Proceedings of the Third IFIP International Conference on Theoretical Computer Science* (TCS 2004). A short version of the paper also appeared in the *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA 2004 Revue). Part of the work was performed while H. Attiya was at Dune Networks.

References

[1] A. Charny, "Providing QoS Guarantees in Input Buffered Crossbar Switches with Speedup," PhD dissertation, Massachusetts Inst. of Technology, Sept. 1998.

[2] S. Iyer and N. McKeown, "Analysis of the Parallel Packet Switch Architecture," *IEEE/ACM Trans. Networking*, vol. 11, no. 2, pp. 314-324, Apr. 2003.

[3] J. Duncanson, "Inverse Multiplexing," *IEEE Comm. Magazine*, vol. 32, no. 4, pp. 34-41, Apr. 1994.

[4] P. Fredette, "The Past, Present, and Future of Inverse Multiplexing," *IEEE Comm. Mag.*, vol. 32, no. 4, pp. 42-46, Apr. 1994.

[5] *Inverse Multiplexing for ATM (IMA) Specification*, version 1.1, AF-PHY-0086.001, ATM Forum, Mar. 1999.

[6] F.M. Chiussi, D.A. Khotimsky, and S. Krishnan, "Generalized Inverse Multiplexing of Switched ATM Connections," *Proc. IEEE Globecom Conf.,* pp. 2905-2912, 1998.
[7] "Inverse Multiplexing over ATM (IMA): A Breakthrough WAN Technology for Corporate Networks," 3Com Corp., 1997, http://www.mcoecn.org/WhitePapers/3COM-Inverse-Multiplexing-ATM. pdf.
[8] *ATM Switch Router Software Configuration Guide,* 12.0(22)W5(25), Cisco Systems, 2004.
[9] "Inverse Multiplexing for ATM, Expanding the Revenue Opportunities for Converged Services over ATM," Lucent Technologies, 2001.
[10] "Inverse Multiplexing over ATM Works Today," PMC-Sierra, http://www.electronicstalk.com/news/pmc/pmc121.html, 2002.
[11] C. Clos, "A Study of Non-Blocking Switching Networks," *Bell System Technical J.,* pp. 406-424, 1953.
[12] S. Iyer, "Analysis of a Packet Switch with Memories Running Slower than the Line Rate," master's thesis, Stanford Univ., May 2000.
[13] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet-Switched Networks," *Proc. IEEE,* vol. 83, no. 10, pp. 1374-1396, Oct. 1995.
[14] J. Turner and N. Yamanaka, "Architectural Choices in Large Scale ATM Switches," *IEICE Trans. Comm.,* vol. E81-B, no. 2, pp. 120-137, Feb. 1998.
[15] C. Minkenberg, R. Luijten, F. Abel, W. Denzel, and M. Gusat, "Current Issues in Packet Switching Design," *ACM SIGCOMM Computer Comm. Rev.,* vol. 33, no. 1, pp. 119-124, Jan. 2003.
[16] Y. Mansour and B. Patt-Shamir, "Jitter Control in QoS Networks," *IEEE/ACM Trans. Networking,* vol. 9, no. 4, pp. 492-502, Aug. 2001.
[17] H. Zhang, "Providing End-to-End Performance Guarantees Using Non-Work-Conserving Disciplines," *Computer Comm.,* vol. 18, no. 10, Oct. 1995.
[18] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching Output Queueing with a Combined Input Output Queued Switch," *Proc. IEEE Conf. Computer Comm. (INFOCOM),* pp. 1169-1178, 1999.
[19] L. Kleinrock, *Queuing Systems,* vol. 2. John Wiley & Sons, 1975.
[20] P. Krishna, N.S. Patel, A. Charny, and R. Simcoe, "On the Speedup Required for Work-Conserving Crossbar Switches," *IEEE J. Selected Areas in Comm.,* vol. 17, no. 6, pp. 1057-1066, June 1999.
[21] B. Prabhakar and N. McKowen, "On the Speedup Required for Combined Input and Output Queued Switching," *Automatica,* vol. 35, no. 12, pp. 1909-1920, Dec. 1999.
[22] J.S. Turner, "New Directions in Communications (or Which Way to the Information Age?)," *IEEE Comm. Magazine,* vol. 24, no. 10, pp. 8-15, Oct. 1986.
[23] Y. Tamir and H. Chi, "Symmetric Crossbar Arbiters for VLSI Communication Switches," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 1, pp. 13-27, Jan. 1993.
[24] M. Hluchyj and M. Karol, "Queueing in High-Performance Packet Switching," *IEEE J. Selected Areas Comm.,* vol. 6, no. 12, pp. 1587-1597, Dec. 1988.
[25] M. Karol, M. Hluchyj, and S. Morgan, "Input versus Output Queueing on a Space-Division Packet Switch," *IEEE Trans. Comm.,* vol. 35, no. 12, pp. 1347-1356, Dec. 1987.
[26] S. Iyer, A. Awadallah, and N. McKeown, "Analysis of a Packet Switch with Memories Running at Slower than the Line Rate," *Proc. IEEE Conf. Computer Comm. (INFOCOM),* pp. 529-537, 2000.
[27] S. Iyer and N. McKeown, "Making Parallel Packet Switches Practical," *Proc. IEEE Conf. Computer Comm. (INFOCOM),* pp. 1680-1687, 2001.
[28] S. Iyer and N. McKeown, "On the Speedup Required for a Multicast Parallel Packet Switch," *IEEE Comm. Letters,* vol. 5, no. 6, pp. 269-271, June 2001.
[29] D. Khotimsky and S. Krishnan, "Stability Analysis of a Parallel Packet Switch with Bufferless Input Demultiplexors," *Proc. IEEE Int'l Conf. Comm. (ICC),* pp. 100-106, 2001.
[30] D. Khotimsky and S. Krishnan, "Evaluation of Open-Loop Sequence Control Schemes for Multi-Path Switches," *Proc. IEEE Int'l Conf. Comm. (ICC),* pp. 2116-2120, May 2002.
[31] A. Aslam and K. Christensen, "A Parallel Packet Switch with Multiplexors Containing Virtual Input Queues," *Computer Comm.,* vol. 27, no. 13, pp. 1248-1263, 2004.
[32] S. Mneimneh, V. Sharma, and K. Siu, "Switching Using Parallel Input-Output Queued Switches with No Speedup," *IEEE/ACM Trans. Networking,* vol. 10, no. 5, pp. 653-665, 2002.
[33] S. Sharif, A. Aziz, and A. Prakash, "An $O(\log^2 N)$ Parallel Algorithm for Output Queuing," *Proc. IEEE Conf. Computer Comm. (INFOCOM),* 2002.
[34] A. Prakash, A. Aziz, and V. Ramachandran, "A Near Optimal Schedule for Switch-Memory-Switch Routers," *Proc. ACM Symp. Parallelism in Algorithms and Architectures (SPAA),* pp. 343-352, 2003.
[35] A. Prakash, A. Aziz, and V. Ramachandran, "Randomized Parallel Schedulers for Switch-Memory-Switch Routers: Analysis and Numerical Studies," *Proc. IEEE Conf. Computer Comm. (INFOCOM),* 2004.
[36] P. Giaccone, E. Leonardi, B. Prabhakar, and D. Shah, "Delay Bounds for Combined Input-Output Switches with Low Speed-Up," *Performance Evaluation,* vol. 55, nos. 1-2, pp. 113-128, 2004.
[37] J.Y. LeBoudec, "Network Calculus Made Easy," Technical Report EPFL-DI 96/218, École Polytechnique Fédérale, Lausanne (EPFL), 1996.
[38] R.L. Cruz, "A Calculus for Network Delay, Part I: Network Elements in Isolation," *IEEE Trans. Information Theory,* vol. 37, no. 1, pp. 114-131, Jan. 1991.
[39] H. Attiya and D. Hay, "Randomization Does Not Reduce the Average Delay in Parallel Packet Switches," *Proc. ACM Symp. Parallelism in Algorithms and Architectures (SPAA),* pp. 11-20, 2005.
[40] M. Andrews, B. Awerbuch, A. Fernandez, J. Kleinberg, T. Leighton, and Z. Liu, "Universal Stability Results for Greedy Contention-Resolution Protocols," *J. ACM,* vol. 48, no. 1, pp. 39-69, 2001.
[41] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D.P. Williamson, "Adversarial Queueing Theory," *J. ACM,* vol. 48, no. 1, pp. 13-38, 2001.
[42] A. Rosen, "A Note on Models for Non-Probabilistic Analysis of Packet Switching Network," *Information Processing Letters,* vol. 84, no. 5, pp. 237-240, Dec. 2002.
[43] C. Wu, J. Jiau, and K. Chen, "Characterizing Traffic Behavior and Providing End-to-End Service Guarantees within ATM Networks," *Proc. IEEE Conf. Computer Comm. (INFOCOM),* pp. 336-344, 1997.
[44] S. Keshav, *An Engineering Approach to Computer Networking.* Addison-Wesley, 1997.
[45] H. Zhang and D. Ferrari, "Rate-Controlled Service Disciplines," *J. High Speed Networks,* vol. 3, no. 4, pp. 389-412, 1994.

Hagit Attiya received the BSc degree in mathematics and computer science from the Hebrew University of Jerusalem in 1981 and the MSc and PhD degrees in computer science from the Hebrew University of Jerusalem in 1983 and 1987, respectively. She is presently a professor in the Department of Computer Science at the Technion—Israel Institute of Technology. Before joining the Technion, she was a postdoctoral research associate at the Laboratory for Computer Science at MIT. Her general research interests are distributed and parallel computation. More specific interests include fault tolerance, and timing-based and asynchronous algorithms.

David Hay received the BA degree in computer science from the Technion—Israel Institute of Technology in 2001. He is currently a PhD student in the department of computer science, at Technion. His main research interests are algorithmic aspects of high-performance switches and routers; in particular, QoS provisioning, competitive analysis, and distributed scheduling. Between 1999 and 2002, he was with IBM Haifa Research Labs.