THE INHERENT QUEUING DELAY OF PARALLEL PACKET SWITCHES

(Extended Abstract)

Hagit Attiya* and David Hay
Department of Computer Science
Technion — Israel Institute of Technology
Haifa 32000, Israel
{hagit,hdavid}@cs.technion.ac.il

Abstract The *parallel packet switch (PPS)* is extensively used as the core of contemporary commercial switches. This paper investigates the inherent queuing delay and delay jitter introduced by the PPS's demultiplexing algorithm, relative to an optimal work-conserving switch.

We show that the inherent queuing delay and delay jitter of a symmetric and fault-tolerant $N \times N$ PPS, where every demultiplexing algorithm dispatches cells to all the middle-stage switches is $\Omega(N)$, if there are no buffers in the PPS input-ports. If the demultiplexing algorithms dispatch cells only to part of the middle-stage switches, the queuing delay and delay jitter are $\Omega(N/S)$, where S is the PPS speedup. These lower bounds hold unless the demultiplexing algorithm has full and immediate knowledge of the switch status. 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.

Keywords: Inverse multiplexing, Leaky-bucket traffic, Packet switching, Clos networks, Queuing delay, Delay jitter, Load balancing

1. Introduction

The need to support a large variety of applications with *quality of service (QoS)* guarantees demands high-capacity high-speed switching technologies [5]. 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

*Part of the work was performed while this author was at Dune Networks.

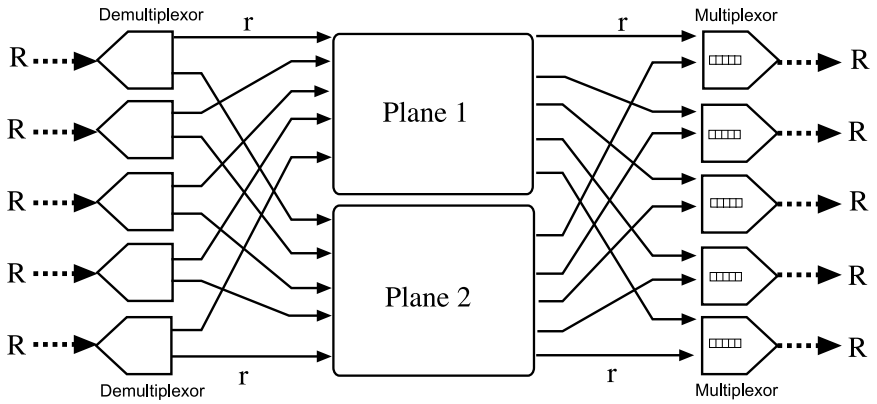

Figure 1. A 5×5 PPS with 2 planes in its center stage, without buffers in the input-ports

switch as fixed-size *cells*; fragmentation and reassembly are done outside of the switch. Cells arrive to the switch as a collection of *flows* from one input port to the same output-port; the switch should preserve the order of cells within a flow and not drop cells. Since there may be conflicts between different flows, certain amount of buffering within the switch may be needed. The location of the buffers, their sizes, and their management, depend on the specific architecture of the switch.

Switching cells in parallel is a natural approach to build switches with very high external line rate, and with large number of ports. A prime example of this approach is the *parallel packet switch* (in short, PPS) model [14], which is based on an architecture used in *inverse multiplexing* systems [12, 11], and especially on the *inverse multiplexing for ATM (IMA)* technology [2, 6]. The feasibility of the PPS and IMA architectures motivated commercial vendors to implement them at the heart of their switching architecture.

A *parallel packet switch (PPS)* is a three-stage Clos network [8], 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 Figure 1). The *speedup* S of the switch 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, namely, $S = \frac{Kr}{R}$. Iyer and McKeown [15] 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.

Perhaps the key issue 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 algorithms*, in which every demultiplexing decision is done based on global information about the status of the switch. Unfortunately, these algorithms must operate at a speed proportional to the aggregate incoming flows 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 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.

1.1 Evaluating PPS Performance. Switch architectures are evaluated by their ability to provide different QoS guarantees. Important figures are the maximum/average queuing delay of cells (i.e., delay resulting from queuing cells within the switch) and the switch's throughput. Another interesting performance number is the *per-flow delay jitter* (or *cell delay variation*), namely, the maximal difference in queuing delay between two cells in the same flow [25, 24, 20].

The performance of a PPS is measured by comparison to an optimal *work-conserving (greedy)* switch, operating at rate R [7, 19, 18]. A work-conserving switch guarantees that if a cell is pending for output port j at time-slot[2] 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 and minimizes its average cell delay.

The switch used for the comparison is called a *shadow* switch or a *reference* switch, and it receives exactly the same stream of flows 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 reference 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 reference 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 [14, 15, 21].

The *relative queuing delay* considers only the delay resulting from queuing within the PPS switch and neglects factors such as different propagation delays over the PPS and the reference switche, or the different number of stages. It 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.

[1] These are also called *independent demultiplexing algorithms* [13].
[2] A *time-slot* is the time required to transmit a cell at rate R.

1.2 Our Results. Our main contributions are lower bounds on the relative queuing delay and relative delay jitter of the PPS when the switch is not flooded. We employ *leaky-bucket constrained flows* [23] to restrict the arrival of cells to the switch: In every time interval of length τ, the number of cells arriving to the switch and sharing the same input-port or the same output-port is bounded by $\tau R + B$, where B is a fixed *burstiness* factor [5]. Comparison between a PPS and a flooded output-queued switch is vague, since the flooded switch either introduces unbounded queuing delay or drops cells. In Section 5, we show that the lower bounds do not hold for certain classes of non leaky-bucket flows that flood the switch.

A *bufferless PPS* (i.e., without buffers at the input-ports) with fully-distributed demultiplexing algorithm induces 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 demultiplexing algorithm statically partitions the planes 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 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, under leaky-bucket flows with burstiness factor of $\overline{u}^2\frac{N}{K}-\overline{u}$, where $\overline{u}=\min\{u,\frac{1}{2}\frac{R}{r}\}$. In contrast, Iyer et al. [14] show that there is a bufferless PPS with centralized demultiplexing algorithm with zero relative queuing delay, provided that the switch has speedup $S \geq 2$.

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., [14]). This demonstrates that a lower bound of $\Omega(N/S)$ time-slots does not hold when the input-buffers are sufficiently large. 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. This is significant since great effort is currently invested in building switches with a large number of ports (where $N = 512$ or even 1024). Note that large relative queuing delays usually imply that the buffer sizes at the middle-stage switches or at the external ports should be large as well, so that the cells can be queued.

Note that due to space limitations proofs are omitted throughout the paper. For detailed proofs and further discussion, the reader is referred to [3].

1.3 Related Work. The CPA centralized demultiplexing algorithm [14] allows a PPS with speedup $S \geq 2$ to mimic an FCFS output-queued switch with

zero relative queuing delay. This algorithm is impractical for real switches, because it gathers information from all the input-ports in every scheduling decision. A fully-distributed version of this algorithm [15] mimics a FCFS output-queued switch with relative queuing delay of $\lceil N\frac{R}{r} \rceil$ time-slots.

Another family of fully-distributed algorithms, called *fractional traffic dispatch (FTD)* [17], works with switch speedup $S \geq \frac{K}{\lceil K/2 \rceil}$, and their relative queuing delay is at least $2NR/r$ time-slots. An extension to the FTD algorithms is introduced in Section 5, in which there is zero relative queuing delay as long as all the queues in all the planes are not empty.

Arbitrated crossbar switches [22] are prime examples of $u-RT$ demultiplexing algorithms. In these switches, a request is made by the input-port, and the cell is sent once a grant is received back from the arbiter. This implies that global information is used by the arbiter, with a certain delay. Cells are queued in the input-port buffers while waiting for a grant. Input-port buffers should operate at the external line rate R, and therefore must reside on chip. This limits their size and causes large values of u to be impractical.

Earlier research presents lower bounds on the speedup required for an input-queued switch to *exactly* mimic a reference output-queued switch. Chaung et al. [7] show that a combined input-output-queued switch needs speedup $\geq 2 - \frac{1}{N}$ in order to mimic an output-queued switch. This worst-case lower bound proof does not assume any statistical distribution on the arrival of packets to the switch; it uses specific input-flows with burstiness factor N.

2. Formal Model for Parallel Packet Switches

To prove lower bounds on the behavior of the PPS, a formal model of the switch is needed.

We assume that cells arrive to the switch and leave it in discrete *time-slots*. Since a time-slot is the time it takes to transmit a cell at rate R, in each time-slot at most one cell arrives to each input-port, and at most one cell leaves any output-port. The internal lines of the switch operate at lower rate $r < R$; for simplicity, we assume that $\frac{R}{r} = \lceil \frac{R}{r} \rceil$, and denote this value by r'. This lower rate r enforces constraints on the switch [14]: 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.

Recall that 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 to the output-port, provided that no other cell is leaving the same output-port on this time-slot.

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

A *switch configuration* is comprised of 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 (i.e, the set of applicable configurations induces strongly-connected graph).

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

DEFINITION 1 *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 extended for the input-buffered PPS variant: 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. $\mathbb{B}$ denotes the set of the applicable buffer's states, and $\mathbb{B}_i$ includes the applicable states of the buffer residing in input-port i. We refer to the set of states of the i^{th} demultiplexor as $\mathbb{S}_i \times \mathbb{B}_i$. A switch configuration describes also the input-buffers content.

DEFINITION 2 *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.

3. 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 [14] is a centralized demultiplexing algorithm with zero relative queuing delay, assuming the PPS has speedup $S \geq 2$ and the reference output-queued switch follows a global FCFS discipline.[3] Practical demultiplexing algorithms must operate with local, or out-dated, 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 non-negligible queuing delay.

Our lower bounds are obtained using feasible flows that do not flood the switch, obeying the *leaky-bucket constrained flows* model [23]. We require only 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 [5].

DEFINITION 3 *An* (R, B) leaky-bucket traffic *satisfies the following expression for every time-slot* t*, integer* $\tau \geq 1$*, input-port* i*, and output-port* j*:*

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

where $A_i(t_1, t_2)$ *is the number of cells arriving to 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 to 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 [9].

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

LEMMA 4 *Assume output-port* j*'s buffer is empty at time* t*, and that* m *cells destined for the same output-port* j *arrive to 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 to 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 reference switch. Since the reference switch is work-conserving, 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,

[3] i.e., cells should leave the switch in the same order they arrived, regardless of the flow they are in.

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 to 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 to 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 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 serve the incoming cells in a FCFS manner (e.g. 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 reference switch is at least $(cr' - s) - B = c\frac{R}{r} - (s + B)$ time-slots, proving (2). ■

The concentration scenario, described in Lemma 4, does not depend on the scheduling policies of the planes, which may be optimal. It only assumes that cells are not dropped.

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

DEFINITION 5 *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 i^{th} bufferless demultiplexor operating under 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 to a specific input-port in bufferless PPS, its demultiplexor does not change its state.

The relative queuing delay of a PPS with 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.

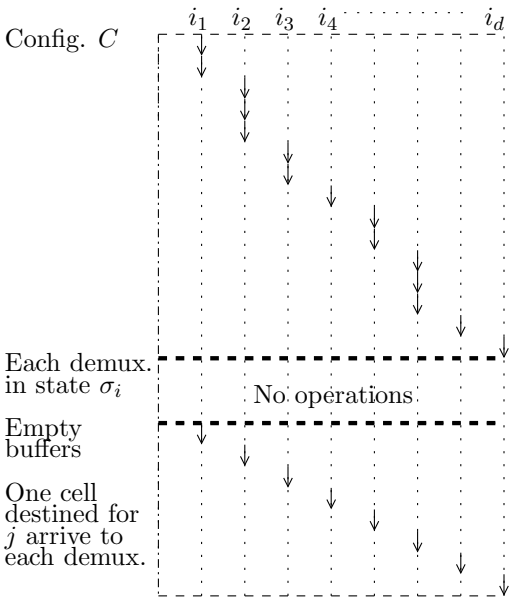

Figure 2. Schematic view of the proof of Theorem 6.

THEOREM 6 *A bufferless PPS, with 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.*

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 recall that there is a traffic that causes the switch to transit between any two applicable configurations. Let A_i be a traffic in which cells arrive to 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 to the switch in every time-slot, therefore this traffic has no bursts.

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 to the switch in the same time-slots as in traffic A_i, until demultiplexor i reaches state σ_i. Then, no cells arrive to 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, to 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 Figure 2).

LB has no bursts, and the last d cells that arrive to the switch under traffic LB arrive during d consecutive time-slots. By substituting in Lemma 4, we get that the relative queuing delay and relative delay jitter are at least $\left(\frac{R}{r} - 1\right) d$ time-slots. ■

Statically partitioning the planes among the different demultiplexors is failure-prone. For example, if demultiplexor sends cells only through $d < K$ planes, a 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 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 6 immediately implies:

COROLLARY 7 *A bufferless PPS, with unpartitioned fully-distributed demultiplexing algorithm, has relative queuing delay and relative delay jitter of $\left(\frac{R}{r} - 1\right) N$ time-slots, under leaky-bucket traffic without bursts.*

Iyer and McKeown [15] present an unpartitioned fully-distributed demultiplexing algorithm, which allows a bufferless PPS with speedup $S \geq 2$ to mimic a FCFS output-queued switch with a relative queuing delay of $\lceil \frac{NK}{S} \rceil = \lceil \frac{R}{r} N \rceil$ time-slots. This result implies that $\Theta\left(\frac{R}{r} N\right)$ is a tight bound on the relative queuing delay of a bufferless PPS with speedup $S \geq 2$ operating under a fully-distributed demultiplexing algorithm.

Even with static partitioning, the PPS input constraint implies that each demultiplexor must send incoming cells through at least r' planes[4]. This implies that 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 8 *A bufferless PPS, with fully-distributed demultiplexing algorithm, has relative queuing delay and relative delay jitter of $\left(\frac{R}{r} - 1\right) \frac{N}{S}$ time-slots, under leaky-bucket traffic without bursts.*

Note that centralized demultiplexing algorithms and fully-distributed demultiplexing algorithms are both extreme considering the amount of global information they use. The first kind has perfect knowledge of the switch status, and the second uses no global information at all. An interesting middle-ground is the following class of demultiplexing algorithms:

DEFINITION 9 *A u* real-time distributed (u-RT) demultiplexing algorithm *is an algorithm that demultiplexes a cell, arriving at time t, according to the*

[4]In this extreme case, failure even in one plane, immediately causes cells dropping

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 i^{th} bufferless demultiplexor operating under u-RT demultiplexing algorithm is $S_i(t) : \mathbb{S}_i \times \mathbb{C}^{t-u+1} \times \{1,\ldots,N\} \rightarrow \mathbb{S}_i$, 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. A state of demultiplexor can change even if no cell arrives to the input-port.

The additional global information reduces the relative queuing delay. For example, when a 1-RT demultiplexing algorithm is operating under $(R,0)$ leaky-bucket traffic, it practically has full information about the switch status, and therefore it can emulate a centralized algorithm. Yet, lack of information about recent events yields non-negligible relative queuing delay, caused by leaky-bucket traffic with a non-zero burst:

THEOREM 10 *A bufferless PPS, with 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 for any real-time distributed demultiplexing algorithm:

COROLLARY 11 *A bufferless PPS, with any real-time 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 with burstiness factor* $\frac{N}{K}-1$.

4. 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 reference switch. Generally, input buffers increase the flexibility of the demultiplexing algorithms, which leads to weaker 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 [14].

THEOREM 12 *There is a u-RT demultiplexing algorithm for an globally FCFS input-buffered PPS, with buffer size* $\geq u$ *and speedup* $S \geq 2$*, and a relative queuing delay of at most u time-slots.*

This reduction gives an algorithm that 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, it can be shown 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 $u(\frac{N}{K}-1)$.

The following lower-bound holds for any fully distributed demultiplexing algorithm, regardless of buffer size:

THEOREM 13 *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.*

5. The Relative Queuing Delay in Congested Periods

In this section we introduce a parameterized fully-distributed demultiplexing algorithm for bufferless PPS that has zero relative queuing delay in *congested periods*. A time period $(t_1, t_2]$ is *congested* if for a certain output-port j, all the queues of cells destined for this output-port in all the planes, are continuously backlogged.

THEOREM 14 *A bufferless PPS has a parameterized fully-distributed demultiplexing algorithm, which introduce no relative queuing delay in congested periods, after a certain warm-up period.*

The algorithm is an extension of the *fractional traffic dispatch (FTD)* demultiplexing algorithm [17]. In this fully-distributed demultiplexing algorithm, each flow (i, j) is segmented into *blocks* of size $h\lceil R/r\rceil$, where $h > 1$ is a parameter of the specific algorithm. The cells in the flow (i, j) are dispatched, so that two cells from the same block are not sent through the same plane.

This algorithm requires speedup $S \geq h$ in order to operate correctly. During an initial *warm-up* period there is still relative queuing delay. This period can be shortened by enlarging h. (See [3] for detailed analysis of this algorithm).

In the following proposition we show that the above proof does not contradict Theorem 8 since the traffic which causes the congestion is not an (R, B) leaky-bucket traffic:

PROPOSITION 15 *Any traffic that causes congestion in a PPS, under any demultiplexing algorithm described in the proof of Theorem 14, is not an* (R, B) *leaky-bucket traffic, for any* B *independent of time and the duration of the congested period.*

Note that the definition of congestion depends on both the incoming traffic to the switch, and the demultiplexing algorithm. Characterizing the non leaky-bucket traffics that can cause congestion under certain demultiplexing algorithms, and does not introduce relative queuing delay, is an open question.

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 on-line algorithms, is appealing because it does not require stochastic characterization of the incoming traffic.

Our results use Leaky-bucket traffic [5, 23] is used to avoid flooding the switch. 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 [1, 4]; our flows satisfy these stronger restrictions as well.

Traffic shaping with low jitter may prefer non-work-conserving switches [10, 25], and therefore it is interesting to compare with such switches. When cells are not dropped within the switch, a non-work-conserving reference 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 [20, 16, 26]; in particular, Mansour and Patt-Shamir [20] present competitive analysis of jitter regulators with bounded internal buffer size. It might be possible to translate our lower bounds on the relative queuing delay to bounds on the size of this internal buffer.

We are currently looking for demultiplexing algorithms that match the lower bounds presented in this paper. Our lower bounds present worst-case traffics also for randomized demultiplexing algorithms, but it would be interesting to study the distribution of the relative queuing delay when randomization is employed.

References

[1] M. Andrews, B. Awerbuch, A. Fernandez, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy Contention-Resolution protocols. *Journal of the ACM*, 48(1):39–69, 2001.

[2] The ATM Forum. *Inverse Multiplexing for ATM (IMA) specification*, March 1999. Version 1.1, AF-PHY-0086.001.

[3] H. Attiya and D. Hay. The inherent queuing delay of parallel packet switches. Technical Report CS-2004-02, Technion - Israel Institute of Technology, 2004.

[4] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial Queueing Theory. *Journal of the ACM*, 48(1):13–38, 2001.

[5] A. Charny. *Providing QoS guarantees in input buffered crossbar switches with speedup*. PhD thesis, Massachusetts Institute Of Technology, September 1998.

[6] F. M. Chiussi, D. A. Khotimsky, and S. Krishnan. Generalized inverse multiplexing of switched atm connections. In *IEEE Globecom*, 1998.

[7] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input output queued switch. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1169–1178, 1999.

[8] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, pages 406–424, 1953.

[9] R. L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[10] K.J. Chen C.S. Wu, J.C. Jiau. Characterizing traffic behavior and providing end-to-end service guarantees within ATM networks. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 336–344, 1997.

[11] J. Duncanson. Inverse multiplexing. *IEEE Communications Magazine*, 32(4):34–41, April 1994.

[12] P. Fredette. The past, present, and future of inverse multiplexing. *IEEE Communications Magazine*, 32(4):42–46, April 1994.

[13] S. Iyer. Analysis of a packet switch with memories running slower than the line rate. Master's thesis, Stanford University, May 2000.

[14] S. Iyer, A. Awadallah, and N. McKeown. Analysis of a packet switch with memories running at slower than the line rate. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 529–537, 2000.

[15] S. Iyer and N. McKeown. Making parallel packet switches practical. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1680–1687, 2001.

[16] S. Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley Publishing Co., 1997.

[17] D. Khotimsky and S. Krishnan. Stability analysis of a parallel packet switch with bufferless input demultiplexors. In *IEE International Conference on Communications (ICC)*, pages 100–106, 2001.

[18] L. Kleinrock. *Queuing Systems, Volume II*. Jhon Wiley&Sons, 1975.

[19] P. Krishna, N. S. Patel, A. Charny, and R.J. Simcoe. On the speedup required for work-conserving crossbar switches. *IEEE Journal on Selected Areas in Communications*, 17(6):1057–1066, June 1999.

[20] Y. Mansour and B. Patt-Shamir. Jitter control in Qos networks. *IEEE/ACM Transactions on Networking*, 9(4):492–502, August 2001.

[21] B. Prabhakar and N. McKowen. On the speedup required for combined input and output queued switching. *Automatica*, 35(12):1909–1920, December 1999.

[22] Y. Tamir and H.C. Chi. Symmetric crossbar arbiters for VLSI communication switches. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):13–27, January 1993.

[23] J. S. Turner. New directions in communications (or which way to the information age?). *IEEE Communications Magazine*, 24(10):8–15, October 1986.

[24] H. Zhang. Providing end-to-end performance guarantees using non-work-conserving disciplines. *Computer Communications: Special Issue on System Support for Multimedia Computing*, 18(10), October 1995.

[25] H. Zhang. Service disciplines for guaranteed performance service in packet-switched networks. *Proceedings of the IEEE*, 83(10):1374–1396, October 1995.

[26] H. Zhang and D. Ferrari. Rate-controlled service disciplines. *Journal of High Speed Networks*, 3(4), 1994.