Analysis of the Parallel Packet Switch Architecture

Sundar Iyer, *Associate Member, IEEE,* and Nick W. McKeown, *Senior Member, IEEE*

***Abstract*—Our work is motivated by the desire to design packet switches with large aggregate capacity and fast line rates. In this paper, we consider building a packet switch from multiple lower speed packet switches operating independently and in parallel. In particular, we consider a (perhaps obvious) parallel packet switch (PPS) architecture in which arriving traffic is demultiplexed over k identical lower speed packet switches, switched to the correct output port, then recombined (multiplexed) before departing from the system. Essentially, the packet switch performs packet-by-packet load balancing, or *inverse multiplexing*, over multiple independent packet switches. Each lower speed packet switch operates at a fraction of the line rate R. For example, each packet switch can operate at rate R/k. It is a goal of our work that all memory buffers in the PPS run slower than the line rate. Ideally, a PPS would share the benefits of an output-queued switch, i.e., the delay of individual packets could be precisely controlled, allowing the provision of guaranteed qualities of service.**

In this paper, we ask the question: Is it possible for a PPS to precisely emulate the behavior of an output-queued packet switch with the same capacity and with the same number of ports? We show that it is theoretically possible for a PPS to emulate a first-come first-served (FCFS) output-queued (OQ) packet switch if each lower speed packet switch operates at a rate of approximately $2R/k$. We further show that it is theoretically possible for a PPS to emulate a wide variety of quality-of-service queueing disciplines if each lower speed packet switch operates at a rate of approximately $3R/k$. It turns out that these results are impractical because of high communication complexity, but a practical high-performance PPS can be designed if we slightly relax our original goal and allow a small fixed-size *coordination buffer* running at the line rate in both the demultiplexer and the multiplexer. We determine the size of this buffer and show that it can eliminate the need for a centralized scheduling algorithm, allowing a full distributed implementation with low computational and communication complexity. Furthermore, we show that if the lower speed packet switch operates at a rate of R/k (i.e., without speedup), the resulting PPS can emulate an FCFS-OQ switch within a delay bound.

***Index Terms*—Clos network, inverse multiplexing, load balancing, output queueing, packet switch.**

I. Introduction

WAVELENGTH division multiplexing (WDM) is making available long-haul fiber-optic links with very high capacity by allowing a single fiber to contain multiple separate channels. Currently, channels operate at OC48c (2.5 Gb/s), OC192c (10 Gb/s), and in some systems OC768c (40 Gb/s). The packets or cells carried on each WDM channel are switched, or routed, by packet switches (e.g., ATM switches, frame relay switches, and IP routers) that process and then switch packets between different channels. It would be desirable to process packets in the optical domain without conversion to electronic form. However, all packet switches need buffering (by definition), and it is not economically feasible today to store packets optically. Thus, packet switches will continue to use electronic buffer memories for some time to come. However, at the data rates anticipated for individual WDM channels, we may not be able to buffer packets as fast as they arrive and depart. As line rates increase beyond OC192 (10 Gb/s) to OC768 (40 Gb/s) and even to OC3072 (160 Gb/s), it becomes difficult, perhaps impossible, to buffer packets as fast as they arrive using conventional memory devices. For example, a packet buffer built using currently available DRAM would require a 16 000-bit-wide data bus.[1]

The purpose of this paper is not to argue that line rates will continue to increase; on the contrary, it could be argued that DWDM will lead to a larger number of logical channels each operating no faster than, say, 10 Gb/s. We simply make the observation that if line rates do increase, then memory bandwidth limitations may make packet buffers and, hence, packet switches difficult or impossible to implement.

It is the overall goal of our work to design a high-capacity packet switch (e.g., multiple terabits/second) that: 1) supports individual line rates in excess of the speeds of available electronic memory and 2) is capable of supporting the same qualities of service as an output-queued (OQ) switch. These two goals cannot be realized alone by a conventional OQ switch; this is because OQ switches require buffer memory that operates at N times the line rate, where N is the number of ports of the switch. This certainly does not meet our goal of memory running *slower* than any individual line rate.

Likewise, we cannot use the other widely used techniques for reducing memory bandwidth, namely, input-queued (IQ) and combined input-and-output queued (CIOQ) switches. In an IQ switch, each memory operates at the same speed as the external line rate. While an improvement over OQ switches, neither of our goals are met: 1) an IQ switch does not meet our requirement to use memories slower than the line rate and 2) IQ switches are unable to provide the same QoS guarantees as an OQ switch. It is known that a variety of qualities of service are possible in a CIOQ switch in which the memory operates at *twice* the line rate [7]. Obviously, this does not meet our goal for memory speed.

Manuscript received November 7, 2001; revised February 4, 2002 and May 24, 2002; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Ajmone Marsan. This work was supported by the National Science Foundation under NGI Contract ANI-9872761, by the Industrial Technology Research Institute, Taiwan, R.O.C., and by the Alfred P. Sloan Foundation.

The authors are with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305-9030 USA (e-mail: sundaes@stanford.edu; nickm@stanford.edu).

Digital Object Identifier 10.1109/TNET.2003.810315

[1]At the time of writing, the random access time (the time to retrieve data at random from any memory location) of a DRAM is approximately 50 ns. Although the access time will be reduced over time, the rate of improvement is much slower than Moore's Law [1]. The random access time should not be confused with the memory I/O time (the time to send retrieved data off chip to the requester). While new memory technologies, such as RAMBUS [6], SDRAMs, and DDRAMs have fast I/O times, the memory core and, hence, the random access time are essentially unchanged.

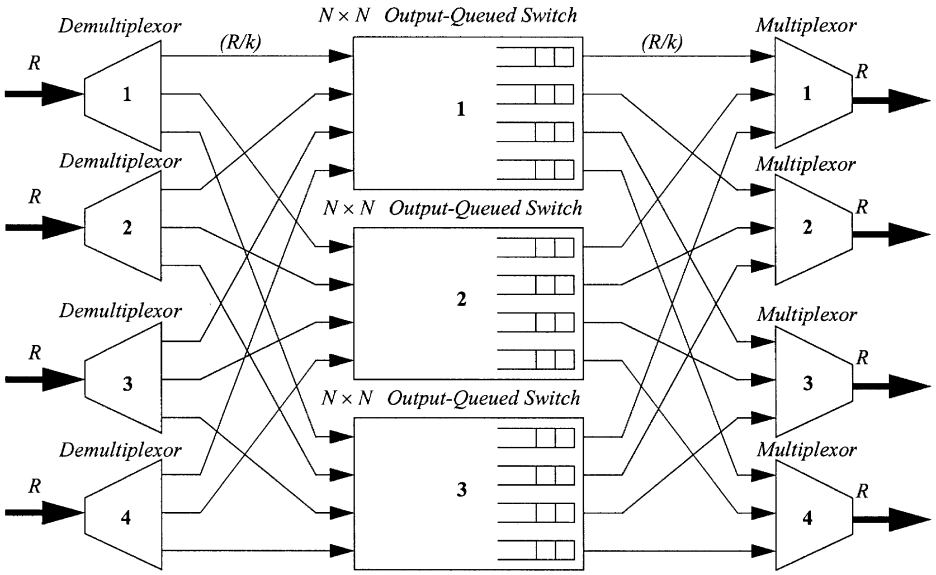


Fig. 1. Architecture of a PPS based on output-queued switches. The architecture resembles a Clos network. The demultiplexers, slower speed packet switches, and multiplexers can be compared to be the three stages of a Clos network.

We would like an architecture that overcomes these limitations, yet is practical.

II. Background

The parallel packet switch (PPS) aims to overcome the memory bandwidth limitation. The PPS is comprised of multiple identical lower speed packet switches operating independently and in parallel. An incoming stream of packets is spread, packet by packet, by a demultiplexer across the slower packet switches, then recombined by a multiplexer at the output. The PPS architecture resembles that of a Clos Network [4], as shown in Fig. 1. The demultiplexer, the center-stage packet switches, and the multiplexer can be compared to the three stages of an unbuffered Clos network.

As seen by an arriving packet, a PPS is a single-stage packet switch; all of the buffering is contained in the slower packet switches, and so our first goal is met because no buffers in a PPS need to run as fast as the external line rate.[2] The demultiplexer selects an internal lower speed packet switch (or "layer") and sends the arriving packet to that layer, where it is queued until its departure time. When the packet's departure time arrives, it is sent to the multiplexer that places the packet on the outgoing line. However, they must make intelligent decisions, and as we shall see, the precise nature of the demultiplexing ("spreading") and multiplexing functions are key to the operation of the PPS.

Although the specific PPS architecture seems novel, *load-balancing* and *inverse-multiplexing* systems [8]–[10] have been around for some time, and the PPS architecture is a simple extension of these ideas. Related work studied inverse ATM multiplexing and how to use sequence numbers to resynchronize cells sent through parallel switches or links [11]–[16]. However, we are not aware of any analytical studies of the PPS architecture prior to this work. As we shall see, there is an interesting and simple analogy between the (buffered) PPS architecture and the (unbuffered) Clos network [4].

We are interested in the question: Can we select the demultiplexing and multiplexing functions so that a PPS can emulate[3] the behavior of an OQ switch? If it is possible for a PPS to emulate an OQ switch, it will be possible to control delay of individual packets and, therefore, provide QoS. In this paper, we will describe and analyze the PPS for unicast traffic only. An existing paper [5] describes how a PPS can emulate an OQ switch for multicast traffic.

The rest of the paper is organized as follows. In Sections III and IV, we introduce some terminology and definitions. In Section V, we find the conditions under which the PPS can emulate a first-come-first-served (FCFS)-OQ switch. In Section VI, we show how a PPS can emulate an OQ switch with different qualities of service. However, our initial algorithms require a large communication complexity that makes them impractical, and so in Section VII we modify the PPS and allow for a small buffer (that must run at the line rate) in the multiplexer and the demultiplexer. We describe a different distributed algorithm which eliminates the communication complexity and appears to be more practical. In Section VIII, we show how the modified PPS can emulate an FCFS-OQ switch within a delay bound without speedup. We briefly describe some implementation issues in Section IX.

III. Definitions

Before proceeding it will be useful to define some terms used throughout this paper:

Cell: A fixed-length packet, though not necessarily equal in length to a 53-byte ATM cell. Although packets arriving to the switch may have variable length, for the purposes of this paper we will assume that they are segmented and processed internally

[2]There will, of course, be small staging buffers in the demultiplexers and multiplexers for rate conversion between an external link operating at rate R and internal links operating at rate R/k. Because these buffers are small (approximately k packets), we will ignore them in the rest of the paper.

[3]We will describe in Section III the exact meaning of the term *emulate*.

as fixed length cells. This is common practice in high-performance routers; variable-length packets are segmented into cells as they arrive, carried across the switch as cells, and reassembled back into packets before they depart.

Time slot: Refers to the time taken to transmit or receive a fixed length cell at a link rate of R.

Internal time slot: This is the time taken to transmit or receive a fixed length cell at a link rate of R/k, where k is the number of center-stage switches in the PPS.

OQ Switch: A switch in which arriving packets are placed immediately in queues at the output, where they contend with other packets destined to the same output. The departure order might be FCFS, in which case we call it an FCFS-OQ switch. Other service disciplines, such as WFQ [19], GPS [20], virtual clock [21], and DRR [22] are widely used to provide QoS guarantees. One characteristic of an OQ switch is that the buffer memory must be able to accept (write) N new cells per time slot where N is the number of ports, and read one cell per cell time. Hence, the memory must operate at $N+1$ times the line rate.

Work conserving: A packet switch is said to be work conserving if an output is busy whenever there is a cell in the system for it. If a packet switch is work conserving, its throughput is maximized, and the average latency of cells is minimized.

In this paper, we will compare the performance of a PPS and an OQ switch. The following definitions help us formally compare the two switches.

Shadow OQ switch: We will assume that there exists an OQ switch, called the shadow OQ switch, with the same number of input and output ports as the PPS. The ports on the shadow OQ switch receive identical input traffic patterns and operate at the same line rate as the PPS.

Mimic: Two different switches are said to mimic [7], [17], [18] each other, if under identical inputs, identical packets depart from each switch at the same time.

An FCFS-OQ switch is work conserving, and so a necessary (but not sufficient) condition for a switch to mimic output queueing is that it be work conserving. A work-conserving switch may reorder packets, while a switch which mimics an FCFS OQ switch cannot. On the other hand, an IQ switch is not in general work conserving because a cell can be held at an input queue even though its output is idle.

Relative queueing delay: A cell's relative queueing delay is the increased queueing delay (if any) that it receives in a switch relative to the delay it receives in the shadow OQ switch. Our definition of relative queueing delay only includes differences attributed to queueing. Differences in fixed delay (e.g., because of differences in propagation delay) are not included in this measure.

Emulate: Two different switches are said to emulate each other if, under identical inputs, they have identical relative queueing delays. Thus, the term *emulate* is identical to *mimic* if we ignore the fixed propagation delays in the switches. We shall use the term *emulate* in the rest of the paper to compare the PPS with an OQ switch.

For example, in a PPS, cells are sent over slower speed internal links of rate sR/k, and so incur a larger (but constant) propagation delay relative to an OQ switch.

Push-In First-Out (PIFO) Queues: A PIFO [7] can be used to represent a class of QoS scheduling disciplines such as WFQ, GPS, and strict priorities.[4] A PIFO queue is defined as follows:

1) arriving cells are placed at (or push-in to) an arbitrary location in the queue;
2) the relative ordering of cells in the queue does not change once cells are in the queue, i.e., cells in the queue cannot switch places;
3) cells may be selected to depart from the queue only from the head of line.

IV. PPS Architecture

In this paper, we focus on the specific type of PPS illustrated in Fig. 1 in which the center-stage switches are OQ. The figure shows a 4×4 PPS, with each port operating at rate R. Each port is connected to all three OQ switches (we refer to the center-stage switches as *layers*). When a cell arrives at an input port, the demultiplexer selects a layer to send the cell to; the demultiplexer makes its choice of layer using a policy that we will describe later. Since the cells from each external input, of line rate R, are spread (demultiplexed) over k links, each input link must run at a speed of at least R/k.

Each layer of the PPS may consist of a single OQ or CIOQ switch with memories operating slower than the rate of the external line. Each of the layers receive cells from the N input ports, then switches each cell to its output port. During times of congestion, cells are stored in the output queues of the center stage, waiting for the line to the multiplexer to become available. When the line is available, the multiplexer selects a cell among the corresponding k output queues in each layer. Since each multiplexer receives cells from k output queues, the queues must operate at a speed of at least R/k to keep the external line busy.

Externally, the switch appears as an $N \times N$ switch with each port operating at rate R. Note that neither the multiplexer nor the demultiplexer contain any memory, and that they are the only components running at rate R.

We can compare the memory-bandwidth requirements of an $N \times N$ PPS with those for an OQ switch with the same aggregate bandwidth. In an OQ switch, the memory bandwidth on each port must be at least $(N+1)R$, and in a PPS at least $(N+1)R/k$, but we can reduce the memory bandwidth further using a CIOQ switch. From [7], we know that an OQ switch can be mimicked precisely by a CIOQ switch operating at a speedup of two. So, we can replace each of the OQ switches in the PPS with a CIOQ switch, without any change in operation. The memory bandwidth in the PPS is reduced to $3R/k$ (one read operation and two write operations per cell time), which is independent of N and may be reduced arbitrarily by increasing the number of layers k.

Choosing the value of k: Our goal in this paper is to design switches in which all the memories run at slower than the line rate. If the center stage switches are CIOQ switches, this means that $3R/k < R \Rightarrow k > 3$. Similarly, for center-stage OQ switches, we require that $(N+1)R/k < R \Rightarrow k > N+1$. This

[4]Note that some QoS scheduling algorithms do not use PIFO queueing, such as weighted round robin and WF^{2}Q [24].

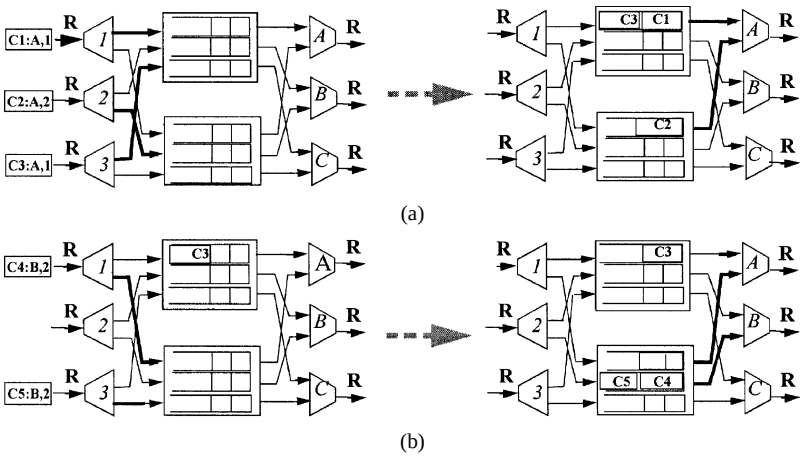


Fig. 2. A 3×3 PPS with an arrival pattern that makes it non-work-conserving. The notation Ci: A, m denotes a cell numbered i, destined to output port A, and sent to layer m. (a) Cells arriving at time slot 1 and being sent to the center stage switches. (b) Cells arriving at time slot 2 and being sent to the center stage switches.

gives a lower bound on k. Further, one can increase the value of k beyond the lower bound, allowing us to use an arbitrarily slow memory device.

As an example, suppose $N = 1024$ ports, $R = 40$ Gb/s, and cells are 64 bytes long. Then a PPS with $k = 10$ center-stage CIOQ switches can be built such that the fastest memories run at a speed no greater than $3R/k = 12$ Gb/s. For a 64-byte cell, this corresponds to an access time of 42.6 ns, which is within the random access time of commercial DRAMs.

A. The Need for Speedup

It is tempting to assume that because each layer is output queued, it is possible for a PPS to emulate an OQ switch. This is actually not the case unless we use speedup. As can be seen from the following counterexample, without speedup a PPS is not work conserving and, hence, cannot emulate an OQ switch.

Theorem 1: A PPS without speedup is not work conserving.

Proof: (By Counterexample): Consider the PPS in Fig. 2 with three ports and two layers ($N = 3$ and $k = 2$). The external lines operate at rate R and the internal lines at rate $R/2$.

Assume that the switch is empty at time $t = 1$, and that three cells arrive, one to each input port, and all destined to output port A. If all the input ports choose the same layer, then the PPS is non-work-conserving. If not, then at least two of these inputs will choose the same layer and the other input will choose a different layer. Without loss of generality, let inputs 1 and 3 both choose layer 1 and send cells $C1$ and $C3$ to layer 1 in the first time slot. This is shown in Fig. 2(a). Also, let input port 2 send cell $C2$ to layer 2. These cells are shown in the output queues of the internal switches and await departure. Now, we create an adversarial traffic pattern. In the second time slot, the adversary picks the input ports which sent cells to the same layer in the first time slot. These two ports are made to receive cells destined to output port B. As shown in the figure, cells $C4$ and $C5$ arrive at input ports 1 and 3 and they both must be sent to layer 2; this is because the internal line rate between the demultiplexer and each layer is only $R/2$, limiting a cell to be sent over this link only once every other time slot. Now the problem becomes apparent: cells $C4$ and $C5$ are in the same layer, and they are the only cells in the system destined for output port B at time slot 2. These two cells cannot be sent back to back in consecutive time slots, because the link between the layer and the multiplexer operates only at rate $R/2$. So, cell $C4$ will be sent, followed by an idle time slot at output port B, and the system is no longer work conserving. And so, trivially, a PPS without speedup cannot emulate an FCFS-OQ switch. □

Definition 1: Concentration: Concentration is a term we will use to describe the situation when a disproportionately large number of cells destined to the same output are concentrated on a small number of the internal layers.

Concentration is undesirable as it leads to unnecessary idling because of the limited line rate between each layer and the multiplexer. One way to alleviate the effect of concentration is to use faster internal links. In general, we will use internal links that operate at a rate $S(R/k)$, where S is the speedup of the internal link.

For example, in our counterexample in Theorem 1, the problem could be eliminated by running the internal links at a rate of R instead of $R/2$ (i.e., a speedup of two). This solves the problem because the external output port can now read the cells back to back from layer two. However, this appears to defeat the purpose of operating the internal layers slower than the external line rate. Fortunately, we will see in Section IV that the speedup required to eliminate the problem of concentration is independent of the arriving traffic, R, N, and is almost independent of k. In particular, we find that with a speedup of two, the PPS is work conserving and can emulate an FCFS-OQ switch.

B. Link Constraints

The operation of a PPS is limited by two constraints. We call these the *input link constraint* and the *output link constraint*, as defined below.

Definition 2: Input Link Constraint : An external input port is constrained to send a cell to a specific layer at most once every $\lceil k/S \rceil$ time slots. This is because the internal input links operate S/k times slower than the external input links. We call this constraint the input link constraint (ILC).

Definition 3: Allowable Input Link Set: The ILC gives rise to the allowable input link set $\mathrm{AIL}(i, n)$, which is the set of layers

to which external input port i can start sending a cell in time slot n. This is the set of layers that external input i has not started sending any cells within the last $\lceil k/S \rceil - 1$ time slots. Note that $|\text{AIL}(i,n)| \leq k, \forall\ (i,n)$.

$\text{AIL}(i,n)$ evolves over time, with at most one new layer being added to, and at most one layer being deleted from, the set in each time slot. If external input i starts sending a cell to layer l at time slot N, then layer l is removed from $\text{AIL}(i,n)$. The layer is added back to the set when it becomes free at time $n+\lceil k/S \rceil$.

Definition 4: Output Link Constraint: In a similar manner to the ILC, a layer is constrained to send a cell to an external output port at most once every $\lceil k/S \rceil$ time slots. This is because the internal output links operate S/k times slower than the external output links. Hence, in every time slot an external output port may not be able to receive cells from certain layers. This constraint is called the output link constraint (OLC).

Definition 5: Departure Time: When a cell arrives, the demultiplexer selects a departure time for the cell. A cell arriving to input i at time slot n and destined to output j is assigned the departure time $\text{DT}(n,i,j)$. The departure time could, for example, be the first time that output j is free (in the shadow OQ switch) and able to send the cell. As we shall see later in Section VI, other definitions are possible.

Definition 6: Available Output Link Set: The OLC gives rise to the available output link set $\text{AOL}(j,\text{DT}(n,i,j))$, which is the set of layers that can send a cell to external output j at time slot $\text{DT}(n,i,j)$ in the future. $\text{AOL}(j,\text{DT}(n,i,j))$ is the set of layers that have not started sending any cells to external output j in the last $\lceil k/S \rceil - 1$ time slots before time slot $\text{DT}(n,i,j)$. Note that, since there are a total of k layers, $|\text{AOL}(j,\text{DT}(n,i,j))| \leq k$, $\forall\ (j,\text{DT}(n,i,j))$.

Like $\text{AIL}(i,n)$, $\text{AOL}(j,\text{DT}(n,i,j))$ can increase or decrease by at most one layer per departure time slot, i.e., if a layer l starts to send a cell to output j at time slot $\text{DT}(n,i,j)$, the layer is deleted from $\text{AOL}(j,\text{DT}(n,i,j))$ and then will be added to the set again when the layer becomes free at time $\text{DT}(n,i,j) + \lceil k/S \rceil$. However, whenever a layer is deleted from the set, the index $\text{DT}(n,i,j)$ is incremented. Because, in a single time slot, up to N cells may arrive at the PPS for the same external output, the value of $\text{DT}(n,i,j)$ may change up to N times per time slot. This is because $\text{AOL}(j,\text{DT}(n,i,j))$ represents the layers available for use at some time $\text{DT}(n,i,j)$ in the future. As each arriving cell is sent to a layer, a link to its external output is reserved for some time in the future. So, effectively, $\text{AOL}(j,\text{DT}(n,i,j))$ indicates the schedule of future departures for output j, and at any instant, $\text{Max}(\text{DT}(n,i,j)) + 1, \forall\ (n,i)$ indicates the first time in the future that output j will be free.

C. Lower Bounds on the Size of the Link Constraint Sets

The following two lemmas will be used later to demonstrate the conditions under which a PPS can emulate an FCFS-OQ switch.

Lemma 1: The size of the available input link set, $|\text{AIL}(i,n)| \geq k - \lceil k/S \rceil + 1$, for all $i, n \geq 0$, where S is the speedup on the internal input links.

Consider external input port i. The only layers to which i cannot send a cell are those which were used in the last $\lceil k/S \rceil - 1$ time slots. (The layer which was used $\lceil k/S \rceil$ time slots ago is now free to be used again). $|\text{AIL}(i,n)|$ is minimized when a cell arrives to the external input port in each of the previous $\lceil k/S \rceil - 1$ time slots, hence, $|\text{AIL}(i,n)| \geq k - (\lceil k/S \rceil - 1) = k - \lceil k/S \rceil + 1$. □

Lemma 2: The size of the available output link set $|\text{AOL}(j,\text{DT}(n,i,j))| \geq k - \lceil k/S \rceil + 1$, for all $i, j, n \geq 0$.

Proof: The proof is similar to Lemma 1. We consider an external output port which reads cells from the internal switches instead of an external input port which writes cells to the internal switches. □

V. Emulating an FCFS-OQ Switch

In this section, we shall explore how a PPS can emulate an FCFS-OQ switch. Note that in this section, in lieu of the FCFS policy, the departure time of a cell arriving at input i and destined to output j at time n, $\text{DT}(n,i,j)$, is simply the first time that output j is free (in the shadow FCFS-OQ switch) and able to send a cell.

A. Conditions for a PPS to Emulate an FCFS-OQ Switch

Theorem 2: (Sufficiency): If a PPS guarantees that each arriving cell is allocated to a layer l, such that $l \in \text{AIL}(i,n)$ and $l \in \text{AOL}(j,\text{DT}(n,i,j))$, (i.e., if it meets both the ILC and the OLC) then the switch is work conserving.

Proof: Consider a cell C that arrives to external input port i at time slot n destined for output port j. The demultiplexer chooses a layer l that meets both the ILC and the OLC; i.e., $l \in \{\text{AIL}(i,n) \cap \text{AOL}(j,\text{DT}(n,i,j))\}$, where $\text{DT}(n,i,j)$ is the index of $\text{AOL}(\cdot)$ and represents the first time that external output j is free in the shadow FCFS-OQ switch in the future, at time slot n. Since $l \in \text{AIL}(i,n)$, the ILC is met, and cell C can be immediately written to layer l in the PPS. Since the center-stage switches are OQ switches, the cell C is immediately queued in the output queues of the center-stage switch l, where it awaits its turn to depart. Since the departure time of the cell $\text{DT}(n,i,j)$ has already been picked when it arrived at time n, C is removed from its queue at time $\text{DT}(n,i,j)$ and sent to external output port j. The reason that C can depart at $\text{DT}(n,i,j)$ immediately is because the link from multiplexer j to layer l is available at time $\text{DT}(n,i,j)$, as layer l was chosen such that $l \in \text{AOL}(j,\text{DT}(n,i,j))$. Thus, if for cell C the chosen layer l meets both the ILC and OLC, then the cell leaves the PPS at time $\text{DT}(n,i,j)$. By definition, each cell can be made to leave the PPS at the first time that output j would be idle in the shadow FCFS-OQ switch, before cell C arrived. Thus, output j is continuously kept busy if there are cells destined for it, similar to that of the shadow OQ switch. Obviously, since the shadow OQ switch is work conserving, the PPS is work conserving. □

Theorem 3: (Sufficiency): A speedup of two is sufficient for a PPS to meet both the input and output link constraints for every cell.

For the ILC and OLC to be met, it suffices to show that there will always exist a layer l such that $l \in \{\text{AIL}(i,n) \cap \text{AOL}(j,\text{DT}(n,i,j))\}$, i.e., that $\text{AIL}(i,n) \cap (\text{AOL}(j,\text{DT}(n,i,j))) \neq \emptyset$, which must be satisfied if

$|\text{AIL}(i,n)| + |\text{AOL}(j,\text{DT}(n,i,j))| > k$. From Lemma 1 and Lemma 2, we know that $|\text{AIL}(i,n)|+|\text{AOL}(j,\text{DT}(n,i,j))| > k$ if $S \geq 2$. □

Corollary 1: A PPS can be work conserving if $S \geq 2$.

Having shown that a PPS can be work conserving, we now show that with the same speedup, the PPS can emulate an FCFS-OQ switch.

Theorem 4: (Sufficiency): A PPS can emulate an FCFS-OQ switch with a speedup of $S \geq 2$.[5]

Proof: Consider a PPS with a speedup of $S \geq 2$ which, for each arriving cell, selects a layer that meets both the ILC and the OLC. A cell destined to output j and arriving at time slot n is scheduled to depart at time slot $\text{DT}(n,i,j)$, which is the index of $\text{AOL}(j,\text{DT}(n,i,j))$. By definition, $\text{DT}(n,i,j)$ is the first time in the future that output j is idle in the shadow FCFS-OQ switch. Since the center-stage switches are OQ switches, the cell is queued in the output queues of the center-stage switches and encounters zero relative delay. After subtracting for the propagation delays of sending the cell over lower speed links of rate $2R/k$, $\text{DT}(n,i,j)$ is equal to the time that the cell would depart in an FCFS-OQ switch. Hence, a PPS can emulate an FCFS-OQ switch. □

It is interesting to compare the above proof with the requirements for a three-stage symmetrical Clos network to be *strictly nonblocking* [23], [25]. On the face of it, these two properties are quite different. A PPS is a buffered packet switch, whereas a Clos network is an unbuffered fabric, but because each theorem relies on links to and from the central stage being free at specific times, the method of proof is identical and relies on the pigeonhole principle. A detailed description of the PPS algorithm suggested by Theorem 4, called the centralized packet scheduling algorithm (CPA), appears in [2, Appendix A].

VI. Providing QoS Guarantees

We now extend our results to find the speedup requirement for a PPS to provide QoS guarantees. To do this, we find the speedup required for a PPS to implement any PIFO scheduling discipline.

Theorem 5: (Sufficiency): A PPS can emulate any OQ switch with a PIFO queueing discipline with a speedup of $S \geq 3$.[6]

Proof: As defined in Section III, a PIFO queueing policy can insert a cell anywhere in the queue but it cannot change the relative ordering of cells once they are in the queue. Consider a cell C that arrives to external input port i at time slot n destined to output port j. The demultiplexer determines the time that each arriving cell must depart, $\text{DT}(n,i,j)$, to meet its delay guarantee. The decision made by the demultiplexer at input i amounts to selecting a layer so that the cell may depart on time. Notice that this is very similar to Section V in which cells departed in FCFS order, requiring only that a cell depart the first time that its output is free after the cell arrives. The difference here is that $\text{DT}(n,i,j)$ may be selected to be ahead of cells already scheduled to depart from output j. So, the demultiplexer's choice of sending to layer l an arriving cell C must now meet the following three constraints.

1) The link connecting the demultiplexer at input i to layer l must be free at time slot n. Hence, $l \in \{\text{AIL}(i,n)\}$.
2) The link connecting layer l to output j must be free at $\text{DT}(n,i,j)$, i.e., $l \in \{\text{AOL}(j,\text{DT}(n,i,j))\}$.
3) All the other cells destined to output j after C must also find a link available. In other words, if the demultiplexer picks layer l for cell C, it needs to ensure that no other cell requires the link from l to output j within the next $(\lceil k/S \rceil - 1)$ time slots. The cells that are queued in the PPS for output port j (and have a departure time between $(\text{DT}(n,i,j), \text{DT}(n,i,j)+\lceil k/S \rceil - 1)$), may have already been sent to specific layers (since they could have arrived earlier than time t). It is, therefore, necessary that the layer l be distinct from the layers that the next $\lceil k/S \rceil - 1$ cells use to reach the same output. We can write this constraint as $l \in \{\text{AOL}(j,\text{DT}(n,i,j) + \lceil k/S \rceil - 1)\}$.

The following natural questions arise.

1) *What if some of the cells which depart after cell C have not yet arrived?* This is possible, since cell C may have been pushed in toward the tail of the PIFO queue. In such a case, the cell C has more choice in choosing layers and the constraint set $\text{AOL}(j,\text{DT}(n,i,j) + \lceil k/S \rceil - 1)$ will allow more layers.[7] Note that cell C need not bother about the cells which have not as yet arrived at the PPS, because the future arrivals, which can potentially conflict with cell C, will take into account the layer l to which cell C was sent to. The CPA algorithm will send these future arrivals to a layer distinct from l.

2) *Are these constraints sufficient?* The definitions of the OLC and AOL mandate that when a multiplexer reads the cells in a given order from the layers, the layers should always be available. When a cell C is inserted in a PIFO queue, the only affect it has is that it can conflict with the $\lceil k/S \rceil - 1$ cells which are scheduled to leave before and after it in the PIFO queue. For these $2(\lceil k/S \rceil - 1)$ cells, the arriving cell C can only increase the time between when these cells depart. Hence, these $2(\lceil k/S \rceil - 1)$ cells will not conflict with each other, even after insertion of cell C. Also, if conditions 1 and 2 are satisfied, then these $2(\lceil k/S \rceil - 1)$ cells will also not conflict with cell C. Note that cell C does not affect the order of departures of any other cells in the PIFO queue. Hence, if the PIFO queue satisfied the OLC constraint before the insertion of cell C, then it will continue to satisfy the OLC constraint after it is inserted.

Thus, layer l must satisfy

$$l \in \Bigg\{ \text{AIL}(i,n) \cap \text{AOL}(j,\text{DT}(n,i,j)) \cap \text{AOL}\left(j, \text{DT}(n,i,j) + \left\lceil \frac{k}{S} \right\rceil - 1\right) \Bigg\}.$$

For a layer l to exist, we require

$$\text{AIL}(i,n) \cap \text{AOL}(j,\text{DT}(n,i,j)) \cap \text{AOL}\left(j, \text{DT}(n,i,j) + \left\lceil \frac{k}{S} \right\rceil - 1\right) \neq \varnothing$$

[5]A tighter bound $S \geq k/\lceil k/2 \rceil$ can easily be derived, which is of theoretical interest for small k.

[6]Again, a tighter bound $S \geq k/\lceil k/3 \rceil$ can easily be derived, which is of theoretical interest for small k.

[7]FCFS is a special limiting case of PIFO. Newly arriving cells are pushed in at the tail of an output queue and there are no cells scheduled to depart after a newly arriving cell. Hence, $\text{AOL}(j,\text{DT}(n,i,j)+\lceil k/S \rceil - 1)$, defined at time t, will include all the k layers and so the constraint disappears, leaving us with only two of the three conditions above, as for FCFS-OQ in Section V.

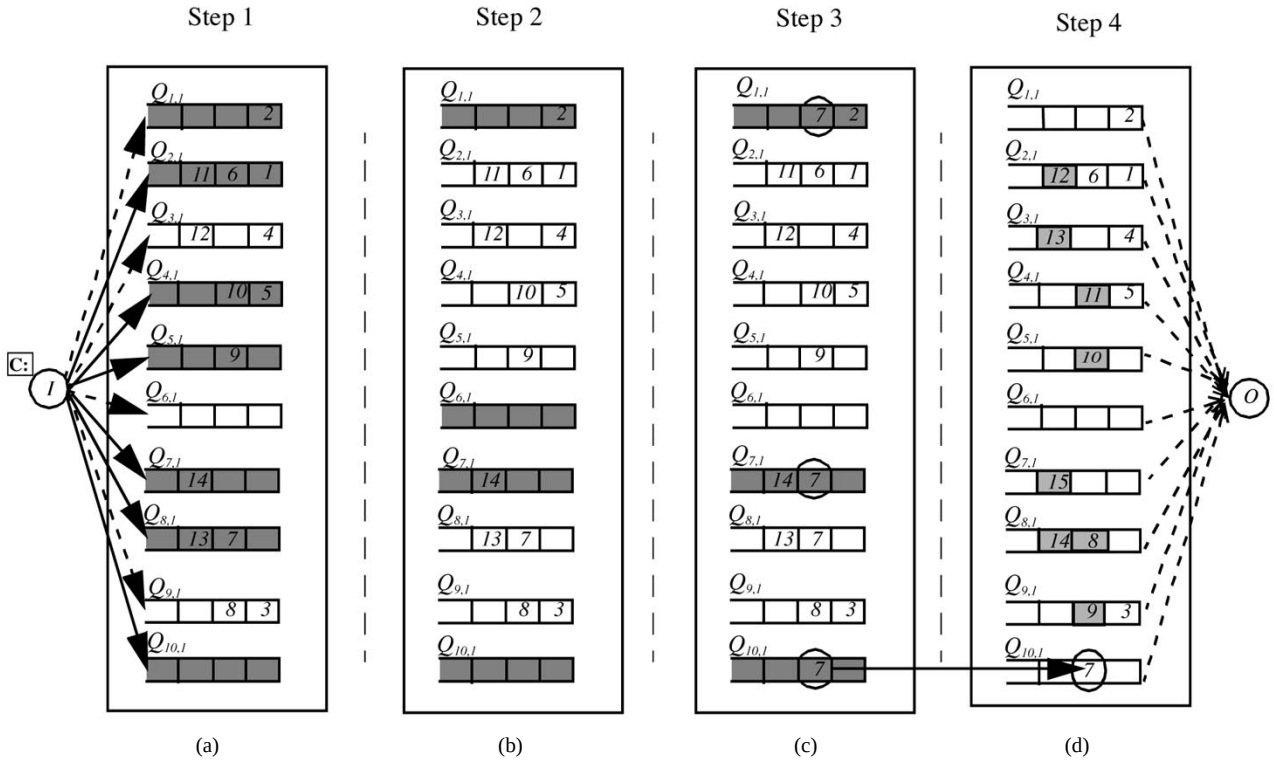


Fig. 3. Insertion of cells in a PIFO order in a PPS with ten layers. $Q_{k,1}$ refers to output queue number one in the internal switch k. The shaded layers describe the sets specified for each figure. (a) AIL and the existing PIFO order. The AIL constrains the use of layers $\{1, 2, 4, 5, 7, 8, 10\}$. (b) Intersection of the two AOL sets. The cell is to be inserted before cell number 7. The two AOLs constrain the use of layers $\{1, 6, 7, 10\}$ (c) Candidate layers for insertion of cell. The intersection constrains the use of layers $\{1, 7, 10\}$. (d) New PIFO order after insertion. Layer 10 is chosen. The cell number 7 is inserted.

which is satisfied when

$$|\mathrm{AIL}(i,n)| + |\mathrm{AOL}(j, \mathrm{DT}(n,i,j))| + \left|\mathrm{AOL}\left(j, \mathrm{DT}(n,i,j) + \left\lceil \frac{k}{S} \right\rceil - 1\right)\right| > 2k.$$

From Lemmas (1) and (2), we know that

$$|\mathrm{AIL}(i,n)| + |\mathrm{AOL}(j, \mathrm{DT}(n,i,j))| + \left|\mathrm{AOL}\left(j, \mathrm{DT}(n,i,j) + \left\lceil \frac{k}{S} \right\rceil - 1\right)\right| > 2k$$

if $S \geq 3$. □

Fig. 3 shows an example of a PPS with $k = 10$ layers and $S = 3$. A new cell C arrives at time t, destined to output 1, and has to be inserted in the priority queue for output 1 which is maintained in a PIFO manner. Assume that the AIL at time t constrains the use of layers $\{1, 2, 4, 5, 7, 8, 10\}$. These layers are shown shaded in Fig. 3(a). It is decided that cell C must be inserted between $C6$ and $C7$. That means that cell C cannot use any layers to which the previous $\lceil k/S \rceil - 1 = 3$ cells before $C7$ (i.e., $C4$, $C5$, and $C6$) were sent to. Similarly, cell C cannot use any layers to which the three cells after $C7$ including $C7$, (i.e., $C7$, $C8$, and $C9$) were sent to. The above two constraints are derived from the AOL sets for output 1. They require that only layers in $\{1, 5, 6, 7, 8, 9, 10\}$ be used, and only layers in $\{1, 2, 3, 4, 6, 7, 10\}$ be used, respectively. Fig. 3(b) shows the intersection of the two AOL sets for this insertion. Cell C is constrained by the AOL to use layers $\{1, 6, 7, 10\}$, which satisfies both the above AOL sets. Finally, a layer is chosen such that the AIL constraint is also satisfied. Fig. 3(c) shows the candidate layers for insertion i.e., layers 1, 7, and 10. Cell C is then inserted in layer 10.

VII. Distributed Approach

A. Limitations of Centralized Approach

Unfortunately, the centralized approach described up until now is useful only in theory. It suffers from two main problems.

1) **Communication complexity**: The centralized approach requires each input to contact a centralized scheduler every arbitration cycle. With N ports, N requests must be communicated to and processed by the arbiter each cycle. This requires a high-speed control path running at the line rate between every input and the central scheduler. Furthermore, the centralized approach requires that the departure order (i.e., the order in which packets are sent from each layer to a multiplexer) be conveyed to each multiplexer and stored.
2) **Speedup**: The centralized approach requires a speedup of two (for an FCFS PPS) in the center-stage switches. The PPS, therefore, overprovisions the required capacity by a factor of two and the links are on average only 50% utilized. This gets worse for a PPS which supports qualities of service, where a speedup of three implies that the links on average are only 33% utilized.

In addition to the difficulty of implementation, the centralized approach does not distribute traffic equally among the center-

stage switches, making it possible for buffers in a center-stage switch to overflow even though buffers in other switches are not full. This leads to inefficient memory usage.[8]

Another problem with the centralized approach is that it requires each multiplexer to explicitly read, or fetch, each packet from the correct layer in the correct sequence. This feedback mechanism makes it impossible to construct each layer from a preexisting unaltered switch or router.

Thus, a centralized approach leads to large communication complexity, high speedup requirement, inefficient utilization of buffer memory, and special-purpose hardware for each layer. In this section, we overcome these problems via the introduction of small memories (presumably on-chip) in the multiplexers and demultiplexers and a distributed algorithm which:

1) enables the demultiplexers and multiplexers to operate independently, eliminating the communication complexity;
2) removes the speedup requirement for the internal layers;
3) allows the buffers in the center-stage switches to be utilized equally;
4) allows a feedforward data path in which each layer may be constructed from preexisting standard OQ switches.

B. Use of a Distributed Algorithm

The goals outlined in Section VII-A naturally lead to the following modifications.

1) **Distributed decisions**: A demultiplexer decides which center-stage switch to send a cell to, based only on the knowledge of cells that have arrived at its input. The demultiplexers do not know the AOL$(\cdot)$ sets, and so have no knowledge of the distribution of cells in the center-stage switches for a given output. Hence, a demultiplexer cannot choose a center-stage switch such that the load is globally distributed over the given output. However, it is possible to distribute the cells which arrive at the demultiplexer for every output equally among all center-stage switches. Given that we also wish to spread traffic uniformly across the center-stage switches, each demultiplexer will maintain a separate round-robin pointer for each output, and dispatch cells destined for each output to center-stage switches in a round-robin manner.
2) **Small coordination buffers operating at the line rate**: If the demultiplexers operate independently and implement a round robin to select a center stage, they may violate the input link constraint. The input link constraint can be met by the addition of a coordination buffer in the demultiplexer which can buffer the cells temporarily before sending them to the center-stage switches. Similarly, it is possible for multiple independent demultiplexers to choose the same center-stage switch for cells destined to the same output. This causes concentration and cells can become missequenced. The order of packets can be restored by the addition of a coordination buffer in each multiplexer to resequence the cells before transmitting them on the external line.

[8]It is possible to create a traffic pattern that does not utilize up to 50% of the buffer memory for a given output port.

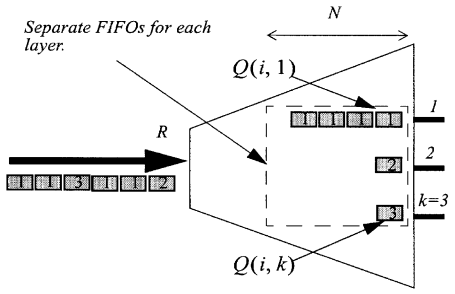


Fig. 4. Demultiplexer, showing k FIFOs, one for each layer, with each FIFO of length d cells. The example PPS has $k = 3$ layers.

We will see that the coordination buffers are small and are the same size for both the multiplexer and demultiplexer. More importantly, they help to eliminate the need for speedup.

C. Consequences of Using a Distributed Algorithm

The coordination buffer operates at the line rate R and so compromises our original goal of having no memories running at the line rate. However, we will show that the buffer size is proportional to the product of the number of ports N and the number of layers k. Depending on these values, it may be small enough to be placed on chip, and so may be acceptable.

Since there is concentration in the PPS and the order of cells has to be restored, we will have to give up the initial goal of emulating an OQ switch. However, we will show that the PPS can emulate an FCFS-OQ switch within a small relative queueing delay bound.

D. Modifications Made to the PPS

Addition of coordination buffers: Fig. 4 shows how the coordination buffers are arranged in each demultiplexer as multiple equal-size FIFOs, one per layer. FIFO $Q(i,l)$ holds cells at demultiplexer i destined for layer l. When a cell arrives, the demultiplexer makes a local decision (described below) to choose which layer the cell will be sent to. If the cell is to be sent to layer l, the cell is queued first in $Q(i,l)$ until the link becomes free. When the link from input i to layer l is free, the head of line cell (if any) is removed from $Q(i,l)$ and sent to layer l.

The buffers in each multiplexer are arranged the same way, and so FIFO $Q'(j,l)$ holds cells at multiplexer j from layer l. We will refer to the maximum length of a FIFO ($Q(i,l)$ or $Q'(j,l)$) as the FIFO length.[9] Note that if each FIFO is of length d, then the coordination buffer can hold a total of kd cells.

Modified PPS dispatch algorithm: The modified PPS algorithm proceeds in three distinct parts.

Step 1) **Split every flow in a round-robin manner in the demultiplexer**: Demultiplexer i maintains N separate round-robin pointers $P_1, \ldots, P_N$; one for each output. The pointers contain a value in the range $\{1, \ldots, k\}$. If pointer $P_j = l$, it indicates that the next arriving cell destined to output j will be sent to layer l. Before being sent, the cell is written temporarily into the coordination FIFO $Q(i,l)$ where it

[9]It will be convenient for the FIFO length to include any cells in transmission.

waits until its turn to be delivered to layer l. When the link from demultiplexer i to layer l is free, the head-of-line cell (if any) of $Q(i,l)$ is sent.

Step 2) **Schedule cells for departure in the center-stage switches**: When scheduling cells in the center stage, our goal is to deliver cells to the output link at their corresponding departure time in the shadow OQ switch (except for a small relative delay). Step 1) introduced a complication that we must deal with: cells reaching the center stage have already encountered a variable queueing delay in the demultiplexer while they waited for the link to be free. This variable delay complicates our ability to ensure that cells depart at the correct time and in the correct order. Shortly, we will see that although variable, this queueing delay is bounded, and so we can eliminate the variability by deliberately delaying all cells as if they had waited for the maximum time in the demultiplexer and, hence, equalize the delays. Though strictly not required, we do this at the input of the center-stage switch. Each cell records how long it was queued in the demultiplexer, and then the central stage delays it further until it equals the maximum. We refer to this step as *delay equalization*. We will see later that delay equalization helps us simplify the proofs for the delay bounds in Section VIII.

After the delay equalization, cells are sent to the output queues of the center-stage switches and are scheduled to depart in the usual way, based on the arrival time of the cell to the demultiplexer. When the cell reaches the head of the output queues of the center-stage switch, it is sent to the output multiplexer when the link is next free.

Step 3) **Reordering the cells in the multiplexer**: The coordination buffer in the multiplexer stores cells where they are resequenced and then transmitted in the correct order.

It is interesting to compare this technique with the load-balanced switch proposed by Chang *et al.* in [28]. In their scheme, load balancing is performed by maintaining a single round-robin list at the inputs (i.e., demultiplexers) for a two-stage switch. The authors show that this leads to guaranteed throughput and low average delays, although packets can be missequenced. In [29], the authors extend their earlier work by using the same technique proposed here: send packets from each input to each output in a round-robin manner. As we shall see, this technique helps us bound the missequencing in the PPS and also gives a delay guarantee for each packet.

VIII. Emulating an FCFS-OQ Switch With a Distributed Algorithm

In what follows, we shall use T to denote time in units of time slots. We shall also use t to denote time, and use it only when necessary. Recall that if the external line rate is R and cells are of fixed size P, then each cell takes P/R units of time to arrive, and $t = TP/R$.

Lemma 3: The number of cells $D(i,l,T)$ that demultiplexer i queues to FIFO $Q(i,l)$ in T time slots is bounded by

$$D(i,l,T) \leq T, \qquad \text{if } T \leq N$$
$$D(i,l,T) < \frac{T}{k} + N, \qquad \text{if } T > N.$$

Proof: Since the demultiplexer dispatches cells in a round-robin manner for every output, for every k cells received by a demultiplexer for a specific output, exactly one cell is sent to each layer. We can write $S(i,T) = \sum_{j=1}^{N} \bar{S}(i,j,T)$, where $\bar{S}(i,j,T)$ is the sum of the number of cells sent by the demultiplexer i to output j in any time interval of T time slots, and $S(i,T)$ is the sum of the number of cells sent by the demultiplexer to all outputs in that time interval T. Let $T > N$. Then we have

$$\begin{aligned} D(i,l,T) &\leq \sum_{j=1}^{N} \left\lceil \frac{\bar{S}(i,j,T)}{k} \right\rceil \\ &\leq \left\lceil \sum_{j=1}^{N} \frac{\bar{S}(i,j,T)}{k} \right\rceil + N - 1 \\ &= \left\lceil \frac{S(i,T)}{k} \right\rceil + N - 1 \leq \left\lceil \frac{T}{k} \right\rceil + N - 1 \\ &< \frac{T}{k} + N \end{aligned}$$

since $S(i,T)$ is bounded by T. The proof for $T \leq N$ is obvious. □

We are now ready to determine the size of the coordination buffer in the demultiplexer.

Theorem 6: (Sufficiency): A PPS with independent demultiplexers and no speedup can send cells from each input to each output in a round-robin order with a coordination buffer at the demultiplexer of size Nk cells.

Proof: A cell of size P corresponds to P/R units of time, allowing us to rewrite Lemma 3 as $D(i,l,t) \leq Rt/Pk + N$ (where t is in units of time). Thus the number of cells written into each demultiplexer FIFO is bounded by $Rt/Pk + N$ cells over all time intervals of length t. This can be represented as a leaky bucket source with an average rate $\rho = R/Pk$ cells per unit time and a bucket size $\sigma = N$ cells for each FIFO. Each FIFO is serviced deterministically at rate $\mu = R/Pk$ cells per unit time. Hence, by the definition of a leaky bucket source [26], a FIFO buffer of length N will not overflow. □

It now remains for us to determine the size of the coordination buffers in the multiplexer. This proceeds in an identical fashion.

Lemma 4: The number of cells $D'(j,l,T)$ that multiplexer j delivers to the external line from FIFO $Q'(j,l)$ [10] in time interval T time slots, is bounded by

$$D'(j,l,T) \leq T, \qquad \text{if } T \leq N,$$
$$D'(j,l,T) < \frac{T}{k} + N, \qquad \text{if } T > N.$$

Proof: Cells destined to multiplexer j from a demultiplexer i are arranged in a round-robin manner, which means that for every k cells received by a multiplexer from a specific

[10] FIFO $Q'(j,l)$ holds cells at multiplexer j arriving from layer l.

input, exactly one cell is read from each layer. We write , $S'(j,T) = \sum_{i=1}^{N} \bar{S}'(i,j,T)$, where $\bar{S}'(i,j,T)$ is the sum of the number of cells from demultiplexer i which were delivered to the external line by multiplexer j in time interval T, and $S'(i,T)$ is the sum of the number of cells from all the demultiplexers that were delivered to the external line by the multiplexer in time interval T. Let $T > N$. Then we have

$$\begin{aligned} D'(i,l,T) &\leq \sum_{i=1}^{N} \left\lceil \frac{\bar{S}'(i,j,T)}{k} \right\rceil \\ &\leq \left\lceil \sum_{i=1}^{N} \frac{\bar{S}'(i,j,T)}{k} \right\rceil + N - 1 \\ &= \left\lceil \frac{S'(i,T)}{k} \right\rceil + N - 1 \leq \left\lceil \frac{T}{k} \right\rceil + N - 1 \\ &< \frac{T}{k} + N \end{aligned}$$

because $S'(i,T)$ is bounded by T. The proof for $T \leq N$ is obvious. □

Finally, we can determine the size of the coordination buffers at the multiplexer.

Theorem 7: (Sufficiency): A PPS with independent multiplexers and no speedup can receive cells for each output in a round-robin order with a coordination buffer of size Nk cells.

Proof: The proof is almost identical to Theorem 6. From Lemma 4, we can bound the rate at which cells in a multiplexer FIFO need to be delivered to the external line by $Rt/Pk + N$ cells over all time intervals of length t. Cells are sent from each layer to the multiplexer FIFO at fixed rate $\mu = R/Pk$ cells per unit time. We can see as a result of the *delay equalization* step in Section VII-D that the demultiplexer and multiplexer systems are exactly symmetrical. Hence, if each FIFO is of length N cells, the FIFO will not overflow. □

Now that we know the size of the buffers at the input demultiplexer and the output multiplexer, both of which are serviced at a deterministic rate, we can bound the relative queueing delay with respect to an FCFS-OQ switch.

Theorem 8: (Sufficiency): A PPS with independent demultiplexers and multiplexers and no speedup, with each multiplexer and demultiplexer containing a coordination buffer of size Nk cells, can emulate an FCFS-OQ switch with a relative queueing delay bound of $2N$ internal time slots.

Proof: We consider the path of a cell in the PPS where the cell may potentially face a queueing delay. These are as follows.

1) The cell may be queued at the FIFO of the demultiplexer before it is sent to its center-stage switch. From Theorem 6, we know that this delay is bounded by N internal time slots.
2) The cell first undergoes delay equalization in the center-stage switches and is sent to the output queues of the center-stage switches. It then awaits service in the output queue of a center-stage switch.
3) The cell may then face a variable delay when it is read from the center-stage switches. From Theorem 7, this is bounded by N internal time slots.

Thus, the additional queueing delay, i.e., the relative queueing delay faced by a cell in the PPS, is no more than $N + N = 2N$ internal time slots. □

IX. Implementation Issues

Given that our main goal is to find ways to make an FCFS PPS (more) practical, we now reexamine its complexity in light of the techniques described.

1) Demultiplexer
   a) Each demultiplexer maintains a buffer of size Nk cells running at the line rate R, arranged as k FIFOs. Given our original goal of having no buffers run at the line rate, it is worth determining how large the buffers need to be and whether they can be placed on chip. For example, if $N = 1024$ ports, cells are 64 bytes long, $k = 10$, and the center-stage switches are CIOQ switches, then the coordination buffer is about 5 Mb per multiplexer and demultiplexer. This can be (just) placed on chip using today's SRAM technology, and so can be made both fast and wide. However, for much larger N, k, or C, this approach may not be practicable.
   b) The demultiplexer must add a tag to each cell indicating the arrival time of the cell to the demultiplexer. Apart from that, no sequence numbers need to be maintained at the inputs or added to cells.
2) Center-stage OQ switches
   The input delay D_i (the number of internal time slots for which a cell had to wait in the demultiplexer's buffer) can be calculated by the center-stage switch using the arrival timestamp. If a cell arrives to a layer at internal time slot t, it is first delayed until internal time slot $\hat{t} = t + N - D_i$, where $1 \leq D_i \leq N$, to compensate for its variable delay in the demultiplexer. After the cell has been delayed, it can be placed directly into the center-stage switch's output queue.
3) Multiplexers
   a) Each multiplexer maintains a coordination buffer of size Nk running at the line rate R.
   b) The multiplexer reorders cells based upon the arrival timestamp. Note that if FCFS order only needs to be maintained between an input and an output, then the timestamps can be eliminated. A layer simply tags a cell with the input port number on which it arrived. This would then be a generalization of the methods described in [27].
   c) We note that if a cell is dropped by a center-stage switch, then the multiplexers cannot detect the lost cell in the absence of sequence numbers. This would cause the multiplexers to resequence cells incorrectly. A solution to this is to mandate the center-stage switches to make the multiplexers aware of dropped cells by transmitting the headers of all dropped cells.

X. Conclusion

While it is difficult to predict the growth of the Internet over the coming years, it seems certain that packet switches will be required with: 1) increased switching capacity; 2) support for higher line rates; and 3) support for differentiated qualities of service. All three of these requirements present challenges of

their own. For example, higher capacity switches may require new architectures, higher line rates may grow to exceed the capabilities of commercially available memories, making it impractical to buffer packets as they arrive, and the need for differentiated qualities of service may require performance comparable to OQ switches.

We consider here a mechanism that attempts to satisfy these goals: a PPS which achieves high capacity by placing multiple packet switches in parallel, rather than in series as is common in multistage switch designs. Hence, each packet that passes through the system encounters only a single stage of buffering; furthermore, and of greatest interest, the packet buffer memories in the center-stage switches operate slower than the line rate.

The main result of this paper is that it is possible to build in a practical way a PPS that can emulate an FCFS-OQ packet switch regardless of the nature of the arriving traffic. In theory, a PPS could emulate an OQ switch which supports guaranteed qualities of service, although the implementation of such a PPS does not yet seem practical.

In summary, we think of this work as a step toward building high-capacity switches in which memory bandwidth is not the bottleneck.

REFERENCES

[1] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A performance comparison of contemporary DRAM architectures," in *Proc. 26th Int. Symp. Computer Architecture (ISCA'99)*, Atlanta, GA, May 1999, pp. 222–233.

[2] S. Iyer, A. Awadallah, and N. McKeown, "Analysis of a packet switch with memories running slower than the line rate," in *Proc. IEEE INFOCOM 2000*, pp. 529–537.

[3] S. Iyer and N. McKeown, "Making parallel packet switches practical," in *Proc. IEEE INFOCOM 2001*, vol. 3, pp. 1680–1687.

[4] C. Clos, "A study of nonblocking switching networks," *Bell Syst. Tech. J.*, vol. 32, 1953.

[5] S. Iyer and N. McKeown, "On the speedup required for a multicast parallel packet switch," *IEEE Commun. Lett.*, vol. 5, pp. 269–271, June 2001.

[6] Rambus, Inc., Los Altos, CA. [Online]. Available: http://www.rambus.com

[7] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE J. Select. Areas Commun.*, vol. 17, pp. 1030–1039, June 1999.

[8] P. Fredette, "The past, present, and future of inverse multiplexing," *IEEE Commun. Mag.*, vol. 32, pp. 42–46, Apr. 1994.

[9] J. Duncanson, "Inverse multiplexing," *IEEE Commun. Mag.*, vol. 32, pp. 34–41, Apr. 1994.

[10] J. Frimmel, "Inverse multiplexing: Tailor made for ATM," *Telephony*, vol. 231, no. 3, pp. 28–34, July 1996.

[11] J. Turner, "Design of a broadcast packet switching network," *IEEE Trans. Commun.*, vol. 36, pp. 734–743, June 1988.

[12] H. Kim and A. Leon-Garcia, "A self-routing multistage switching network for broadband ISDN," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 459–466, Apr. 1990.

[13] I. Widjaja and A. Leon-Garcia, "The helical switch: A multipath ATM switch which preserves cell sequence," *IEEE Trans. Commun.*, vol. 42, pp. 2618–2629, Aug. 1994.

[14] F. Chiussi, D. Khotimsky, and S. Krishnan, "Generalized inverse multiplexing of switched ATM connections," in *Proc. IEEE Globecom*, vol. 5, Sydney, Australia, Nov. 1998, pp. 3134–3140.

[15] F. Chiussi, D. Khotimsky, and S. Krishnan, "Advanced frame recovery in switched connection inverse multiplexing for ATM," in *Proc. ICATM*, Colmar, France, June 1999.

[16] *Interoperability Requirements for Nx56/64 Kbit/s Calls*, ISO/IEC 13 871, 1995.

[17] B. Prabhakar and N. McKeown, "On the speedup required for combined input and output queued switching," *Automatica*, vol. 35, pp. 1909–1920, Dec. 1999.

[18] P. Krishna, N. Patel, A. Charny, and R. Simcoe, "On the speedup required for work-conserving crossbar switches," *IEEE J. Select. Areas Commun.*, vol. 17, pp. 1057–1066, June 1999.

[19] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *J. Internetworking: Research and Experience*, pp. 3–26, 1990.

[20] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Trans. Networking*, vol. 1, pp. 344–357, June 1993.

[21] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," *ACM Trans. Comput. Syst.*, vol. 9, no. 2, pp. 101–124, 1990.

[22] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proc. ACM SIGCOMM*, Sept. 1995, pp. 231–242.

[23] V. E. Benes, *Mathematical Theory of Connecting Network and Telephone Traffic*. New York: Academic, 1965.

[24] J. Bennett and H. Zhang, "WF2Q: Worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 1996, pp. 120–128.

[25] J. Hui, *Switching and Traffic Theory for Integrated Broadband Network*. Boston, MA: Kluwer, 1990.

[26] R. L. Cruz, "A calculus for network delay," *IEEE Trans. Inform. Theory*, vol. 37, pp. 114–131, Jan. 1991.

[27] H. Adiseshu, G. Parulkar, and G. Varghese, "A reliable and scalable striping protocol," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 4, Oct. 1996.

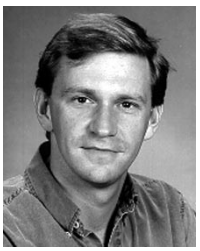
[28] C.-S. Chang, D.-S. Lee, and Y.-S. Jou, "Load balanced Birkhoff–Von Neumann switches—Part I: One-stage buffering," *Comput. Commun.*, vol. 25, pp. 611–622, 2002.

[29] C.-S. Chang, D.-S. Lee, and C.-M. Lien, "Load balanced Birkhoff–Von Neumann switches—Part II: Multi-stage buffering," *Comput. Commun.*, vol. 25, pp. 623–634, 2002.

Sundar Iyer (S'99–A'01) received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Bombay, India, and the M.S. degree in computer science from Stanford University, Stanford, CA, in 1998 and 2000, respectively. He is currently working toward the Ph.D. degree in computer science at Stanford University.

He is currently with the Computer Science Department, Stanford University, where he works in the High-Performance Networking group. From 1999 to 2001, he also was a Senior Systems Architect with SwitchOn Networks (now a part of PMC-Sierra). His research interests include load-balancing algorithms for network design with an emphasis on packet switching, packet buffer design, packet classification, and routing.

Mr. Iyer received the Christofer Stephenson Award for the best Master's thesis in computer science at Stanford University in 2000 and is a recipient of the Siebel Scholars Fellowship. He is a member of the Association for Computing Machinery.



Nick W. McKeown (S'91–M'95–SM'97) received the Ph.D. degree from the University of California, Berkeley, in 1995.

He is currently a Professor of electrical engineering and computer science with Stanford University, Stanford, CA. From 1986 to 1989, he was with Hewlett-Packard Laboratories in their Network and Communications Research group, Bristol, U.K. During the spring of 1995, he was briefly with Cisco Systems, where he helped architect their GSR 12000 router. In 1997, he cofounded Abrizio Inc. (now part of PMC-Sierra), where he was CTO. He is the Robert Noyce Faculty Fellow at Stanford University and a Fellow of the Alfred P. Sloan Foundation. His research interests include the architecture, analysis, and design of high-performance switches and Internet routers, IP lookup and classification algorithms, scheduling algorithms, Internet traffic analysis, traffic modeling and network processors.

Dr. McKeown is the recipient of a CAREER award from the National Science Foundation. In 2000, he received the IEEE Rice Award for the best paper in communications theory. He serves as an Editor for the IEEE TRANSACTIONS ON COMMUNICATIONS and the IEEE/ACM TRANSACTIONS ON NETWORKING. He has served as a Guest Editor for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the *IEEE Network Magazine*, and the *IEEE Communications Magazine*, and is on the Technical Advisory Committee of the ACM SIGCOMM.