Chapter 6: Analyzing Parallel Routers with Slower Memories

Mar 2008, Berkeley, CA

Contents

List of Dependencies

- **Background:** The memory access time problem for routers is described in Chapter 1. Section 1.5.2 describes the use of load balancing techniques, and Section 1.5.3 describes the use of caching techniques; both of which are used in this chapter.

Additional Readings

- **Related Chapters:** The router described in this chapter is an example of a single-buffered router (See Section 2.2), and the general technique to analyze this router is introduced in Section 2.3. The load balancing technique is also used to analyze other router architectures in Chapters 2, 3, 4, and 10.

Table: *List of Symbols.*

c, C	Cell
N	Number of Ports of a Router
k	Number of Center Stage Switches ("Layers")
R	Line Rate
S	Speedup
T	Time slot

Table: *List of Abbreviations.*

FCFS	First Come First Serve (Same as FIFO)
OQ	Output Queued
PIFO	Push In First Out
PPS	Parallel Packet Switch
CPA	Centralized Parallel Packet Switch Algorithm
DPA	Distributed Parallel Packet Switch Algorithm
DWDM	Dense Wavelength Division Multiplexing
WDM	Wavelength Division Multiplexing

"But Parallel Packet Switches are not very practical?"

— The Art of the Airport Security Questionnaire[†]

6

Analyzing Parallel Routers with Slower Memories

6.1 Introduction

Our goal in this thesis was to build high-speed routers that give deterministic performance guarantees as described in Section 1.4. In Chapter 1 we introduced the OQ router. An OQ router gives theoretically ideal performance, is easy to build, and gives deterministic performance guarantees. As we saw, it is, unfortunately, not easy to scale the performance of an OQ router, because the access rate on the memories of an OQ router cannot keep up with increasing line rates. This chapter is motivated by the following question — *Wouldn't it be nice if we could put together many slower-speed OQ routers (which individually give deterministic performance guarantees) and build a high-speed router that can give deterministic performance guarantees?*

✎**Example 6.1.** Figure 6.1 shows an example of a high-speed router (operating at rate R) which is built from k slower-speed routers. The arriving traffic is load-balanced and distributed over k slower speed OQ routers that operate at rate R/k.

[†]En route to Infocom 2000, at the Ben Gurion Airport in Tel Aviv, Israel.

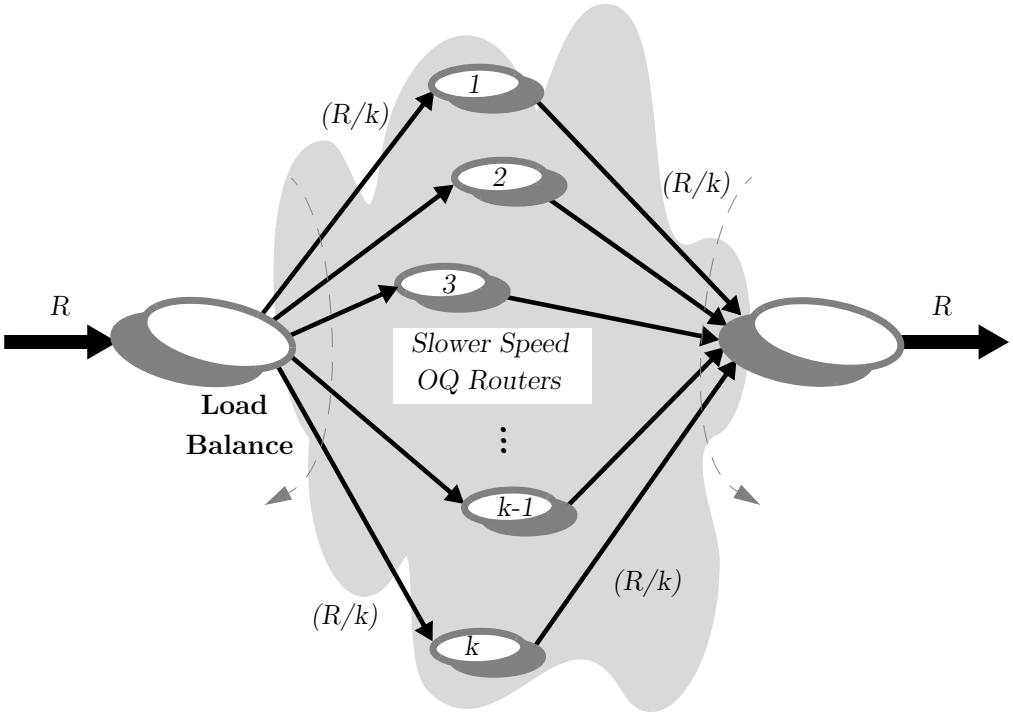

Figure 6.1: *Using system-wide massive parallelism to build high-speed routers.*

6.1.1 Why Do We Need a New Technique to Build High-Speed Routers that Give Deterministic Performance Guarantees?

In the previous chapters, we analyzed various router architectures and described a number of load balancing techniques to alleviate the memory access time problem. The routers that we described were single chassis monolithic routers. They can support very high line rates, and provide deterministic performance guarantees. This begs the question — why do we need a new technique to build another high-speed router that can give deterministic guarantees?

There are three key reasons why we may want to build high-speed routers as described in Figure 6.1:

1. **Rapidly increasing line rates mandate massive parallelism:** We cannot predict the speeds at which line rates will increase. For example, the advent of optical technology such as dense wavelength division multiplexing (DWDM) will provide the capability to rapidly increase line rates. This is because it allows long-haul fiber-optic links to carry very high capacity by enabling a single fiber to contain multiple, separate high-speed channels. Today, channels operate at OC48c (2.5 Gb/s), OC192c (10 Gb/s), and in some systems, OC768c (40 Gb/s). As channel speeds increase beyond OC192 to OC768, and even OC3072 (160 Gb/s), and the number of channels increases, the access rate of the prevailing memory technology may become orders of magnitude slower than the arriving line rate.

   It is not our purpose to argue that line rates will continue to increase – on the contrary, it could be argued that optical DWDM technology will lead to a larger number of logical channels, each operating no faster than, say, 10 Gb/s. We simply make the following observation: *if line rates do increase rapidly, then the memories*[1] *may be so slow in comparison to the line rate that we will be forced to consider architectures where the memory access rate is much slower than the line rate.*
2. **Cost:** Instead of building a monolithic router, it may in fact be cheaper to take the most widely used commodity routers, connect many of them in parallel, and build a higher-speed router. At any given time, given the technological constraints and capabilities, there is always a router capacity beyond which it becomes prohibitively expensive (but still technologically feasible) to build a single-chassis monolithic router; such a router is expected to be cheaper when built in the manner suggested above.

[1]It would be desirable also to process packets in the optical domain, without conversion to electronic form; however, it is not economically feasible today to store packets optically, so for some time to come routers will continue to use electronic memories, which are much slower than current high-speed line rates.

3. **Time to Market:** The development-to-deployment cycle for a typical high-speed router is long: First, new hardware technologies (*e.g.*, fabrication technology, interconnects, memories, boards *etc.*) that a router must use are evaluated and tested. Then the packet processing and other control and data-path ASICs are designed, and they undergo verification. These steps typically take anywhere between 9 and 18 months. Then the ASICs are sent for fabrication, testing, and packaging, and when they are ready (usually another 4-6 months), they are assembled on system boards or line cards. These boards and line cards are then tested individually. The router is then assembled by putting multiple such line cards into a system chassis. Then the router is tested in a "network testbed" among other routers. Finally, after undergoing customer field trials, it is ready to be deployed. For most high-speed routers, the above steps typically take ∼3-4 years.

   In contrast, if we can take a number of slower-speed routers (which are already tested and in deployment), and connect them together in parallel, we may be able to drastically shorten the time to build and deploy such high-speed routers. Of course, in order to do this, the architecture used to connect the slower-speed routers, and the load balancing algorithms used to distribute packets among them, must be made fairly simple to implement.

6.1.2 Why Can't We Use Existing Load Balancing Techniques to Build a Monolithic High-speed Router?

We are in search of router architectures, where the memory runs significantly slower than the line rate. In the previous chapters, we introduced a number of monolithic router architectures and described techniques that can achieve this goal. These techniques load-balanced packets across a number of parallel memories, such that each memory was slower than the line rate. But there are implementation limits to the number of parallel memories that can be interfaced to and load-balanced in a monolithic router.

✎**Example 6.2.** For example, hardware chips today are already limited by interconnect pins. Using current 65nm ASIC fabrication technology, any chip with over 1500 interconnect pins becomes prohibitively expensive. If each memory has approximately 50 pins (fairly typical of most commodity memory today), it is hard for a single ASIC to interface to (and load-balance over) >30 memories.

However, it is possible that line rates will increase so rapidly that we may need to load-balance over hundreds of memories. And so, even the load balancing techniques (that use parallel memories) that we described to alleviate the memory access time problem on the PSM (Section 2.4), DSM (Section 3.2 & 3.3), and PDSM (Section 3.7) router architectures may become impossible to scale to hundreds of memories, and will not give us the massive parallelism we require.

☞**Observation 6.1.** What about other router architectures? We considered CIOQ routers in Chapters 4 and 5. We showed that these routers give performance guarantees, but their memories run at 3X the line rate. Obviously, this doesn't meet our goal that the memories in our router must run at a rate much slower than the line rate R. Even an IQ switch (for which it is known that it cannot give deterministic performance guarantees) requires memories that operate at rate $2R$, which of course does not meet our requirements.

6.1.3 Can We Leverage the Existing Load-Balancing Techniques?

In Figure 6.1, the slower-speed routers (over which we load-balance) were OQ routers. However, we don't really need OQ routers, since when we load-balance across a large number of routers, we treat these routers as "black-boxes". This motivates the following idea —

Idea. *"Since we are not concerned with the internal architectures of the routers that we load-balance over, it will suffice that packets leave these routers at predictable times, ideally at the same time as an OQ router".*

This means that routers that emulate an OQ router are sufficient for our purpose! This means that we can use the more practical router architectures and leverage the load balancing techniques described in Chapters 2-5 such that they can emulate OQ routers.

6.1.4 Goal

So in order to meet our goals, we will explore an architecture that — (1) is practical to build, (2) allows for massive system-level parallelism that enables us to use memories that can run significantly slower than the line rate, and (3) can leverage the techniques we have described in the previous chapters to emulate OQ routers.

6.2 Background

The parallel packet router or switch (PPS)[2] 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 demultiplexor across the slower packet switches, then recombined by a multiplexor at the output. The PPS architecture resembles that of a Clos network [110] as shown in Figure 6.2. The demultiplexor, the center stage packet switches, and the multiplexor can be compared to the three stages of an unbuffered Clos network.

If the center stage switches are OQ, then each packet that passes through the system encounters only a single stage of buffering, making the PPS a single-buffered switch, and it fits our model of SB switches introduced in Chapter 2. As seen by an

[2]We used the terminology "switch" instead of router when we first analyzed this architecture in [35]. So we will continue to use this terminology and refer to this router as the parallel packet switch (PPS).

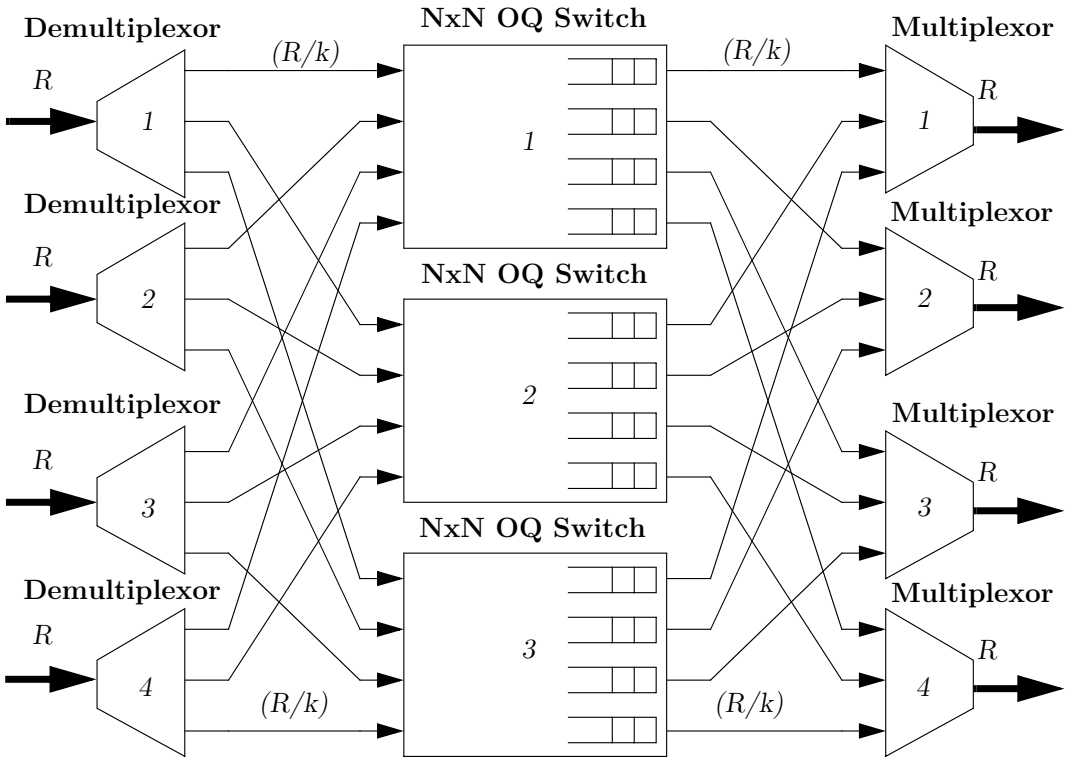

Figure 6.2: *The architecture of a Parallel Packet Switch based on output queued switches. The architecture resembles a Clos network. The demultiplexors, slower-speed packet switches, and multiplexors can be compared to be the three stages of a Clos network.*

arriving packet, all of the buffering is contained in the slower packet switches, and so our first goal is met because no buffers[3] in a PPS need run as fast as the external line rate. The demultiplexor 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 multiplexor that places the packet on the outgoing line. However, the demultiplexor and multiplexor must make intelligent decisions; and as we shall see, the precise nature of the demultiplexing

[3]There will, of course, be small staging buffers in the demultiplexors and multiplexors 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 this chapter.

("spreading") and multiplexing functions are key to the operation of the PPS.

As speed requirements rise, we suspect the PPS architecture is finding wider use in industry. Recent examples that we are aware of include Cisco's highest-speed Internet core router, the CRS-1 [5], and Nevis Networks' enterprise router [111]. We also suspect that some of Juniper's M series core routers (M160 and M320) use a similar architecture [112].

We are interested in the question: Can we select the demultiplexing and multiplexing functions so that a PPS can emulate[4] (Definition 1.2) the behavior of an output queued switch and provide deterministic performance guarantees?

6.2.1 Organization

The rest of the chapter is organized as follows. In Section 6.3 we describe the PPS architecture. In Section 6.4 we introduce some terminology, definitions, and define constraint sets that will help us apply the pigeonhole principle. In Section 6.5, we find the conditions under which the PPS can emulate an FCFS-OQ switch. In Section 6.6, we show how a PPS can emulate an OQ switch with different qualities of service. However, our initial algorithms require a large communication complexity, which makes them impractical. So in Section 6.7 we modify the PPS and allow for a small cache (that must run at the line rate) in the multiplexor and demultiplexor. In Section 6.8 we describe a different distributed algorithm that eliminates the communication complexity and appears to be more practical. In Section 6.9, 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 6.10 and cover related work in 6.11.

[4]Note that 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. So cells in a PPS can never leave at exactly the same time as an OQ switch. Thus a PPS cannot *mimic* (See Section 1.5.4) an OQ router; but as we will see, it can emulate an OQ router.

6.3 The Parallel Packet Switch Architecture

We now focus on the specific type of PPS illustrated in Figure 6.2, 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 output queued switches (we will refer to the center stage switches as "layers"). When a cell arrives at an input port, the demultiplexor selects a layer to send the cell to, and the demultiplexor 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 router with memories operating slower than the rate of the external line. Each of the layers receives 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 multiplexor to become available. When the line is available, the multiplexor selects a cell among the corresponding k output queues in each layer. Since each multiplexor receives cells from 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 muliplexor or the demultiplexor contains any memory, and that they are the only components running at rate R. We can compare the memory bandwidth requirements of an $N \times N$ parallel packet switch with those of an OQ switch with the same aggregate bandwidth. In an OQ switch (refer to Table 2.1), 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 further reduce the memory bandwidth by leveraging any of the load balancing and scheduling algorithms that we introduced for the PSM, DSM, PDSM, CIOQ, or buffered CIOQ router architectures described in previous chapters.

✎**Example 6.3.** As an example, we can use a CIOQ router in the center stage. From Chapter 4, we know that an OQ switch can be emulated 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 k, the number of layers.

Choosing the value of k. Our goal is to design switches in which all the memories run slower than the line rate. If the center stage switches are CIOQ routers, 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 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. The following example makes this clear.

✎**Example 6.4.** Consider a router with $N = 1024$ ports, $R = 40$ Gb/s, and cells 64 bytes long. Then a PPS with $k = 100$ center stage CIOQ routers can be built such that the fastest memories run at a speed no greater than $3R/k = 1.2$ Gb/s. For a 64-byte cell this corresponds to an access time of 426 ns, which is well within the random access time of commercial DRAMs.

6.3.1 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 counter-example, without speedup a PPS is not work-conserving, and hence cannot emulate an OQ switch.

Theorem 6.1. *A PPS without speedup is not work-conserving.*

Proof. (By counter-example). See Appendix F.1. ❒

ℵ**Definition 6.1. Concentration:** Concentration occurs 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 multiplexor. Unfortunately, the counter-example in Appendix F.1 shows that concentration is unavoidable in our current PPS architecture. 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. Note that the N memories in any OQ switch in the center stage run at an aggregate rate of $N(N+1)R'$, where $R' = SR/k$. So the total memory bandwidth in the PPS when it is sped up is $k \times N(N+1)R' = SN(N+1)R$.

☞**Observation 6.2.** Concentration can be eliminated by running the internal links at a rate 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. But this appears to defeat the purpose of operating the internal layers slower than the external line rate! Fortunately, we will see in the next section that the speedup required to eliminate the problem of concentration is independent of the arriving traffic, as well as R and N, and is almost independent of k. In particular, we find that with a speedup of 2, the PPS is work-conserving and can emulate an FCFS-OQ switch.

6.4 Applying the Pigeonhole Principle to the PPS

We will now apply the pigeonhole principle introduced in Chapter 2 to analyze the PPS. First, however, we need to carefully distinguish the operations in time of the external demultiplexor and multiplexor (which run faster), and the internal OQ switches (which run slower). So we will define two separate units of time to distinguish between them:

ℵ**Definition 6.2. Time slot:** This refers to the time taken to transmit or receive a fixed-length cell at a link rate of R.

ℵ**Definition 6.3. 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.

6.4.1 Defining Constraint Sets

We are now ready to define constraint sets for the PPS as described in Algorithm 2.1. 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.

ℵ**Definition 6.4. 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, or ILC.

ℵ**Definition 6.5. Allowable Input Link Set** – The ILC gives rise to the allowable input link set, $AIL(i,n)$, 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 to which external input i has not started sending any cells within the last $\lceil k/S \rceil - 1$ time slots. Note that $|AIL(i,n)| \leqslant k, \forall (i,n)$.

$AIL(i,n)$ 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 $AIL(i,n)$. The layer is added back to the set when it becomes free at time $n + \lceil k/S \rceil$.

ℵ**Definition 6.6. 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, or OLC.

ℵ**Definition 6.7. Departure Time** – When a cell arrives, the demultiplexor 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 $DT(n, i, j)$. The departure time for a FIFO queuing policy 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 6.6, other definitions are possible in the case of WFQ policies.

ℵ**Definition 6.8. Available Output Link Set** – The OLC gives rise to the available output link set $AOL(j, DT(n, i, j))$, which is the set of layers that can send a cell to external output j at time slot $DT(n, i, j)$ in the future. $AOL(j, DT(n, i, j))$ 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 $DT(n, i, j)$. Note that, since there are a total of k layers, $|AOL(j, DT(n, i, j))| \leqslant k, \forall (j, DT(n, i, j))$.

Like $AIL(i, n)$, $AOL(j, DT(n, i, j))$ 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 $DT(n, i, j)$, the layer is deleted from $AOL(j, DT(n, i, j))$ and then will be added to the set again when the layer becomes free at time $DT(n, i, j) + \lceil k/S \rceil$. However, whenever a layer is deleted from the set, the index $DT(n, i, j)$ 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 $DT(n, i, j)$ may change up to N times per time slot. This is because $AOL(j, DT(n, i, j))$ represents the layers available for use at some time $DT(n, i, j)$ 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, $AOL(j, DT(n, i, j))$ indicates the schedule of future departures for output j, and at any instant, $\max(DT(n, i, j) + 1, \forall (n, i))$ indicates the first time in the future that output j will be free.

6.4.2 Lower Bounds on the Size of the Constraint Sets

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

Lemma 6.1. *The size of the available input link set, for all $i, n \geqslant 0$; where is S the speedup on the internal input links is given by,*

$$|AIL(i,n)| \geqslant k - \lceil k/S \rceil + 1. \tag{6.1}$$

Proof. Consider external input port i. The only layers that i cannot send a cell to 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). $|AIL(i,n)|$ is minimized when a cell arrives to the external input port in each of the previous $\lceil k/S \rceil - 1$ time slots, hence $|AIL(i,n)| \geqslant k - (\lceil k/S \rceil - 1) = k - \lceil k/S \rceil + 1)$. ❐

Lemma 6.2. *The size of the available output link set, for all $i, j, n \geqslant 0$; where is S the speedup on the internal input links is given by,*

$$|AOL(j, DT(n,i,j))| \geqslant k - (\lceil k/S \rceil + 1). \tag{6.2}$$

Proof. The proof is similar to Lemma 6.1. We consider an external output port that reads cells from the internal switches instead of an external input port that writes cells to the internal switches. ❐

6.5 Emulating an FCFS-OQ Router

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, $DT(n,i,j)$, is simply the first time that output j is free (in the shadow FCFS-OQ switch) and able to send a cell. We will now introduce an algorithm, Centralized Parallel Packet Switch Algorithm (CPA). CPA is directly motivated by the pigeonhole principle, and attempts to route cells to center stage switches as described below.

Algorithm 6.1: CPA for FCFS-OQ emulation.

input : Arrival and departure times of each cell.
output : A central stage switch for each cell to emulate an FCFS policy.

for *each cell* C **do**
  Demultiplexor:
    When a cell arrives at time n at input i destined to output j, the cell is sent to any center stage switch, l, that belongs to the intersection of $AIL(i,n)$ and $AOL(j, DT(n,i,j))$.
  Multiplexor:
    Read cell C from center stage switch l at departure time $DT(n,i,j)$.

✎**Example 6.5.** A detailed example of the CPA algorithm appears in Appendix A of [113].

6.5.1 Conditions for a PPS to Emulate an FCFS-OQ Router

We will now derive the conditions under which CPA can always find a layer l to route arriving cells to, and then analyze the conditions under which it can emulate an FCFS-OQ router.

Lemma 6.3. *(Sufficiency) A speedup of 2 is sufficient for a PPS to meet both the input and output link constraints for every cell.*

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

$$AIL(i,n) \cap AOL(j, DT(n,i,j)) \neq \varnothing. \tag{6.3}$$

We know that Equation 6.3 is satisfied if,

$$|AIL(i,n)| + |AOL(j, DT(n,i,j))| > k. \tag{6.4}$$

From Lemma 6.1 and Lemma 6.2 we know that, $|AIL(i,n)| + |AOL(j, DT(n,i,j))| > k$ if $S \geqslant 2$ ❒

Theorem 6.2. *(Sufficiency) A PPS can emulate an FCFS-OQ switch with a speedup of* $S \geqslant 2$.[5]

Proof. Consider a PPS with a speedup of $S \geqslant 2$. From Lemma 6.3 we know that for each arriving cell, the demultiplexor can select a layer that meets both the ILC and the OLC in accordance with the CPA algorithm. A cell destined to output j and arriving at time slot n is scheduled to depart at time slot $DT(n,i,j)$, which is the index of $AOL(j, DT(n,i,j))$. By definition, $DT(n,i,j)$ 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 $2R/k$, $DT(n,i,j)$ is equal to the time that the cell would depart in an FCFS-OQ switch. Hence a PPS can emulate an FCFS-OQ switch. ❒

☞**Observation 6.3.** It is interesting to compare the above proof with the requirements for a 3-stage symmetrical Clos network to be strictly non-blocking [114, 115]. 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 similar, and relies on the pigeonhole principle.

6.6 Providing QoS Guarantees

We now extend our results to find the speedup requirement for a PPS to provide QoS guarantees. To do this, we define constraint sets, and find the speedup required for a

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

✂Box 6.1: A Work-conserving PPS Router✂

We can ask an identical question to the one posed in the previous chapters: What are the conditions under which a buffered CIOQ router is work-conserving *with and without* mandating FCFS-OQ emulation? Because we need to keep the outputs of a work-conserving router busy, we start by ensuring that any cell that arrives at time n, leaves at the first future time the shadow OQ switch is free, at time slot n. We denote this time as $DT(n,i,j)$.

Lemma 6.4. *(Sufficiency) If a PPS guarantees that each arriving cell is allocated to a layer l, such that $l \in AIL(i,n)$ and $l \in AOL(j, DT(n,i,j))$, then the switch is work-conserving.*

Proof. Consider a cell C that arrives to external input port i at time slot n and destined for output port j. The demultiplexor chooses a layer l that meets both the ILC and the OLC; *i.e.*, $l \in \{AIL(i,n) \cap AOL(j, DT(n,i,j))\}$. Since, the ILC is met, cell C can be immediately written to layer l in the PPS. 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 $DT(n,i,j)$ has already been picked when it arrived at time n, C is removed from its queue at time $DT(n,i,j)$ and sent to external output port j. Cell C can depart at $DT(n,i,j)$ because the link from multiplexor j to layer l satisfies, $l \in AOL(j, DT(n,i,j))$. Thus, if for cell C the chosen layer l meets both the ILC and OLC, then the cell can reach switch l, and can leave the PPS at time $DT(n,i,j)$.

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. The output is continuously kept busy if there are cells destined for it, similar to the output of the shadow OQ switch. And so, the PPS is work-conserving. ❒

As a consequence of Lemma 6.4 and Lemma 6.3, we have the following theorem:

Theorem 6.3. *(Digression) A PPS can be work-conserving if $S \geqslant 2$.*

To achieve work conservation in Theorem 6.3, we required that cell C leave at time $DT(n,i,j)$. While this is enough for work conservation, this requirement actually results in FCFS-OQ emulation! If we did not want FCFS-OQ emulation, the PPS is still work-conserving as long as *any cell* leaves the multiplexor for output j at time $DT(n,i,j)$.

It is possible to permute the order in which cells are read by the multiplexor. Each permutation by the multiplexor could give greater choice to the demultiplexor to choose a center stage OQ switch (depending on the last few cells that the multiplexor reads in the permutation) when it routes newly arriving cells. So there may be different algorithms which can permute the packet order, and the PPS may be work-conserving with a lower speedup than what is derived in Theorem 6.3.[a]

[a] A direct analogy to this result is the work on re-arrangeably non-blocking Clos networks [116].

PPS to implement any PIFO scheduling discipline. As we will see, we will need to modify our CPA algorithm.

In Section 1.4.2, we saw that a PIFO queuing policy can insert a cell anywhere in its 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 and destined to output port j. The demultiplexor determines the time that each arriving cell must depart, $DT(n, i, j)$, to meet its delay guarantee. The decision made by the demultiplexor at input i amounts to selecting a layer so that the cell may depart on time. Notice that this is very similar to the previous section, 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 $DT(n, i, j)$ may be selected to be ahead of cells already scheduled to depart from output j. So, the demultiplexor's choice of sending an arriving cell C to layer l must now meet three constraints:

1. The link connecting the demultiplexor at input i to layer l must be free at time slot n. Hence, $l \in \{AIL(i, n)\}$.

2. The link connecting layer l to output j must be free at $DT(n, i, j)$. Hence, $l \in \{AOL(j, DT(n, i, j))\}$.

3. All the other cells destined to output j after C must also find a link available. In other words, if the demultiplexor 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 $(DT(n, i, j), DT(n, i, j) + \lceil k/S \rceil - 1)$, 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 $l \in \{AOL < (j, DT(n, i, j) + \lceil k/S \rceil - 1)\}$.

The following natural questions arise:

1. *What if some of the cells that 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 $AOL(j, DT(n, i, j) + \lceil k/S \rceil - 1)$ will allow more layers.[6] Note that cell C need not bother about the cells that 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. 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 multiplexor 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 effect it has is that it can conflict with the $\lceil k/S \rceil - 1$ cells that 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 interval 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 departure 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.

We are now ready to summarize our modified CPA algorithm to emulate a PIFO-OQ router as shown in Algorithm 6.2.

Theorem 6.4. *(Sufficiency) A PPS can emulate any OQ switch with a PIFO queuing discipline, with a speedup of* $S \geqslant 3$.[7]

[6] 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, $AOL(j, DT(n, i, j) + \lceil k/S \rceil - 1)$ defined at time t, will include all the k layers, and so the constraint disappears, leaving us with just two of the three conditions above, as for FCFS-OQ in Section 6.5.

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

Algorithm 6.2: Modified CPA for PIFO emulation on a PPS.

input : Arrival and departure times of each cell.
output: A central stage switch for each cell to emulate a PIFO policy.

for *each cell* C **do**
  Demultiplexor:
    When a cell arrives at time n at input i destined to output j, the cell is sent to any center stage switch that belongs to the intersection of $AIL(i,n)$, $AOL(j,DT(n,i,j))$, and $AOL(j,DT(n,i,j)+\lceil k/S\rceil-1)$
  . Multiplexor:
    Read cell C from center stage switch l *whenever it reaches head of line.*

Proof. In order for our modified CPA algorithm to support a PIFO queuing discipline, we require layer l to satisfy

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

For a layer l to exist we require

$$AIL(i,n) \cap AOL(j,DT(n,i,j)) \cap AOL(j,DT(n,i,j)+\lceil k/S\rceil-1) \neq \varnothing,$$

which is satisfied when

$$|AIL(i,n)| + |AOL(j,DT(n,i,j))| + |AOL(j,DT(n,i,j)+\lceil k/S\rceil-1)| > 2k.$$

From Lemma 6.1 and 6.2 we know that

$$|AIL(i,n)| + |AOL(j,DT(n,i,j))| + |AOL(j,DT(n,i,j)+\lceil k/S\rceil-1)| > 2k,$$

if $S \geqslant 3$. ❐

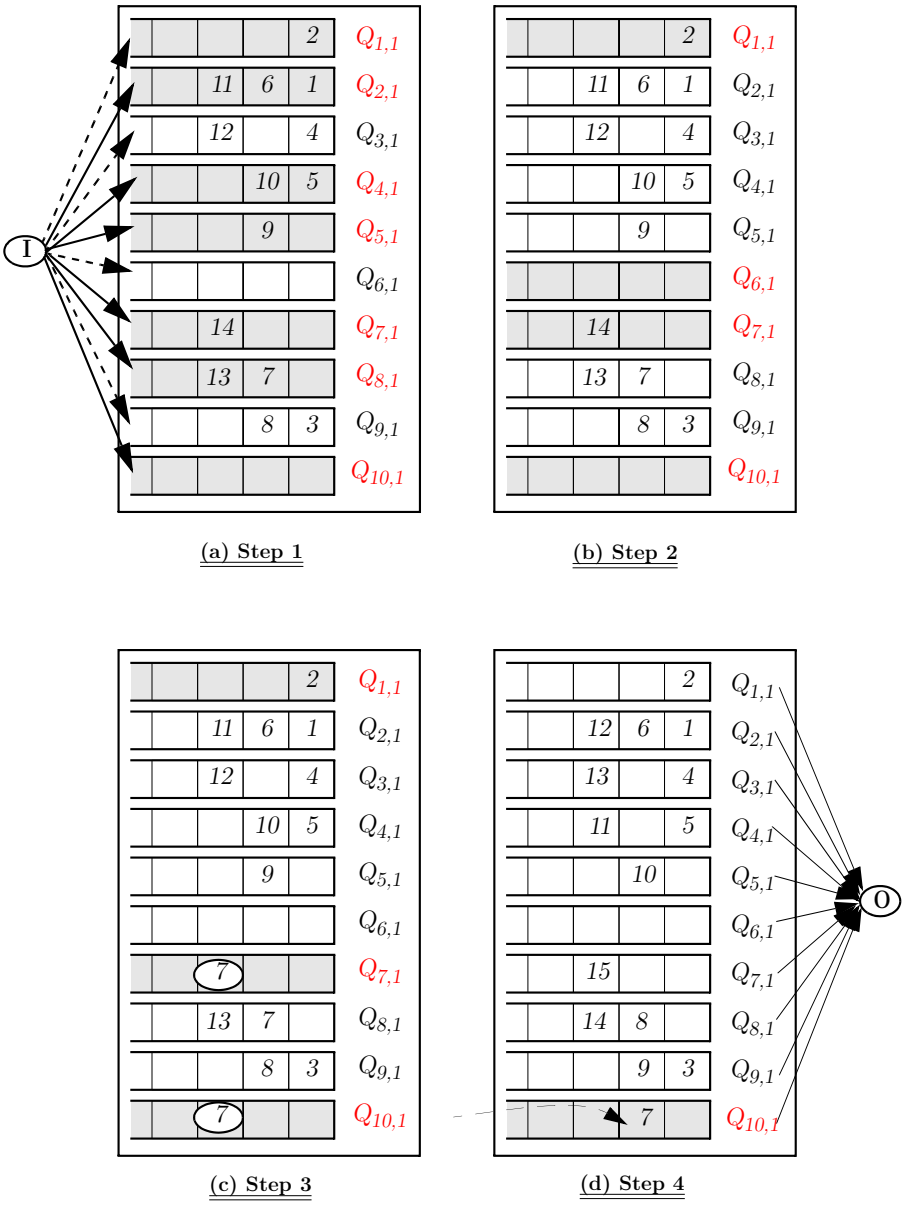

Figure 6.3: *Insertion of cells in a PIFO order in a PPS with ten layers. $Q_{k,1}$ refers to output queue number one in the internal switch k. The shaded layers describe the sets specified for each figure. (a) The AIL constrains the use of layers $\{1, 2, 4, 5, 7, 8, 10\}$. (b) The cell is to be inserted before cell number 7. The two AOLs constrain the use of layers $\{1, 6, 7, 10\}$. (c) The intersection constrains the use of layers $\{1, 7, 10\}$. (d) Layer 10 is chosen. The cell number 7 is inserted.*

✎**Example 6.6.** Figure 6.3 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 Figure 6.3(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. Similarly, cell C cannot use any layers to which the 3 cells after $C7$ including $C7$ (*i.e.*, $C7$, $C8$, $C9$) were sent. 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. Figure 6.3(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. Figure 6.3(c) shows the candidate layers for insertion *i.e.*, layers 1, 7, and 10. Cell C is then inserted in layer 10 as shown in Figure 6.3(d).

6.7 Analyzing the Buffered PPS Router

Unfortunately, the load balancing algorithms that we have described up until now are complex to implement. In Chapter 4 we faced a similar problem with the CIOQ router. We showed in Chapter 6 that we could simplify these algorithms with the introduction of a cache. In the rest of the chapter we answer the following question: *Can a cache (preferably small in size!) simplify the centralized load balancing algorithms for a PPS?*

6.7.1 Limitations of Centralized Approach

The centralized approach described above suffers from two main problems:

✂Box 6.2: A Discussion on Work Conservation✂

We introduced different router architectures in the previous chapters, *i.e.*, the PSM, DSM, PDSM, and the time reservation-based CIOQ router. We can ask a general question: What are the conditions under which these routers achieve work-conservation (without mandating FCFS-OQ emulation)? Before we answer this, note that these routers share a common trait — *i.e.*, they were all analyzed using the basic constraint set technique, because they all attempt to assign a packet to memory immediately on arrival.

☞**Observation 6.4.** The basic constraint set technique and the extended constraint set technique fundamentally differ in the way they schedule packets. In the latter case, packets are held back and transferred to the output memory based on their priority. In the former case, the scheduler attempts to assign packets to a memory immediately on arrival.[a] As we will see, this has repercussions to work-conservation as described below.

A scheduler which uses the extended constraint set technique can pick *any* cell (which is still awaiting service at the input) to transfer to an output in future, since in a work-conserving router, all cells destined to an output have equal priority. In Chapters 4 and 5 we showed that for routers that do not assign a packet to memory immediately (and which are analyzed using the extended pigeon hole principle), requiring work-conservation without FCFS-OQ emulation results in a simplified switch scheduler. Since this was a useful goal, we modified the extended constraint set technique (Algorithm 4.2) to create a simplified technique (Algorithm 4.3) to analyze work-conservation.

Unfortunately, for a scheduler that uses the basic constraint set technique, all previously arrived packets have already been allocated to memories. The only way that the scheduler can take advantage of the relaxed requirements of work-conservation is to permute the existing order of these cells which are already buffered (as described in Box 6.1). This has two problems — (1) it is impractical to compute permutations for existing packets (there can be a very large number of possible permutations) destined to an output, and (2) computing such a permutation only makes the scheduler more complex, not simpler!

So work-conservation is only of theoretical interest for routers that attempt to assign packets to memories immediately on arrival, *i.e.*, the routers that are analyzed using the basic constraint set technique. This, of course, includes the PPS, as well as the PSM, DSM, PDSM, and the time reservation-based CIOQ router, which were described in previous chapters. And so, in contrast to the extended constraint set technique, we do *not* modify the basic constraint set technique (Algorithm 2.1) to analyze work-conserving SB routers.

[a] In the case of a time reservation-based CIOQ router, the assignment is done immediately, though the packet is actually transferred at a later time.

1. **Communication complexity:** The centralized approach requires each input to contact a centralized scheduler at every arbitration cycle. With N ports, N requests must be communicated to and processed by the arbiter during 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 multiplexor) be conveyed to each multiplexor and stored.
2. **Speedup:** The centralized approach requires a speedup of two (for an FCFS PPS) in the center stage switches. The PPS therefore over-provisions the required capacity by a factor of two, and the links are on average only 50% utilized. This gets worse for a PPS that supports qualities of service, where a speedup of three implies that the links, on average, are only 33% utilized.

☞**Observation 6.5.** 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 multiplexor 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 pre-existing 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 multiplexors and demultiplexors, and a distributed algorithm, that:

[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.

1. Enables the demultiplexors and multiplexors 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, and
4. Allows a feed-forward data-path in which each layer may be constructed from pre-existing, "standard" output queued switches.

6.8 A Distributed Algorithm to Emulate an FCFS-OQ Router

The goals outlined in the previous subsection naturally lead to the following modifications:

1. **Distributed decisions**. A demultiplexor decides which center stage switch to send a cell to, based only on the knowledge of cells that have arrived at its input. The demultiplexors do not know the $AOL(.)$ sets, and so have no knowledge of the distribution of cells in the center stage switches for a given output. Hence a demultiplexor 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 that arrive at the demultiplexor for every output equally among all center stage switches. Given that we also wish to spread traffic uniformly across the center stage switches, each demultiplexor 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 ("Cache") operating at the line rate**. If the demultiplexors 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 demultiplexor that can cache the cells temporarily before sending them to the center stage switches. Similarly, it is possible for multiple independent demultiplexors to choose the

same center stage switch for cells destined to the same output. This causes concentration, and cells can become mis-sequenced. The order of packets can be restored by the addition of a similar coordination buffer in each multiplexor to re-sequence the cells (and cache earlier-arriving cells) before transmitting them on the external line.

We will see that the coordination buffer caches are quite small, and are the same size for both the multiplexor and demultiplexor. More important, they help to eliminate the need for speedup. The co-ordination buffer operates at the line rate, R, and thus 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. Because 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 with no relative queuing delay. However, we will show that the PPS can emulate an FCFS-OQ switch within a small relative queuing delay bound.

6.8.1 Introduction of Caches to the PPS

Figure 6.4 shows the introduction of small caches to the PPS. It describes how coordination buffers are arranged in each demultiplexor as multiple equal-sized FIFOs, one per layer. FIFO $Q(i,l)$ holds cells at demultiplexor i destined for layer l. When a cell arrives, the demultiplexor 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 caches in each multiplexor are arranged the same way, and so FIFO $Q'(j,l)$ holds cells at multiplexor 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,

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

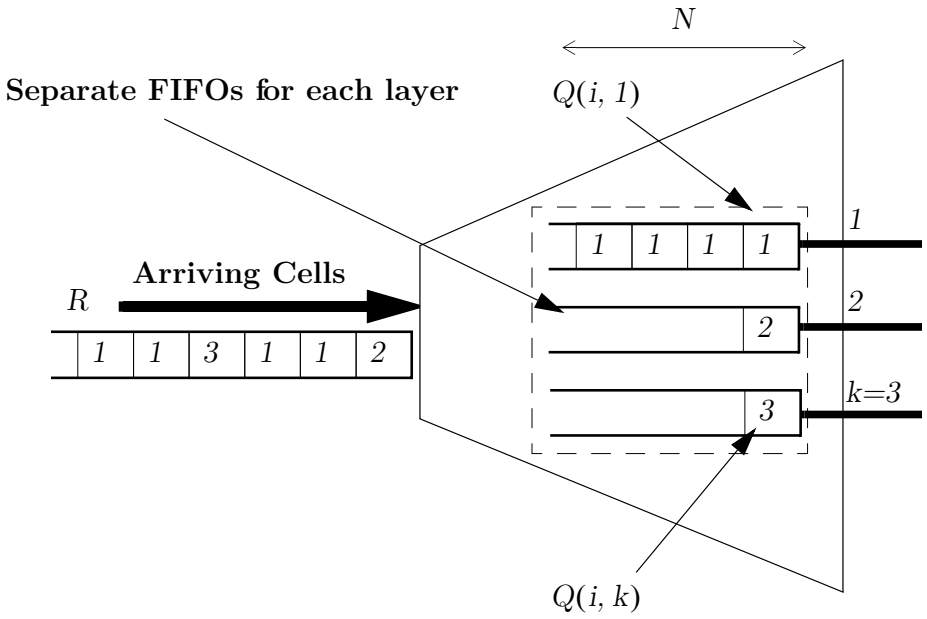

Figure 6.4: *The demultiplexor, showing k FIFOs, one for each layer, and each FIFO of length d cells. The example PPS has $k = 3$ layers.*

then the coordination buffer cache can hold a total of kd cells.

6.8.2 The Modified PPS Dispatch Algorithm

The modified PPS algorithm proceeds in three distinct parts.

1. **Split every flow in a round-robin manner in the demultiplexor:** Demultiplexor 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 waits until its turn to be delivered to layer l. When the link from demultiplexor i to layer l is free, the head-of-line cell (if any) of $Q(i, l)$ is sent.
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) above introduced a complication that we must deal with: cells reaching the center stage have already encountered a variable queuing delay in the demultiplexor 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 this queuing delay is variable, it 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 demultiplexor, and hence equalize the delays. Though not strictly required, we do this at the input of the center stage switch. Each cell records how long it was queued in the demultiplexor, and then the center 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 6.9.

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 demultiplexor. When the cell reaches the head of the output queues of the center stage switch, it is sent to the output multiplexor when the link is next free.

3. **Re-ordering the cells in the multiplexor:** The co-ordination buffer in the multiplexor stores cells, where they are re-sequenced and then transmitted in the correct order.

The load balancing algorithm on the de-multiplexor that results as a consequence of this modification is summarized in Algorithm 6.3.

Algorithm 6.3: DPA for FCFS-OQ emulation on a PPS.

1 **input** : Arrival and departure times of each cell.
2 **output** : A central stage switch for each cell to emulate an FCFS-OQ policy.
3 **for** *each cell* C **do**
4 Demultiplexor:
5 When a cell arrives at time n at input i destined to output j, the cell is sent to the center stage switch that is next in the round robin sequence maintained for the input-output pair (i, j).
6 Multiplexor:
7 Read cell C from center stage switch l at departure time $DT(n, i, j)$ *after delay equalization*.

☞**Observation 6.6.** It is interesting to compare this technique with the load-balanced switch proposed by Chang et al. in [91]. In their scheme, load balancing is performed by maintaining a single round robin list at the inputs (*i.e.*, demultiplexors) for a 2-stage switch. The authors show that this leads to guaranteed throughput and low average delays, although packets can be mis-sequenced. In [92], 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 mis-sequencing in the PPS and also gives a delay guarantee for each packet.

6.9 Emulating an FCFS-OQ Switch with a Distributed Algorithm

Theorem 6.5. *(Sufficiency) A PPS with independent demultiplexors and multiplexors and no speedup, with each multiplexor and demultiplexor containing a co-ordination*

buffer cache of size Nk cells, can emulate an FCFS-OQ switch with a relative queuing delay bound of $2N$ internal time slots.

Proof. The complete proof is in Appendix F.2. We describe the outline of the proof here. Consider the path of a cell in the PPS where it may potentially face a queuing delay:

1. The cell may be queued at the FIFO of the demultiplexor before it is sent to its center stage switch. From Theorem F.1 in Appendix F, 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 F.2 in Appendix F, this is bounded by N internal time slots.

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

6.10 Implementation Issues

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

1. **Demultiplexor:** Each demultiplexor maintains a buffer cache 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 cache needs to be, and whether the cache can be placed on-chip. The demultiplexor must add a tag to each cell indicating the arrival time of the cell to the demultiplexor. Apart from that, no sequence numbers need to be maintained at the inputs or added to cells.

✎**Example 6.7.** If $N = 1024$ ports, cells are 64-bytes long, $k = 100$, and the center stage switches are CIOQ routers, then the cache size for the coordination buffer is about 50 Mbits per multiplexor and demultiplexor. This can be (just) placed on-chip using today's SRAM technology, and so can be made both fast and wide. Embedded DRAM memory technology, which offers double density as compared to SRAM (but roughly half the access rate), could also potentially be used [26]. However, for much larger N or k this approach may not be practicable.

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 demultiplexor'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 $1 \leqslant D_i \leqslant N$, to compensate for its variable delay in the demultiplexor. After the cell has been delayed, it can be placed directly into the center stage switch's output queue.

3. **Multiplexors:** Each multiplexor maintains a coordination buffer of size Nk running at the line rate R. The multiplexor re-orders cells based upon the arrival timestamp. Note that if the 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 [117].

☞**Observation 6.7.** We note that if a cell is dropped by a center stage switch, then the multiplexors cannot detect the lost cell in the absence of sequence numbers. This would cause the multiplexors to re-sequence cells incorrectly. A solution to this is to mandate the center stage switches to make the multiplexors aware of dropped cells by transmitting the headers of all dropped cells.

6.11 Related Work

Although the specific PPS architecture seems novel, "load balancing" and "inverse-multiplexing" systems [118, 119, 120] 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 re-synchronize cells sent through parallel switches or links [121, 122, 123, 124, 125, 126]. However, we are not aware of any analytical studies of the PPS architecture prior to this work. As we saw, there is an interesting and simple analogy between the (buffered) PPS architecture and Clos's seminal work on the (unbuffered) Clos network [110].

6.11.1 Subsequent Work

In subsequent work, Attiya and Hay [127, 128] perform a detailed analysis of the distributed PPS and prove lower bounds on the relative delay faced by cells in the PPS for various demultiplexor algorithms. Saad et al. [129] consider a PPS with a large link speed of KNR in the center stage switches, and show that a PPS can emulate an OQ switch with low communication complexity. Attiya and Hay [128] also study a randomized version of the PPS.

6.12 Conclusions

In this chapter, we set out to answer a simple but fundamental question about high-speed routers: *Is it possible to build a high-speed switch (router) from multiple slower-speed switches and yet give deterministic performance guarantees?*

The PPS achieves this goal by exploiting system-level parallelism, and placing multiple packet switches in parallel, rather than in series as is common in multistage switch designs. More interestingly, the memories in the center stage switches can operate much slower than the line rate. Further, the techniques that we used to build high-speed switches that emulate OQ switches (Chapters 2- 5) can be leveraged to build the center stage switches.

Our results are as follows: A PPS with a centralized scheduler can emulate an PIFO-OQ switch — *i.e.*, it can provide different qualities of service; although the implementation of such a PPS does not yet seem practical. So we adopted a distributed approach, and showed 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).

We are aware of the use of distributed algorithms on a buffered PPS fabric in current commercial Enterprise routers [130, 111]. We suspect that Cisco's CRS-1 [5] and Juniper's M series [112] Internet routers also deploy some variant of distributed algorithms to switch packets across their PPS-like fabrics. These implementations show that high-speed PPS fabrics are practical to build. At the time of writing, switch fabrics similar to the PPS are also being considered for deployment in the next generation of the low-latency Ethernet market [131] as well as high-bandwidth Ethernet "fat tree" [132] networks.

In summary, we think of this work as a practical step toward building extremely high-capacity FIFO switches, where the memory access rates can be orders of magnitude slower than the line rate.

Summary

1. This chapter is motivated by the desire to build routers with extremely large aggregate capacity and fast line rates.
2. We consider building a high-speed router using system-level parallelism — *i.e.*, 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.
3. 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.
4. *Why do we need a new technique to build high-performance routers?* There are three

reasons why we may need system-level parallelism — (1) the memories may be so slow in comparison to the line rate that we may need the massive system parallelism that the PPS architecture provides, as there are limits to the degree of parallelism that can be implemented in the monolithic router architectures seen in the previous chapters; (2) it may in fact be cheaper to build a router from multiple slower-speed commodity routers; and (3) it can reduce the time to market to build such a router.

5. It is a goal of our work that all memory buffers in the PPS run slower than the line rate. Of course, we are interested in the conditions under which a PPS can emulate an FCFS-OQ router and an OQ router that supports qualities of service.
6. In this chapter, we ask the question: Is it possible for a PPS to precisely emulate the behavior of an output queued router with the same capacity and with the same number of ports?
7. We show that it is theoretically possible for a PPS to emulate an FCFS output queued (OQ) packet switch if each lower-speed packet switch operates at a rate of approximately $2R/k$ (Theorem 6.2).
8. We further show that it is theoretically possible for a PPS to emulate an OQ router that supports qualities of service if each lower-speed packet switch operates at a rate of approximately $3R/k$ (Theorem 6.4).
9. 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 "co-ordination buffer" running at the line rate in both the demultiplexor and the multiplexor.
10. We determine the size of this buffer to be Nk bytes (where N is the number of ports in the PPS, and k is the number of center stage OQ switches), and show that it can eliminate the need for a centralized scheduling algorithm, allowing a full distributed implementation with low computational and communication complexity.
11. 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 (Theorem 6.5).